RELATÓRIO DE CONCLUSÃO DA META 4

Implementação do Observatório do Sistema Nacional do Patrimônio Cultural

ibict
Instituto Brasileiro de Informação em Ciência e Tecnologia

PRESIDÊNCIA DA REPÚBLICA

Luiz Inácio Lula da Silva
Presidente da República

Geraldo José Rodrigues Alckmin Filho
Vice-Presidente da República

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

Luciana Santos
Ministra da Ciência, Tecnologia e Inovação

INSTITUTO BRASILEIRO DE INFORMAÇÃO EM CIÊNCIA E TECNOLOGIA

Tiago Emmanuel Nunes Braga
Diretor

Carlos André Amaral de Freitas
Coordenador de Administração - COADM

Ricardo Medeiros Pimenta
Coordenador de Ensino e Pesquisa em Informação para a Ciência e Tecnologia - COEPI

Henrique Denes Hilgenberg Fernandes
Coordenador de Planejamento, Acompanhamento e Avaliação - COPAV

Cecília Leite Oliveira
Coordenadora Geral de Informação Tecnológica e Informação para a Sociedade - CGIT

Washington Luís Ribeiro de Carvalho Segundo
Coordenador Geral de Informação Científica e Técnica - CGIC

Alexandre Faria de Oliveira
Coordenador Geral de Tecnologias de Informação e Informática - CGTI

Milton Shintaku
Coordenador de Tecnologias para Informação - COTEC

RELATÓRIO DE CONCLUSÃO DA META 4

Implementação do Observatório Nacional do Patrimônio Cultural

Gustavo Cardoso Paiva
Danielle do Carmo
Mirele Carolina Souza Ferreira Costa
Milton Shintaku

Brasília
2025

EQUIPE TÉCNICA

Diretor do Instituto Brasileiro de Informação em Ciência e Tecnologia
Tiago Emmanuel Nunes Braga

Coordenador-Geral de Tecnologias de Informação e Informática – CGTI
Alexandre Faria de Oliveira - substituto

Coordenador do Projeto
Milton Shintaku

Elaborado por
Gustavo Cardoso Paiva
Danielle do Carmo
Mirele Carolina Souza Ferreira Costa
Milton Shintaku

Normalização
Elton Mártires Pinto

Pesquisadores do projeto
André José Ribeiro Guimarães, Clara Andreozzi de La Rocque Couto, Danielle do Carmo, Elton Mártires Pinto, Fernando de Jesus Pereira, Favia Karla Ribeiro Santos, Gustavo Cardoso Paiva, Lucas Rodrigues Costa, Lucas Ângelo da Silveira, Maison Roberto Mendonça Gonçalves, Mateus Machado Luna, Mirele Carolina Souza Ferreira Costa, Nuielle Cristine de Medeiros da Silva, Rafael Fernandez Gomes, Rafael Teixeira de Souza, Raíssa da Veiga de Menêses.

Este Relatório de Técnico é um produto do Projeto: Estudos voltados ao desenvolvimento de Modelo para criação e plataforma de Gestão da Informação no Instituto do Patrimônio Histórico e Artístico Nacional (Iphan).

Ref. Ibict - Processo SEI: 01302.000684/2023-11 (Execução)
Ref. Ibict - Processo SEI: 01302.000339/2023-79
Ref. FUNDEP - 30985

LISTA DE QUADROS

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O Instituto do Patrimônio Histórico e Artístico Nacional (Iphan) e o Instituto Brasileiro de Informação em Ciência e Tecnologia (Ibict) firmaram parceria para desenvolver um ecossistema informacional voltado à gestão estratégica do patrimônio cultural, atendendo às diretrizes de transparência ativa e democratização do acesso à informação.

A partir dessa parceria, surgiu o projeto "Estudos voltados a um Observatório do Sistema Nacional do Patrimônio Cultural Brasileiro", cujo objetivo principal é consolidar o Observatório do Sistema Nacional do Patrimônio Cultural (SNPC) como um instrumento de gestão pública capaz de articular dados, monitorar ações e subsidiar políticas públicas no campo cultural. Este observatório é concebido como um sistema de informação dedicado à agregação, sistematização e tratamento de dados, oferecendo um panorama abrangente e atualizado do patrimônio cultural (Gusmão, 2006).

O projeto foi estruturado em cinco metas principais, iniciando no levantamento das fontes informacionais e na formação da equipe (Meta 1), seguida pela conceituação do modelo de observatório (Meta 2) e pela proposição tecnológica e informacional (Meta 3). A Meta 4 é responsável pela implementação e transferência do pacote tecnológico desenvolvido, garantindo sua aplicabilidade e continuidade. Já a Meta 5 trata da produção da documentação de apoio aos softwares, assegurando sua usabilidade e manutenção.

No presente momento, a Meta 4 surge com a finalidade de consolidar e entregar o pacote tecnológico elaborado nas metas anteriores, reunindo os instrumentos, sistemas e componentes necessários para a operacionalização plena do observatório. Esta meta contempla a finalização técnica dos módulos informacionais, a integração das bases de dados, a validação das funcionalidades do ambiente digital e o aperfeiçoamento dos mecanismos de acesso e visualização pública.

A entrega do pacote tecnológico na Meta 4 representa um marco estratégico no projeto, pois viabiliza a efetiva utilização do observatório pelos gestores públicos, pesquisadores e sociedade civil, promovendo maior transparência, eficiência e capacidade analítica nas políticas de patrimônio cultural. Assim, este relatório visa apresentar de forma detalhada os resultados e avanços alcançados, servindo como subsídio técnico para a próxima etapa de consolidação e institucionalização do observatório no âmbito do Iphan e do SNPC.

2 OBJETIVOS

2.1 Objetivo Geral

Implementação do Observatório Nacional de Patrimônio Cultural.

2.2 Objetivos Específicos

- Implementação do ambiente de teste e homologação no Ibict;
- Implementação do ambiente de produção no Iphan.

3 RESULTADOS

3.1 Implementação do ambiente de teste e homologação no Ibict

A presente etapa tem como objetivo a criação de um ambiente controlado, voltado para o teste, homologação e validação técnica das soluções que compõem o Observatório do SNPC. Neste ambiente, é possível implementar as ferramentas e tecnologias previstas no projeto, permitindo a realização de testes sistemáticos antes da implantação definitiva.

Nesse contexto, serão relatadas atividades referente a construção de um ambiente de teste do Observatório do SNPC. Entre estas atividades está a instalação das ferramentas, a elaboração de uma rotina de busca nas fontes já previamente mapeadas e selecionadas para a alimentação automatizada do banco de dados do observatório, realização de testes de conectividade e interoperabilidade entre sistemas, além da preparação de estruturas que permitam a visualização e análise das informações. Essa fase é essencial para garantir que a solução seja segura, funcional e adaptada às necessidades do projeto, antes de ser migrada para o ambiente de produção.

Esse processo permite validar a arquitetura proposta, testar os fluxos de dados e identificar inconsistências ou necessidades de aprimoramento em um ambiente seguro, sem comprometer a operação institucional do Iphan. Válido lembrar que enquanto projeto de pesquisa, metas e etapas se sobrepõem. Com a necessidade de materialidade e visualização do sistema ainda no período de vigência da meta 2 para a equipe Iphan no ambiente de homologação em partes, foi elaborado e implementado na meta 2 enquanto laboratório com aprimoramentos contínuos.

3.1.1 Implementação das ferramentas a comporem o ecossistema do observatório

Conforme prospectado na meta 2 e selecionado na meta 3, foram levantadas as tipologias de tecnologias e as tecnologias para compor o sistema de observatório. Segue a lista de tecnologias selecionadas com suas respectivas funcionalidades no observatório.

Com base na reunião realizada em 14 de Fevereiro no IPHAN a qual foi apresentado as plataformas BI, foi decidido o uso das plataformas Wordpress; Apache Superset; PostgreeSQL. Estabelecidos os softwares foi implementada a primeira versão do observatório e

apresentada em reunião 14 de Março no IBICT. Deste modo segue a tabela do tipo da tecnologia com a tecnologia escolhida:

Quadro 1 - Softwares escolhidos

Tipo de Tecnologia	Tecnologia escolhida
Plataforma de gerenciamento de conteúdo (CMS)	Wordpress
Plataforma Business Intelligence (BI)	Apache Superset
Sistema de Gerenciamento de Banco de Dados - SGBD	PostgreSQL

Fonte: elaborado pelos autores (2025).

Dito isso, foram instalados os *softwares* no ambiente de homologação do Ibict que está no endereço no endereço eletrônico: observatorio.iphan.ibict.br. Para que não fique repetitivo, visto que, a etapa seguinte corresponde à realização da instalação dos mesmos softwares só que no ambiente do Iphan, aqui serão apresentados os requisitos dos softwares que estão implementados no ambiente Cotec/Ibict para rodar essa versão de homologação. A instalação propriamente dita será apresentada na etapa seguinte. Desta forma segue os requisitos de máquina para o Wordpress e o Apache Superset.

3.1.1.1 Wordpress

Conforme apresentado na prospecção de tecnologias, o *WordPress* é um *software* livre que possibilita a arquitetura LAMP, composta de Sistema Operacional Linux, Servidor de Aplicação *Apache*, Banco de Dados MySQL e linguagem de programação PHP. Para instalar a versão mais recente do *Wordpress*, conforme o ambiente do Ibict têm-se os seguintes requisitos:

- Requisitos de *Software*:
  - Sistema operacional distribuição Linux;
  - Servidor Web Apache2;
  - Banco de dados MariaDB ou Mysql;
  - PHP versão.
- Requisitos de Hardware:
  - Procesador: Mínimo 4.0 GHz+ (8 núcleos);
  - Memória RAM: Mínimo 8 GB;

- Armazenamento em Disco: Mínimo 100G.

Foram instalados dois wordpress no ambiente de homologação. O portal do Observatório no endereço: observatorio.iphan.ibict.br e o Mural de Agentes no endereço: https://mural-agentes.prd.ibict.br/.

3.1.1.2 Apache Superset

O *Apache Superset* conforme apresentado na prospecção de tecnologia é um *software* livre, desenvolvido em *Python*, com *back-end* em *Flask, front-end em React*, e integração nativa com *PostgreSQL*. Trata-se de uma solução moderna para BI e *dashboards* interativos, amplamente adotada por organizações que demandam análise de dados escalável e flexível. Para instalar a versão mais recente do *Apache Superset* (4.1.2), conforme o ambiente do Ibict têm-se os seguintes requisitos:

- Requisitos de Software:
  a. Sistema operacional distribuição Linux;
  b. Banco de dados PostgreSQL (versão 14.x+ até 16.x estamos utilizando a 14.17);
  c. Python versão 3.x estamos utilizando a 3.10.12 com psycopg2 instalado para integração com o PostgreSQL.
- Requisitos de *Hardware*:
  - Processador: Mínimo 4.0 GHz+ (16 núcleos);
  - Memória RAM: Mínimo 16 GB;
  - Armazenamento em Disco: 100GB.

O Superset de homologação foi instalado no endereço: https://relatorios-superset.ibict.br/.

3.1.2 Verificar tecnologias para integração de fontes

Para a integração de diferentes fontes de dados no armazém de dados do OBS, foi desenvolvido um pipeline em Python estruturado em três etapas principais para cada script: extração, tratamento e o carregamento dos dados. Para isso, foram estudadas diversas bibliotecas que possibilitam a conexão com bancos de dados, o pré-processamento textual, a manipulação de dados, a realização de comparações para identificar dados relacionados, a normalização, dentre outros procedimentos. A seguir, destacam-se as principais tecnologias em Python adotadas:

- Pandas: biblioteca central utilizada em todas as etapas do processo para a manipulação e o tratamento dos dados em *dataframes*.
- Numpy: utilizada para operações dos *dataframes* e tratamento de dados ausentes ou inválidos em conjunto com o pandas.
- Pymysql: biblioteca utilizada para estabelecer conexões com bancos de dados MySQL, de onde são extraídos dados.
- Psycopg2: biblioteca utilizada para estabelecer conexões com bancos de dados PostgreSQL, de onde são extraídos dados, e também empregada na etapa de carregamento, permitindo a inserção ou atualização dos dados tratados no banco de dados PostgreSQL que compõe o armazém do OBS.
- Pyodbc: biblioteca utilizada para estabelecer conexões com bancos de dados que utilizam o padrão ODBC, como o SQL Server.
- Unidecode e unicodedata: utilizada para a normalização de textos, removendo acentos e caracteres especiais com o objetivo de padronizar os dados e facilitar a comparação entre fontes.
- Re (expressões regulares): utilizada para a limpeza e padronização de strings, como nomes de localidades e instituições, além da extração de coordenadas geográficas a partir de representações textuais no formato *Well-Known Text* (WKT).
- Rapidfuzz: biblioteca de fuzzy matching utilizada para realizar associações aproximadas entre nomes extraídos de fontes distintas, com base em similaridade textual.
- Time: utilizada para controle de tempo e pausas durante a execução de processos, especialmente em etapas sensíveis de leitura ou escrita em banco de dados.

Essas tecnologias possibilitaram o desenvolvimento de scripts em Python para a automação da integração de diferentes fontes de dados no armazém do OBS.

Além disso, com o objetivo de melhorar a performance durante o carregamento e a atualização dos dados no armazém, foram criados índices nas tabelas utilizadas pelas funções de carregamento dos dados. Esses índices otimizam a busca por colunas frequentemente utilizadas. Foram criados os seguintes índices: um índice composto nas

colunas id_valor e id_metadado da tabela obs_valor_metadado; um índice baseado no prefixo de 255 caracteres da coluna value, também na tabela obs_valor_metadado, e um índice sobre a coluna id_colecao da tabela obs_itens. A seguir, são listados os comandos utilizados no banco de dados do armazém:

Quadro 2 - Comandos para índices nas tabelas

```
CREATE INDEX IF NOT EXISTS idx_valor_metadado_composto_indice ON
obs_valor_metadado (id_valor, id_metadado);
CREATE INDEX IF NOT EXISTS idx_valor_metadado_value_prefix ON
obs_valor_metadado (LEFT(value, 255));
CREATE INDEX IF NOT EXISTS idx_itens_colecao ON obs_itens (id_colecao);
```

Fonte: Elaborado pelos autores (2025).

3.1.3 Migração dos dados ou integração de dados de sistemas externos

A atividade 3.1.3 tem como objetivo identificar, testar e validar as tecnologias mais adequadas para viabilizar a integração de diferentes fontes de dados ao ecossistema do Observatório do SNPC. Esta etapa é fundamental para assegurar que os dados oriundos de sistemas diversos possam ser acessados, interpretados e incorporados de forma estruturada e segura à plataforma.

3.1.3.1 Automação da Atualização e Integração de Dados no Armazém

Durante esta fase, foram desenvolvidos scripts em linguagem Python para realizar a atualização automática dos dados no armazém. O processo inclui a coleta automatizada dos dados a partir das fontes mapeadas, o tratamento e o carregamento. A etapa do carregamento garante a sincronização dos dados tratados com o armazém, mantendo atualizações incrementais, controle de duplicidade com base nos campos co_iphan, id_agente e id_controle utilizados, respectivamente, para os indicadores de Bens Culturais, Agentes e Ações além da preservação dos metadados associados a cada item no armazém. Caso sejam identificadas diferenças entre os dados de origem e os já existentes no armazém, os valores são atualizados.

Portanto, com a periodização e rotinas de atualização é possível o acesso direto aos bancos de dados (devidamente levantados na meta 1 e selecionados na meta 2). Esta entrega

materializa a dinâmica automatizada de extração, tratamento e ingestão dos dados no ambiente do Observatório.

Também foram criadas views em SQL para organizar os dados obtidos, promovendo uma estrutura que facilite tanto a visualização quanto a análise posterior. Essas ações permitiram testar a robustez das tecnologias selecionadas, além de fornecer insumos técnicos importantes para as próximas fases de migração e integração definitiva.

Assim, os testes realizados nesta atividade subsidiam diretamente o processo de escolha das ferramentas e métodos que serão utilizados na construção das rotinas de coleta, transformação e carregamento de dados, fortalecendo a base tecnológica do Observatório. Dessa forma, os scripts são apresentados na seguinte ordem:

1) Bens Culturais
2) Agentes
3) Ações

3.1.3.1.1 Indicador - Bens Culturais

Foram desenvolvidos dois scripts de automação, para o indicador de Bens Culturais, sendo: 1) Bens culturais imateriais registrados, fonte: BCR e 2) Bens culturais materiais, ferroviários, arqueológicos, fonte: SICG. Além disso, são integrados dados dos agentes e ações vinculados aos bens.

a) Script de Automação do Indicador de Bens Imateriais

Os processos descritos a seguir foram implementados como parte do fluxo de integração dos dados de bens culturais registrados no armazém de dados.

b) Extração

Foi desenvolvida uma função para realizar a extração de dados a partir da base de dados do BCR em MySQL. A função extrair_dados_mysql() realiza a conexão com o banco de dados e possibilita a leitura e estruturação dos dados dos bens culturais do BCR.

c) Implementação do Tratamento

Após a extração o script realiza o tratamento dos dados extraídos. A seguir, apresentam-se os principais passos:

- Normalização de Nomes: As colunas são renomeadas por exemplo: post_title para nome_bem.
- Usa *fuzzy matching*: É realizado o cruzamento aproximado dos nomes dos bens da base tg_bem_imaterial.csv para recuperar as informações geográficas dos bens, utilizando a biblioteca fuzzywuzzy, com similaridade mínima de 70%.
- Formatação Geográfica: A coluna com ponto geo é convertida de formato POINT() para lista [longitude, latitude], extraindo longitude e latitude separadamente.
- Classificação: A partir do campo "livros", os bens são classificados conforme os tipos previstos no Livro do Registro: Formas de Expressão, Celebrações, Saberes e Lugares.
- Normalização de datas: Datas de registro são padronizadas e transformadas para obter colunas derivadas como ano, década e nome do mês.
- Tipo de agente: Os agentes relacionados são identificados e categorizados como "Instituição Parceira Bem Imaterial".
- A localização dos bens culturais imateriais é representada na base por uma coluna chamada "localização", a qual pode conter múltiplos valores separados por vírgulas. Para tratar corretamente essas informações e extrair outros dados geográficos, o script implementa os seguintes passos:
    - Tratamento multivalorado da localização: Cada entrada da coluna "localização" é decomposta em pares Estado - UF: Município. Os campos UF, UF ISO, UF Estado, Região e Município são extraídos, tratados e armazenados como campos multivalorados (valores separados por |).
    - Preenchimento de municípios ausentes: Em casos em que o nome do município está ausente na localização, o script utiliza a base auxiliar bens_imaterais_municipios_mapeados.csv, que contém a relação entre co_iphan, UF e os respectivos municípios mapeados manualmente pela equipe, para preencher os valores faltantes.
    - Extração da sigla da UF: A partir do nome do estado (uf_estado), é extraída a sigla da UF (ex: "SP", "RJ", "MG") por meio de expressão regular diretamente do campo "localização".

    - Associação de regiões geográficas: A partir da sigla da UF, é determinada a região brasileira correspondente (Norte, Nordeste, Centro-Oeste, Sudeste ou Sul) utilizando o dicionário uf_para_regiao.
    - Normalização segundo ISO 3166-2: Cada unidade da federação é também convertida para seu código no padrão ISO 3166-2 (por exemplo, "BR-SP", "BR-BA"), o que padroniza os dados conforme normas internacionais e facilita sua integração com serviços e bases externas.
    - Formatação padronizada do município: O campo "nome_municipio" é montado no padrão "Município - UF", promovendo consistência e legibilidade nos dados.
- Padronização de agentes: Os nomes dos agentes são agrupados por similaridade e padronizados. Agentes identificados como universidades federais são substituídos por sua nomenclatura oficial. Um caso especial de substituição é aplicado ao agente “Vídeo nas Aldeias”.
- Preenchimento da coluna possui_agentes_acoes: A coluna é preenchida com “Sim” sempre que o bem tiver um tipo de agente associado.
- Tratamento de Caracteres: Aspas simples (') são substituídas por espaço para evitar erros de inserção em SQL.

d) Inserção/Atualização no Armazém de Dados

Após o tratamento dos dados, é realizada a etapa de inserção ou atualização dos valores dos metadados no armazém de dados, utilizando conexão com banco do armazém PostgreSQL. Este processo garante que os dados tratados sejam devidamente sincronizados com o armazém de dados, mantendo atualizações incrementais, controle de duplicidade por meio do co_iphan e preservando os metadados associados a cada item da coleção.

Abaixo, apresenta-se o script completo desenvolvido para os Bens Culturais Registrados.

Quadro 3 - Script Completo - Bens Imateriais

```
#!/usr/bin/env python
# coding: utf-8

import pandas as pd
import pymysql
import re
import unidecode
```

```
import psycopg2
from rapidfuzz import process, fuzz

#----------Extracao Bens Imateriais-------------
#Funcao para MySQL e retorna um DataFrame
def extrair_dados_mysql(query: str, host: str, user: str, password: str, database: str)
-> pd.DataFrame:
   conn = None
   try:
      conn = pymysql.connect(
         host=host,
         user=user,
         password=password,
         database=database
      )
      df = pd.read_sql(query, conn)
      return df
   finally:
      if conn:
         conn.close()

query_bcr = """
SELECT
   p.ID,
   p.post_title,
   p.guid AS url,

   -- Localização com estado incluído (termo pai)
   GROUP_CONCAT(
      DISTINCT
      CASE
         WHEN tntloc.parent != 0 THEN
            CONCAT_WS(
               ' : ',
               (SELECT parent_term.name
                FROM wp_term_taxonomy parent_tt
                JOIN wp_terms parent_term ON parent_tt.term_id = parent_term.term_id
                WHERE parent_tt.term_taxonomy_id = tntloc.parent),
               tloc.name
            )
         ELSE tloc.name
      END
      ORDER BY tloc.name ASC SEPARATOR ', '
   ) AS localização,

   GROUP_CONCAT(DISTINCT tlivro.name ORDER BY tlivro.name ASC SEPARATOR ',
') AS livro_de_registro,

   MAX(CASE WHEN pm.meta_key = '65736' THEN pm.meta_value END) AS
denominacao,
```

```
   MAX(CASE WHEN pm.meta_key = '89375' THEN pm.meta_value END) AS
data_registro,
   MAX(CASE WHEN pm.meta_key = '65751' THEN pm.meta_value END) AS
abrangência_registro,
   MAX(CASE WHEN pm.meta_key = '65770' THEN pm.meta_value END) AS
instituições_parceiras

FROM
   wp_posts p
   INNER JOIN wp_postmeta pm ON p.ID = pm.post_id

   LEFT JOIN wp_term_relationships trloc ON p.ID = trloc.object_id
   LEFT JOIN wp_term_taxonomy tntloc ON trloc.term_taxonomy_id =
tntloc.term_taxonomy_id
      AND tntloc.taxonomy = 'tnc_tax_7447'
   LEFT JOIN wp_terms tloc ON tntloc.term_id = tloc.term_id

   LEFT JOIN wp_term_relationships trlivro ON p.ID = trlivro.object_id
   LEFT JOIN wp_term_taxonomy tntlivro ON trlivro.term_taxonomy_id =
tntlivro.term_taxonomy_id
      AND tntlivro.taxonomy = 'tnc_tax_65757'
   LEFT JOIN wp_terms tlivro ON tntlivro.term_id = tlivro.term_id

WHERE
   p.post_type = 'tnc_col_65733_item'

GROUP BY
   p.ID, p.post_title, p.post_date, p.post_status, p.guid

ORDER BY
   p.post_title ASC;


"""

df = extrair_dados_mysql(
   query=query_bcr,
   host="XXXXXX",
   user="XXXXXX",
   password="XXXXXX",
   database="XXXXXXX"
)

#Salva
df.to_csv("util/base_bcr.csv", index=False)

#----------Tratamento Bens Imateriais----------

def tratar_dados_bcr() -> pd.DataFrame:
   df = pd.read_csv("util/base_bcr.csv")
```

```
   #Carrega os dados baixados do GEOSERVER
   geo_df = pd.read_csv("util/tg_bem_imaterial.csv")
   #Carrega municipios mapeados pela equipe
   df_ref = pd.read_csv("util/bens_imaterais_municipios_mapeados.csv")

   # Renomear colunas
   df.rename(columns={
      "ID": "co_iphan",
      "post_title": "nome_bem",
      "abrangência_registro": "abrangencia_registro",
      "instituições_parceiras": "nome_agente",
      "livro_de_registro": "livros"
   }, inplace=True)

   # Função de normalização
   def normalizar_texto(texto):
      if pd.isnull(texto):
         return ""
      texto = texto.lower().strip()
      texto = unidecode.unidecode(texto)
      texto = texto.replace("–", "-").replace("—", "-")
      texto = re.sub(r'[-——]', ' ', texto)
      texto = re.sub(r'[^\w\s]', '', texto)
      texto = re.sub(r'\s+', ' ', texto)
      return texto

   #Normaliza campos para comparação
   df["nome_bem_norm"] = df["nome_bem"].apply(normalizar_texto)
   geo_df["no_bem_imaterial_norm"] =
geo_df["no_bem_imaterial"].apply(normalizar_texto)

   #Fuzzy matching
   mapeamento = {}
   geo_nomes = geo_df["no_bem_imaterial_norm"].tolist()
   for nome in df["nome_bem_norm"].unique():
      resultado = process.extractOne(nome, geo_nomes, scorer=fuzz.token_sort_ratio)
      if resultado:
         match = resultado[0]
         score = resultado[1]
         if score >= 70:
            mapeamento[nome] = match

   # Match reverso
   geo_df_match =
geo_df[geo_df["no_bem_imaterial_norm"].isin(mapeamento.values())].copy()
   reverse_map = {v: k for k, v in mapeamento.items()}
   geo_df_match["nome_bem_norm"] =
geo_df_match["no_bem_imaterial_norm"].map(reverse_map)

   # Só mantém com ponto válido
```

```
    geo_df_match = geo_df_match.dropna(subset=["geometria_ponto"])


    # Merge aproximado
    df = df.merge(
        geo_df_match[["nome_bem_norm", "geometria_ponto"]],
        on="nome_bem_norm",
        how="left"
    )

    # Extrai ponto_geo
    if "geometria_ponto" in df.columns:
        df.rename(columns={"geometria_ponto": "ponto_geo"}, inplace=True)
        coords = df["ponto_geo"].str.extract(r'POINT \(([-\d\.]+) ([-\d\.]+)\)')
        df["longitude"] = pd.to_numeric(coords[0], errors="coerce")
        df["latitude"] = pd.to_numeric(coords[1], errors="coerce")
        df["ponto_geo"] = df.apply(
            lambda row: [row["longitude"], row["latitude"]] if
pd.notnull(row["longitude"]) and pd.notnull(row["latitude"]) else None,
            axis=1
        )
    else:
        print("Coluna 'geometria_ponto' não encontrada após o merge.")

    # Colunas fixas
    df["bem_tipo_protecao"] = "Registro"
    df["bem_por_tipo"] = "Imateriais"

    #Classificação
    livro_para_classificacao = {
        'Livro das Formas de Expressão': 'Formas de Expressão',
        'Livro das Celebrações': 'Celebrações',
        'Livro dos Saberes': 'Saberes',
        'Livro dos Lugares': 'Lugares'
    }
    df["classificacao"] = df["livros"].map(livro_para_classificacao)

    #Normalizacao das Datas
    df["data_registro"] = pd.to_datetime(df["data_registro"], errors='coerce')
    df["data_registro"] = df["data_registro"].dt.strftime('%d/%m/%Y')
    df["ano"] = pd.to_datetime(df["data_registro"], format='%d/%m/%Y',
errors='coerce').dt.year
    df["decada"] = (df["ano"] // 10 * 10).astype('Int64')
    df["mes_registro_nome"] = pd.to_datetime(df["data_registro"],
format='%d/%m/%Y', errors='coerce').dt.month.map({
        1: "Janeiro", 2: "Fevereiro", 3: "Março", 4: "Abril", 5: "Maio", 6: "Junho",
        7: "Julho", 8: "Agosto", 9: "Setembro", 10: "Outubro", 11: "Novembro", 12:
"Dezembro"
    })
```

```
  #Tipo de agente
  df["tipo_agente"] = df["nome_agente"].apply(
    lambda x: "Instituição Parceira Bem Imaterial" if pd.notnull(x) and str(x).strip()
!= "" else None
  )

  #Dicionário UF -> Região
  uf_para_regiao = {
    'AC': 'Norte', 'AP': 'Norte', 'AM': 'Norte', 'PA': 'Norte', 'RO': 'Norte', 'RR': 'Norte', 'TO':
'Norte',
    'AL': 'Nordeste', 'BA': 'Nordeste', 'CE': 'Nordeste', 'MA': 'Nordeste', 'PB': 'Nordeste',
    'PE': 'Nordeste', 'PI': 'Nordeste', 'RN': 'Nordeste', 'SE': 'Nordeste',
    'DF': 'Centro-Oeste', 'GO': 'Centro-Oeste', 'MT': 'Centro-Oeste', 'MS':
'Centro-Oeste',
    'ES': 'Sudeste', 'MG': 'Sudeste', 'RJ': 'Sudeste', 'SP': 'Sudeste',
    'PR': 'Sul', 'RS': 'Sul', 'SC': 'Sul'
  }

  #Trata a coluna localização com saída multivalorada
  def tratar_localizacao_multivalorada(row):
    locais = re.split(r',\s*', str(row['localização']).replace("–", "-"))

    uf_estados, ufs, uf_isos, regioes, municipios = [], [], [], [], []

    for local in locais:
      if ':' in local:
        estado_uf, municipio = local.split(':')
        estado_uf = estado_uf.strip()
        municipio = municipio.strip()
      else:
        estado_uf = local.strip()
        municipio = None

      match = re.match(r'(.*) - ([A-Z]{2})', estado_uf)
      if match:
        estado, uf = match.groups()
        uf_iso = f"BR-{uf}"
        regiao = uf_para_regiao.get(uf, "")
        nome_municipio = f"{municipio} - {uf}" if municipio else ""

        uf_estados.append(estado)
        ufs.append(uf)
        uf_isos.append(uf_iso)
        regioes.append(regiao)
        municipios.append(nome_municipio)

    return pd.Series({
      "uf_estado": "|".join(uf_estados),
      "uf": "|".join(ufs),
      "uf_iso": "|".join(uf_isos),
```

```
        "regiao": "|".join(regioes),
        "nome_municipio": "|".join(municipios)
      })

    #Aplica a função
    df_multiloc = df.dropna(subset=["localização"]).copy()
    multivalores = df_multiloc.apply(tratar_localizacao_multivalorada, axis=1)

    #Junta com o DataFrame original
    df_final = pd.concat([df_multiloc.drop(columns=["localização",
"nome_bem_norm"], errors="ignore"), multivalores], axis=1)
    df_final.drop(columns=["nome_bem_norm"], inplace=True, errors="ignore")
    df_final.drop(columns=["localização"], inplace=True, errors="ignore")


    #Preenche nome_municipio multivalorado onde estiver faltando
    def preencher_municipios_multivalorados(row):
      ufs = row["uf"].split("|")
      municipios = row["nome_municipio"].split("|")

      novos_municipios = []
      for i in range(len(ufs)):
        municipio = municipios[i]
        uf = ufs[i]

        if municipio.strip() == "":
          ref = df_ref[(df_ref["co_iphan"] == row["co_iphan"]) & (df_ref["uf"] == uf)]
          if not ref.empty:
            # Pega todos os municípios para esse co_iphan e uf
            municipios_uf = ref["nome_municipio"].tolist()
            novos_municipios.extend(municipios_uf)
            continue
        else:
          novos_municipios.append(municipio)

      return "|".join(novos_municipios)


    def expandir_localizacao_completa(row):
      co_iphan = row["co_iphan"]
      ufs_originais = row["uf"].split("|")
      uf_estados_originais = row["uf_estado"].split("|")
      uf_isos_originais = row["uf_iso"].split("|")
      regioes_originais = row["regiao"].split("|")
      municipios_originais = row["nome_municipio"].split("|")

      novos_ufs = []
      novos_uf_estados = []
      novos_uf_isos = []
      novas_regioes = []
```

```
    novos_municipios = []

    for i, uf in enumerate(ufs_originais):
      municipio = municipios_originais[i] if i < len(municipios_originais) else ""
      uf_estado = uf_estados_originais[i] if i < len(uf_estados_originais) else ""
      uf_iso = uf_isos_originais[i] if i < len(uf_isos_originais) else ""
      regiao = regioes_originais[i] if i < len(regioes_originais) else ""

      if municipio.strip() == "":
        # Pega todos os municípios para esse co_iphan e uf
        ref = df_ref[(df_ref["co_iphan"] == co_iphan) & (df_ref["uf"] == uf)]
        municipios_encontrados = ref["nome_municipio"].tolist()

        if municipios_encontrados:
          for m in municipios_encontrados:
            novos_ufs.append(uf)
            novos_uf_estados.append(uf_estado)
            novos_uf_isos.append(uf_iso)
            novas_regioes.append(regiao)
            novos_municipios.append(m)
        else:
          # Mantém o estado com campo de município vazio
          novos_ufs.append(uf)
          novos_uf_estados.append(uf_estado)
          novos_uf_isos.append(uf_iso)
          novas_regioes.append(regiao)
          novos_municipios.append("")
      else:
        novos_ufs.append(uf)
        novos_uf_estados.append(uf_estado)
        novos_uf_isos.append(uf_iso)
        novas_regioes.append(regiao)
        novos_municipios.append(municipio)

    return pd.Series({
      "uf": "|".join(novos_ufs),
      "uf_estado": "|".join(novos_uf_estados),
      "uf_iso": "|".join(novos_uf_isos),
      "regiao": "|".join(novas_regioes),
      "nome_municipio": "|".join(novos_municipios)
    })


  df_final[["uf", "uf_estado", "uf_iso", "regiao", "nome_municipio"]] =
df_final.apply(expandir_localizacao_completa, axis=1)


  #Remove as aspas simples: percorre todos os elementos do DataFrame, e se o
valor for uma string, ele substitui por espaco
  df_final = df_final.applymap(lambda x: x.replace("'", " ") if isinstance(x, str) else x)
```

```
universidades_federais_dict = {
    'unb': 'Universidade de Brasília',
    'ufmg': 'Universidade Federal de Minas Gerais',
    'ufrj': 'Universidade Federal do Rio de Janeiro',
    'ufsc': 'Universidade Federal de Santa Catarina',
    'ufrgs': 'Universidade Federal do Rio Grande do Sul',
    'ufpe': 'Universidade Federal de Pernambuco',
    'ufba': 'Universidade Federal da Bahia',
    'ufscar': 'Universidade Federal de São Carlos',
    'ufrn': 'Universidade Federal do Rio Grande do Norte',
    'ufpr': 'Universidade Federal do Paraná',
    'ufpa': 'Universidade Federal do Pará',
    'ufam': 'Universidade Federal do Amazonas',
    'ufpb': 'Universidade Federal da Paraíba',
    'ufes': 'Universidade Federal do Espírito Santo',
    'ufal': 'Universidade Federal de Alagoas',
    'ufg': 'Universidade Federal de Goiás',
    'ufma': 'Universidade Federal do Maranhão',
    'ufmt': 'Universidade Federal de Mato Grosso',
    'ufms': 'Universidade Federal de Mato Grosso do Sul',
    'uft': 'Universidade Federal do Tocantins',
    'ufopa': 'Universidade Federal do Oeste do Pará',
    'ufvjm': 'Universidade Federal dos Vales do Jequitinhonha e Mucuri',
    'uff': 'Universidade Federal Fluminense',
    'ufrrj': 'Universidade Federal Rural do Rio de Janeiro',
    'utfpr': 'Universidade Tecnológica Federal do Paraná',
    'uffs': 'Universidade Federal da Fronteira Sul',
    'unila': 'Universidade Federal da Integração Latino-Americana',
    'ufcspa': 'Universidade Federal de Ciências da Saúde de Porto Alegre',
    'ufpel': 'Universidade Federal de Pelotas',
    'ufsm': 'Universidade Federal de Santa Maria',
    'unipampa': 'Universidade Federal do Pampa',
    'furg': 'Universidade Federal do Rio Grande',
    'ufape': 'Universidade Federal do Agreste de Pernambuco',
    'ufdpar': 'Universidade Federal do Delta do Parnaíba',
    'ufnt': 'Universidade Federal do Norte do Tocantins',
    'ufabc': 'Universidade Federal do ABC',
    'ufac': 'Universidade Federal do Acre',
    'ufca': 'Universidade Federal do Cariri',
    'ufc': 'Universidade Federal do Ceará',
    'ufcg': 'Universidade Federal de Campina Grande',
    'ufersa': 'Universidade Federal Rural do Semi-Árido',
    'ufjf': 'Universidade Federal de Juiz de Fora',
    'ufla': 'Universidade Federal de Lavras',
    'ufop': 'Universidade Federal de Ouro Preto',
    'ufrb': 'Universidade Federal do Recôncavo da Bahia',
    'ufrpe': 'Universidade Federal Rural de Pernambuco',
    'ufs': 'Universidade Federal de Sergipe',
    'ufse': 'Universidade Federal de São João del-Rei',
```

```
    'ufu': 'Universidade Federal de Uberlândia',
    'ufv': 'Universidade Federal de Viçosa',
    'unifap': 'Universidade Federal do Amapá',
    'unifal': 'Universidade Federal de Alfenas',
    'unifei': 'Universidade Federal de Itajubá',
    'unifesspa': 'Universidade Federal do Sul e Sudeste do Pará',
    'unifesp': 'Universidade Federal de São Paulo',
    'unilab': 'Universidade da Integração Internacional da Lusofonia
Afro-Brasileira',
    'unir': 'Universidade Federal de Rondônia',
    'unirio': 'Universidade Federal do Estado do Rio de Janeiro',
    'univasf': 'Universidade Federal do Vale do São Francisco',
    'ufrr': 'Universidade Federal de Roraima',
    'uftm': 'Universidade Federal do Triângulo Mineiro',
    'ufj': 'Universidade Federal de Jataí',
    'ufcat': 'Universidade Federal de Catalão',
    'ufr': 'Universidade Federal de Rondonópolis'
  }

  #Função de normalização (sem acento, minúsculas, sem espaços extras)
  def normalize(texto):
    if pd.isnull(texto):
      return ''
    return unidecode.unidecode(texto.strip().lower())

  #Agrupar nomes semelhantes por similaridade (mantendo a forma original mais
usada)
  nomes_unicos = df['nome_agente'].dropna().unique()
  grupos = []
  padrao_por_nome = {}

  for nome in nomes_unicos:
    norm = normalize(nome)
    matched = False
    for grupo in grupos:
      representante = grupo[0]
      score = fuzz.ratio(norm, normalize(representante))
      if score >= 85:
        grupo.append(nome)
        padrao_por_nome[nome] = representante
        matched = True
        break
    if not matched:
      grupos.append([nome])
      padrao_por_nome[nome] = nome

  #Aplica agrupamento ao DataFrame
  df_final['nome_agente_padronizado'] =
df_final['nome_agente'].map(padrao_por_nome)
```

```
    # Substitui pelo nome completo da universidade, se estiver no dicionário
    def substituir_por_nome_universidade(nome):
        sigla = normalize(nome)
        if sigla in universidades_federais_dict:
            return universidades_federais_dict[sigla]
        return nome

    #Substitui agente "Vídeo nas Aldeias"
    def substituir(nome):
        norm = normalize(nome)
        if norm == 'video nas aldeias':
            return ''
        elif norm in universidades_federais_dict:
            return universidades_federais_dict[norm]
        return nome

    df_final['nome_agente_padronizado'] =
df_final['nome_agente_padronizado'].apply(substituir_por_nome_universidade)


    #Aplica substituições
    df_final['nome_agente_padronizado'] =
df_final['nome_agente_padronizado'].apply(substituir)

    #Substitui a coluna original e remove temporária
    df_final['nome_agente'] = df_final['nome_agente_padronizado']
    df_final.drop(columns=['nome_agente_padronizado'], inplace=True)
    df_final.rename(columns={"ponto_geo": "ponto"}, inplace=True)

    df_final['possui_agentes_acoes'] = df.apply(
        lambda row: 'Sim' if pd.notna(row['tipo_agente']) and row['tipo_agente'] != ''
else '',
        axis=1
    )

    #Exporta o resultado final
    return df_final  # DataFrame tratado


#----------Carrega os dados no armazém de dados----------
def carregando_dados_armazem():
    df_final = pd.read_csv("util/dados_bcr_tratados.csv")

    # Conexão com o banco
    conn = psycopg2.connect(
        host="XXXXXXX",
        dbname="XXXXXXXX",
        user="XXXXXXXX",
        password="XXXXXXXXX"
    )
```

```
    cursor = conn.cursor()

    # ID correto da coleção
    id_colecao = 1

    # ID fixo do metadado 'co_iphan'
    id_metadado_co_iphan = 1

    # Busca os metadados dessa coleção
    cursor.execute("SELECT id, nome FROM obs_metadados WHERE id_colecao = %s",
(id_colecao,))
    metadados_map = {nome: id_metadado for id_metadado, nome in
cursor.fetchall()}

    # Itera sobre o DataFrame
    for _, row in df_final.iterrows():
        co_iphan = row.get("co_iphan")
        if pd.isna(co_iphan) or str(co_iphan).strip() == "":
            print("Linha sem co_iphan, ignorada.")
            continue

        co_iphan_str = str(co_iphan).strip()

        # Verifica se já existe item com esse co_iphan na coleção correta
        cursor.execute("""
            SELECT i.id
            FROM obs_valor_metadado v
            JOIN obs_itens i ON v.id_valor = i.id
            WHERE v.id_metadado = %s AND v.value = %s AND i.id_colecao = %s
        """, (id_metadado_co_iphan, co_iphan_str, id_colecao))
        result = cursor.fetchone()

        if result:
            id_item = result[0]
            print(f"[UPDATE] Item (id={id_item}) já existe com co_iphan={co_iphan_str}")
        else:
            post_title = row.get("nome_bem") or "Item sem título"
            cursor.execute("""
                INSERT INTO obs_itens (id_colecao, post_title) VALUES (%s, %s)
RETURNING id
            """, (id_colecao, post_title))
            id_item = cursor.fetchone()[0]
            print(f"[INSERT] Novo item (id={id_item}) criado com
co_iphan={co_iphan_str}")

        # Insere ou atualiza os valores dos metadados
        for col in df_final.columns:
            if col in metadados_map:
                valor = row[col]
                if pd.notnull(valor) and str(valor).strip() != "":
```

```
            id_metadado = metadados_map[col]
            valor_str = str(valor).strip()

            # Verifica se valor já existe
            cursor.execute("""
               SELECT 1 FROM obs_valor_metadado
               WHERE id_valor = %s AND id_metadado = %s
            """, (id_item, id_metadado))

            if cursor.fetchone():
               cursor.execute("""
                  UPDATE obs_valor_metadado
                  SET value = %s
                  WHERE id_valor = %s AND id_metadado = %s
               """, (valor_str, id_item, id_metadado))
            else:
               cursor.execute("""
                  INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
                  VALUES (%s, %s, %s)
               """, (id_item, id_metadado, valor_str))

   # Commit e fechamento
   conn.commit()
   cursor.close()
   conn.close()

   print("Inserção/atualização finalizada com sucesso para coleção id = 1.")


if __name__ == "__main__":
   #Etapa 1: Extração (realiza a extracao no inicio)

   #Etapa 2: Tratamento
   df_tratado = tratar_dados_bcr()
   df_tratado.to_csv("util/dados_bcr_tratados.csv", index=False)

   #Etapa 3: Carga
   carregando_dados_armazem()
```

Fonte: elaborado pelos autores (2025).

e) Script de Automação do Indicador de Bens Materiais, Arqueológicos, Ferroviários

Os processos descritos a seguir foram implementados como parte do fluxo de integração dos dados do indicador de bens materiais, arqueológicos e ferroviários no armazém de dados.

f) Extração de Bens Materiais, Arqueológicos, Ferroviários

Foram desenvolvidas quatro funções para realizar a extração de dados a partir de diferentes bancos relacionais: PostgreSQL, MySQL e SQL Server.

- PostgreSQL – Base de dados SICG

A função extrair_dados_postgresql() permite a extração direta de tabelas do banco SICG, utilizado pelo IPHAN. Além disso, foi criada a função extrair_em_blocos_postgres() para realizar extrações paginadas (com LIMIT e OFFSET), para tabela tg_bem visto que possui grandes volumes de dados. As principais tabelas extraídas incluem: tg_bem, tb_protecao_bem, tb_classificacao, tb_natureza, tb_forma_inscricao, tg_uf, tg_municipio, tb_bem_contato, tb_contato.

- MySQL – Base Tainacan (Mural de Agentes)

A função extrair_dados_mysql() realiza a conexão com o banco MySQL onde está hospedado o Tainacan responsável pelo mural de agentes. A função possibilita a leitura e estruturação dos dados dos agentes vinculados aos bens culturais.

- SQL Server – Base FISCALIS

Para acesso ao banco de dados FISCALIS, foi desenvolvida a função extrair_dados_sqlserver(), que conecta-se via ODBC ao servidor SQL Server. A função é utilizada para extrair dados da tabela: acao_fiscalizacao.

g) Implementação do Tratamento

Após a extração o script realiza o tratamento dos dados extraídos. A seguir, apresentam-se os principais passos:

- Filtra apenas os bens que estão ativos no SICG: Seleciona as linhas com a coluna ativo = True.
- Normalização de nomes: Renomeia as colunas. Exemplo: identificacao_bem para nome_bem.

- Remoção de registros de teste: Linhas com "teste" no nome_bem são removidas (ex: “teste”, “um teste”). Palavras como “testemunho” são preservadas.
- Normalização de tipos: tipo_estado_conservacao e tipo_estado_preservacao são convertidos para descrições textuais usando dicionários.
- Mapeamento de proteção: A partir da tabela tb_protecao_bem, é feito o mapeamento do tipo de proteção (id_tipo_protecao) para nomes descritivos, como "Tombamento".
- Formatação geográfica: A coluna ponto é convertida de formato POINT() para lista [longitude, latitude], extraindo longitude e latitude separadamente.
- Geração de URL: A URL para visualização do bem no SICG é criada com base no id_bem.
- Conversão de identificadores geográficos: A partir do campo id_municipio, o script realiza o cruzamento com as tabelas de referência de municípios (tg_municipio.csv) e de unidades federativas (tg_uf.csv) para obter o nome do município e o nome do estado (UF extensa):
    - Extração da sigla da UF (ex: SP, RJ, MG): Com base no nome do estado (uf_estado), é aplicada uma tabela de mapeamento (estado_para_uf) para derivar a sigla correspondente.
    - Associação de regiões geográficas: Utiliza a sigla da UF, o script identifica a região geográfica (Norte, Nordeste, Centro-Oeste, Sudeste ou Sul) por meio do dicionário uf_para_regiao.
    - Normalização com ISO 3166-2: Cada unidade federativa é associada ao seu código padronizado segundo o padrão ISO 3166-2:BR (ex: BR-SP, BR-BA) usando o dicionário estados_map, o que permite maior interoperabilidade com sistemas externos e padronização internacional.
    - Formatação de nome do município: O campo nome_municipio é formatado na forma "Município - UF", unificando os dados para facilitar leitura e indexação.
- Normalização da coluna valorado: O valor é normalizado para “Sim” ou “Não”.
- Formatação de datas: A data de cadastro (dt_cadastro) é convertida para formato dd/mm/yyyy.
- Padronização de Logradouro: Campos vazios ou com “não informado” são padronizados como “Não informado”.

- Datas de inscrição: A partir do arquivo tb_forma_inscricao.csv, mapeia-se a data de inscrição usando id_protecao_bem. Como podem ser múltiplas, os campos data_inscricao, ano e decada são multivalorados separados por barra vertical "|".
- Mapeamento de classificação e natureza: Utiliza os arquivos tb_classificacao.csv e tb_natureza.csv para preencher as colunas correspondentes com base em id_classificacao.
- Livros: Analisa o texto da justificativa (tb_forma_inscricao.csv), aplica regex e normalização para extrair os tipos de livros.
- Mapeamento de tipo: Traduz id_tipo_bem para descrições como "Conjunto Urbano", "Sítio", dentre outros, via dicionário.
- Mapeamento de agentes:
    - Usa tb_contato.csv e tb_bem_contato.csv para recuperar agentes vinculados a cada bem.
    - Tipo de agente: Os agentes relacionados são identificados e categorizados como "Contato Bem Material".
    - Considera-se apenas agentes jurídicos.
    - Usa fuzzy matching (similaridade ≥ 70%) com mural_agentes_wordpress.csv para extrair url_mural_agente, natureza_organizacao_agente e atuacao_agente.
    - Campos relacionados são multivalorados e preservam a ordem dos nomes.
- Mapeamento de ações de fiscalização: Relaciona o co_iphan do bem com o campo cod_bem da tabela acao_fiscalizacao.csv. Cria as colunas total_acoes e tipo_acao.
- Coluna de indicador: A nova coluna possui_agentes_acoes marca como "Sim" se o bem tiver agentes ou ações vinculadas.
- Tratamento de caracteres: Aspas simples (') são substituídas por espaço para evitar erros de inserção em SQL.

h) Inserção/Atualização no Armazém de Dados

Após o tratamento dos dados, é realizada a etapa de inserção ou atualização dos valores dos metadados no armazém de dados, utilizando conexão com banco do armazém PostgreSQL. Este processo garante que os dados tratados sejam devidamente sincronizados

com o armazém de dados, mantendo atualizações incrementais, controle de duplicidade via co_iphan e preservando os metadados associados a cada item da coleção.

Abaixo, apresenta-se o script completo desenvolvido para os bens materiais, ferroviários e arqueológicos.

Quadro 4 - Script Completo - Bens Materiais, Ferroviários e Arqueológicos

```
#!/usr/bin/env python
# coding: utf-8

#SICG - Bens Culturais Materiais, Arqueológicos e Ferroviários

#Bibliotecas
import numpy as np
import pandas as pd
import pymysql
import re
from unidecode import unidecode
import psycopg2
from fuzzywuzzy import process, fuzz
import pyodbc
import time

#--------------Extracao na base de dados do SICG------------

#Executa uma query no PostgreSQL e retorna um DataFrame
def extrair_dados_postgresql(query: str, host: str, user: str, password: str, database:
str, port: int = 5432) -> pd.DataFrame:
    import psycopg2
    conn = None
    try:
        conn = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            dbname=database,
            port=port
        )
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()

#Executa uma query no PostgreSQL e retorna um DataFrame (Extrai de forma
paginada)
def extrair_em_blocos_postgres(
    query_template: str,
```

```
    host: str,
    user: str,
    password: str,
    database: str,
    bloco: int = 50000,
    output_csv: str = "util/tg_bem.csv"
):

    conn = psycopg2.connect(
        host=host,
        user=user,
        password=password,
        dbname=database
    )

    offset = 0
    fim = False
    primeira_iteracao = True

    while not fim:
        query = query_template.format(limit=bloco, offset=offset)
        df = pd.read_sql(query, conn)

        if df.empty:
            fim = True
        else:
            df.to_csv(
                output_csv,
                mode='w' if primeira_iteracao else 'a',
                index=False,
                header=primeira_iteracao
            )
            offset += bloco
            primeira_iteracao = False

    conn.close()


#Executa uma query no MySQL e retorna um DataFrame
def extrair_dados_mysql(
    query: str,
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 3306
) -> pd.DataFrame:

    conn = None
    try:
```

```
        conn = pymysql.connect(
            host=host,
            user=user,
            password=password,
            database=database,
            port=port
        )
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()



def extrair_dados_sqlserver(
    query: str,
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 1433
) -> pd.DataFrame:
    """Executa uma query no SQL Server e retorna um DataFrame."""
    conn = None
    try:
        conn_str = (
            f"DRIVER={{ODBC Driver 17 for SQL Server}};"
            f"SERVER={host},{port};"
            f"DATABASE={database};"
            f"UID={user};"
            f"PWD={password}"
        )
        conn = pyodbc.connect(conn_str)
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()

#EXTRACAO TABELA tg_bem
query_tg_bem = """
SELECT * FROM tg_bem ORDER BY id ASC LIMIT {limit} OFFSET {offset};
"""

extrair_em_blocos_postgres(
    query_template=query_tg_bem,
    host="xxxxx",
    user="xxxxxx",
    password="xxxxxxx",
```

```
    database="xxxxxxxx",
    bloco=50000,
    output_csv="util/tg_bem.csv"
)

#EXTRACAO TABELA tb_protecao_bem
query_tb_protecao_bem = "SELECT * FROM tb_protecao_bem;"
# Executa extração
df = extrair_dados_postgresql(
    query=query_tb_protecao_bem,
    host="xxxxxx",
    user="xxxxxx",
    password="xxxxxx",
    database="xxxxxx"
)
df.to_csv("util/tb_protecao_bem.csv", index=False)

#------------------------------------------------------------

#EXTRACAO TABELA tb_classificacao
query_tb_classificacao = "SELECT * FROM tb_classificacao;"
# Executa extração
df = extrair_dados_postgresql(
    query=query_tb_classificacao,
    host="xxxxxx",
    user="xxxxxx",
    password="xxxxxx",
    database="xxxxxx"
)
df.to_csv("util/tb_classificacao.csv", index=False)

#------------------------------------------------------------

#EXTRACAO TABELA tb_natureza
query_tb_natureza = "SELECT * FROM tb_natureza;"
# Executa extração
df = extrair_dados_postgresql(
    query=query_tb_natureza,
    host="xxxxxx",
    user="xxxxxx",
    password="xxxxxx",
    database="xxxxxx"
)
df.to_csv("util/tb_natureza.csv", index=False)

#------------------------------------------------------------

#EXTRACAO TABELA tb_forma_inscricao
query_tb_forma_inscricao = "SELECT * FROM tb_forma_inscricao;"
# Executa extração
```

```
df = extrair_dados_postgresql(
   query=query_tb_forma_inscricao,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tb_forma_inscricao.csv", index=False)

#------------------------------------------------------------

#EXTRACAO TABELA tb_uf
query_tg_uf = "SELECT * FROM tg_uf;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tg_uf,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tg_uf.csv", index=False)

#------------------------------------------------------------

#EXTRACAO TABELA tb_municipio
query_tg_municipio = "SELECT * FROM tg_municipio;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tg_municipio,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tg_municipio.csv", index=False)


#------------------------------------------------------------

#EXTRACAO TABELA tb_bem_contato
query_tb_bem_contato = "SELECT * FROM tb_bem_contato;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tb_bem_contato,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
```

```
df.to_csv("util/tb_bem_contato.csv", index=False)


#EXTRACAO TABELA tb_contato
query_tb_contato = "SELECT * FROM tb_contato;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tb_contato,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tb_contato.csv", index=False)

#EXTRACAO MURAL DE AGENTES Banco de dados mysql
query_mural_agentes = """
SELECT
   p.post_title,
   p.post_name AS item_slug,
   'agentes-organizacoes' AS colecao_slug,
   CONCAT('https://mural-observatorio.iphan.gov.br/agentes-organizacoes/',
p.post_name, '/') AS url_mural_agente,

   MAX(CASE WHEN pm.meta_key = '1123' THEN pm.meta_value END) AS
natureza_organizacao,
   MAX(CASE WHEN pm.meta_key = '1127' THEN pm.meta_value END) AS atuacao,
   MAX(CASE WHEN pm.meta_key = '1117' THEN pm.meta_value END) AS
id_bem_mural


FROM wp_posts p

LEFT JOIN wp_postmeta pm ON pm.post_id = p.ID

WHERE p.post_type = 'tnc_col_1109_item'

GROUP BY p.ID, p.post_title, p.post_name
ORDER BY p.post_title ASC;
"""

df = extrair_dados_mysql(
   query=query_mural_agentes,
   host="xxxxxxx",
   user="xxxxxxxx",
   password="xxxxxxx",
   database="xxxxxxxxx"
)

#Salva
```

```
df.to_csv("util/mural_agentes_wordpress.csv", index=False)

#EXTRACAO TABELA acao_fiscalizacao Banco de dados SQLserver FISCALIS
query_acao_fiscalizacao = "SELECT * FROM acao_fiscalizacao;"
# Executa extração
df = extrair_dados_sqlserver(
   query=query_acao_fiscalizacao,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/acao_fiscalizacao.csv", index=False)

#-----------Tratamento----------------------------------
def tratar_dados_sicg() -> pd.DataFrame:

   # Dicionarios para uf
   uf_para_regiao = {
      'AC': 'Norte', 'AP': 'Norte', 'AM': 'Norte', 'PA': 'Norte', 'RO': 'Norte', 'RR': 'Norte', 'TO':
'Norte',
      'AL': 'Nordeste', 'BA': 'Nordeste', 'CE': 'Nordeste', 'MA': 'Nordeste', 'PB': 'Nordeste',
      'PE': 'Nordeste', 'PI': 'Nordeste', 'RN': 'Nordeste', 'SE': 'Nordeste',
      'DF': 'Centro-Oeste', 'GO': 'Centro-Oeste', 'MT': 'Centro-Oeste', 'MS':
'Centro-Oeste',
      'ES': 'Sudeste', 'MG': 'Sudeste', 'RJ': 'Sudeste', 'SP': 'Sudeste',
      'PR': 'Sul', 'RS': 'Sul', 'SC': 'Sul'
   }

   estado_para_uf = {
      "Acre": "AC", "Alagoas": "AL", "Amapá": "AP", "Amazonas": "AM", "Bahia": "BA",
      "Ceará": "CE", "Distrito Federal": "DF", "Espírito Santo": "ES", "Goiás": "GO",
      "Maranhão": "MA", "Mato Grosso": "MT", "Mato Grosso do Sul": "MS", "Minas
Gerais": "MG",
      "Pará": "PA", "Paraíba": "PB", "Paraná": "PR", "Pernambuco": "PE", "Piauí": "PI",
      "Rio de Janeiro": "RJ", "Rio Grande do Norte": "RN", "Rio Grande do Sul": "RS",
      "Rondônia": "RO", "Roraima": "RR", "Santa Catarina": "SC", "São Paulo": "SP",
      "Sergipe": "SE", "Tocantins": "TO"
   }

   estados_map = {
      "Acre": "BR-AC", "AC": "BR-AC", "AL": "BR-AL", "Alagoas": "BR-AL", "Amapá":
"BR-AP", "AP": "BR-AP",
      "AM": "BR-AM", "Amazonas": "BR-AM", "BA": "BR-BA", "Bahia": "BR-BA", "Ceará":
"BR-CE", "CE": "BR-CE",
      "Espírito Santo": "BR-ES", "ES": "BR-ES", "Goiás": "BR-GO", "GO": "BR-GO",
"Maranhão": "BR-MA",
      "MA": "BR-MA", "Mato Grosso": "BR-MT", "MT": "BR-MT", "Mato Grosso do Sul":
"BR-MS", "MS": "BR-MS",
      "Minas Gerais": "BR-MG", "MG": "BR-MG", "Pará": "BR-PA", "PA": "BR-PA",
```

```
"Paraíba": "BR-PB",
    "PB": "BR-PB", "Paraná": "BR-PR", "PR": "BR-PR", "Pernambuco": "BR-PE", "PE":
"BR-PE",
    "Piauí": "BR-PI", "PI": "BR-PI", "Rio de Janeiro": "BR-RJ", "RJ": "BR-RJ",
    "Rio Grande do Norte": "BR-RN", "RN": "BR-RN", "Rio Grande do Sul": "BR-RS",
"RS": "BR-RS",
    "Rondônia": "BR-RO", "RO": "BR-RO", "Roraima": "BR-RR", "RR": "BR-RR", "Santa
Catarina": "BR-SC",
    "SC": "BR-SC", "São Paulo": "BR-SP", "SP": "BR-SP", "Sergipe": "BR-SE", "SE":
"BR-SE",
    "Tocantins": "BR-TO", "TO": "BR-TO", "Distrito Federal": "BR-DF", "DF": "BR-DF"
  }


  # Mapas de conversão
  mapa_conservacao = {
    0: 'Bom',
    1: 'Regular',
    2: 'Ruim',
    3: 'Péssimo'
  }

  mapa_preservacao = {
    0: 'Íntegro',
    1: 'Pouco Alterado',
    2: 'Muito Alterado',
    3: 'Descaracterizado'
  }
  # Dicionário de mapeamento protecao
  mapeamento_protecao = {
    "1": "Tombamento",
    "2": "Paisagem cultural chancelada",
    "3": "Inscrição na lista do patrimônio cultural ferroviário",
    "4": "Inscrição na lista do patrimônio cultural do Mercosul",
    "5": "Inscrição na lista do patrimônio cultural mundial",
    "6": "Registro de sítio arqueológico",
    "7": "Tombamento §5º do Art. 216 da Constituição Federal"
  }


  # Mapeamento bem por tipo
  mapa_bem_por_tipo = {
    "Tombamento": "Material",
    "Tombamento §5º do Art. 216 da Constituição Federal": "Material",
    "Inscrição na lista do patrimônio cultural ferroviário": "Ferroviário",
    "Registro de sítio arqueológico": "Arqueológico",
    "Tombamento/Inscrição na lista do patrimônio cultural ferroviário":
"Material/Ferroviário",
    "Tombamento/Registro de sítio arqueológico": "Material/Arqueológico"
  }
```

```
  #Mapeamento dos livros do tombo
  mapa_livros = {
    "belas artes": "Livro do Tombo de Belas Artes",
    "belas aplicadas": "Livro do Tombo de Belas Aplicadas",
    "historico": "Livro do Tombo Histórico",
    "arqueologico": "Livro do Tombo Arqueológico, Etnográfico e Paisagístico",
    "etnografico": "Livro do Tombo Arqueológico, Etnográfico e Paisagístico",
    "paisagistico": "Livro do Tombo Arqueológico, Etnográfico e Paisagístico",
    "arqueologico etnografico paisagistico": "Livro do Tombo Arqueológico,
Etnográfico e Paisagístico"
  }

  regex_livros = {
    "belas artes": re.compile(r"livro.*belas\s+artes", re.IGNORECASE),
    "belas aplicadas": re.compile(r"livro.*belas\s+aplicadas", re.IGNORECASE),
    "historico": re.compile(r"livro.*historico", re.IGNORECASE),
    "arqueologico": re.compile(r"livro.*arqueologico", re.IGNORECASE),
    "etnografico": re.compile(r"livro.*etnografico", re.IGNORECASE),
    "paisagistico": re.compile(r"livro.*paisagistico", re.IGNORECASE),
    "arqueologico etnografico paisagistico": re.compile(

r"livro.*(arqueologico.*etnografico.*paisagistico|paisagistico.*etnografico.*arqueol
ogico)",
       re.IGNORECASE
    ),
  }


    # Dicionário de mapeamento de id_tipo_bem → tipo
  mapa_tipo = {
    '1': "Coleção",
    '2': "Artefato",
    '3': "Acervo",
    '4': "Sítio",
    '5': "Obras de Engenharia",
    '6': "Conjunto Arquitetônico",
    '7': "Conjunto Urbano",
    '8': "Edificação",
    '9': "Jardim Histórico",
    '10': "Paisagem",
    '11': "Bem ou conjunto de bens arqueológicos móveis",
    '12': "Acervo ou Coleção"
  }

  cache_fuzzy = {}


  #Função extrai longitude/latitude e também formatar como lista string
  def tratar_ponto(ponto):
```

```
    match = re.match(r'POINT\s*\(\s*([-\d\.]+)\s+([-\d\.]+)\s*\)', str(ponto))
    if match:
        lon = float(match.group(1))
        lat = float(match.group(2))
        return f"[{lon}, {lat}]", lon, lat
    return "", None, None

def buscar_datas_multiplas(valor):
    if pd.isna(valor):
        return np.nan
    ids = str(valor).split('/')
    datas = [mapa_datas.get(id_.strip(), np.nan) for id_ in ids]
    datas_filtradas = [d for d in datas if pd.notna(d)]
    return '|'.join(datas_filtradas) if datas_filtradas else np.nan


#Funcao extrai ano
def extrair_ano(data_inscricao):
    if pd.isna(data_inscricao):
        return np.nan
    datas = [d.strip() for d in data_inscricao.split('|')]
    anos = []
    for d in datas:
        try:
            ano = pd.to_datetime(d, dayfirst=True).year
            anos.append(str(ano))
        except:
            continue
    return '|'.join(anos) if anos else np.nan

#Funcao extrai decada
def extrair_decada(data_inscricao):
    if pd.isna(data_inscricao):
        return np.nan
    datas = [d.strip() for d in data_inscricao.split('|')]
    decadas = []
    for d in datas:
        try:
            ano = pd.to_datetime(d, dayfirst=True).year
            decada = f"{(ano // 10) * 10}"
            decadas.append(decada)
        except:
            continue
    return '|'.join(decadas) if decadas else np.nan



#Função extrai os livros
def extrair_livros_completo(justificativa):
    if pd.isna(justificativa):
```

```
        return ""

    texto_norm = unidecode(justificativa.lower())
    encontrados = set()

    for chave, pattern in regex_livros.items():
        if pattern.search(texto_norm):
            encontrados.add(mapa_livros[chave])

    return "/".join(sorted(encontrados)) if encontrados else ""


#Função mapeia os agentes no mural
def mapear_agente_para_mural(nome):
    if nome in cache_fuzzy:
        return cache_fuzzy[nome]

    if not nome.strip():
        cache_fuzzy[nome] = (None, None, None)
        return cache_fuzzy[nome]

    resultado = process.extractOne(nome, titulos_mural,
scorer=fuzz.token_sort_ratio)
    if resultado and resultado[1] >= 70:
        linha = df_mural[df_mural['post_title'] == resultado[0]].iloc[0]
        cache_fuzzy[nome] = (
            linha['url_mural_agente'],
            linha['natureza_organizacao_agente'],
            linha['atuacao_agente']
        )
    else:
        cache_fuzzy[nome] = (None, None, None)

    return cache_fuzzy[nome]

#Função trata os campos multivalorados dos agentes
def aplicar_mapeamento_multivalorado(campo_nomes):
    nomes = campo_nomes.split('|') if pd.notna(campo_nomes) else []
    urls, naturezas, atuacoes = [], [], []
    for nome in nomes:
        url, natureza, atuacao = mapear_agente_para_mural(nome.strip())
        urls.append(url)
        naturezas.append(natureza)
        atuacoes.append(atuacao)
    return pd.Series([
        '|'.join([str(x) if pd.notna(x) else '' for x in urls]),
        '|'.join([str(x) if pd.notna(x) else '' for x in naturezas]),
        '|'.join([str(x) if pd.notna(x) else '' for x in atuacoes])
    ])
```

```
    df = pd.read_csv("util/tg_bem.csv", dtype={'ativo': str}, low_memory=False)

    # Converte a coluna 'ativo' para booleano: True se texto for 'true' (ignora
maiúsculas)
    df['ativo'] = df['ativo'].str.lower() == 'true'

    # Filtra apenas as linhas onde 'ativo' é True
    df = df[df['ativo'] == True]

    df.rename(columns={"identificacao_bem": "nome_bem"}, inplace=True)


    # Agora remove essas linhas do DataFrame original
    df = df[~df['nome_bem'].str.contains(r'(?<!\w)teste(?!\w)', case=False, na=False)]


    # Aplica os mapeamentos às colunas correspondentes
    df['tipo_estado_conservacao'] =
df['tipo_estado_conservacao'].map(mapa_conservacao)
    df['tipo_estado_preservacao'] =
df['tipo_estado_preservacao'].map(mapa_preservacao)



    # ----------------------------------------------
    df_relacao = pd.read_csv(
        'util/tb_protecao_bem.csv',
        encoding='utf-8',
        dtype={'id_bem': str, 'id_tipo_protecao': str, 'id_protecao_bem': str}
    )


    # Mapeia id_tipo_protecao para nomes descritivos
    df_relacao['ds_tipo_protecao'] =
df_relacao['id_tipo_protecao'].map(mapeamento_protecao)

    # Agrupa as descrições de proteção por id_bem, ordenando com "Tombamento"
primeiro
    df_protecoes_agrupadas = (
        df_relacao
        .dropna(subset=['ds_tipo_protecao'])
        .groupby('id_bem')['ds_tipo_protecao']
        .agg(lambda prot_list: '/'.join(
            sorted(set(prot_list), key=lambda x: (x != 'Tombamento', x))
        ))
        .reset_index()
    )
    df_protecoes_agrupadas.rename(columns={'ds_tipo_protecao':
```

```
'bem_tipo_protecao'}, inplace=True)

    # Agrupa também os id_protecao_bem por id_bem
    df_ids_protecao = (
        df_relacao
        .dropna(subset=['id_protecao_bem'])
        .groupby('id_bem')['id_protecao_bem']
        .agg(lambda ids: '/'.join(sorted(set(ids))))
        .reset_index()
    )

    # Junta os IDs com as descrições agregadas
    df_protecoes_agrupadas = df_protecoes_agrupadas.merge(df_ids_protecao,
on='id_bem', how='left')

    # Garante que ambos os lados do merge tenham o mesmo tipo de id_bem
    df['id_bem'] = df['id_bem'].astype(str)
    df_protecoes_agrupadas['id_bem'] = df_protecoes_agrupadas['id_bem'].astype(str)

    # Junta ao DataFrame principal
    df = df.merge(df_protecoes_agrupadas, on='id_bem', how='left')

    # ------------------------------------------------

    # Aplica a função e cria as colunas
    df['ponto_formatado'], df['longitude'], df['latitude'] =
zip(*df['ponto'].map(tratar_ponto))

    # Atualiza a coluna 'ponto' com o novo formato string
    df['ponto'] = df['ponto_formatado']
    df.drop(columns=['ponto_formatado'], inplace=True)

    # Cria a coluna 'url' com base no id_bem
    df['url'] = df['id_bem'].apply(lambda x:
f"https://sicg.iphan.gov.br/sicg/bem/visualizar/{x}")


    # Lê os arquivos de municípios e estados
    df_municipios = pd.read_csv('util/tg_municipio.csv', encoding='utf-8',
dtype={'id_municipio': str, 'id_uf': str})
    df_ufs = pd.read_csv('util/tg_uf.csv', encoding='utf-8', dtype={'id_uf': str})

    # Normaliza textos
    df_municipios['nome'] = df_municipios['nome'].str.strip()
    df_ufs['nome'] = df_ufs['nome'].str.strip()

    # Garante tipo compatível para merge com df
    df['id_municipio'] = df['id_municipio'].astype(str)
    df_municipios['id_municipio'] = df_municipios['id_municipio'].astype(str)
```

```
    # Junta df com municípios (pelo id_municipio)
    df = df.merge(df_municipios[['id_municipio', 'nome', 'id_uf']], on='id_municipio',
how='left')
    df.rename(columns={'nome': 'nome_municipio'}, inplace=True)

    # Junta com a tabela de estados
    df = df.merge(df_ufs[['id_uf', 'nome']], on='id_uf', how='left')
    df.rename(columns={'nome': 'uf_estado'}, inplace=True)

    # Extrai sigla da UF (ex: GO) a partir de id_uf
    df['uf'] = df['uf_estado'].map(estado_para_uf)

    # Adiciona coluna 'regiao'
    df['regiao'] = df['uf'].map(uf_para_regiao)

    # Adiciona coluna 'uf_iso'
    df['uf_iso'] = df['uf_estado'].map(estados_map)

    # Formata 'nome_municipio' como "Município - UF"
    df['nome_municipio'] = df['nome_municipio'] + ' - ' + df['uf']

    #-----------------------------------------------

    # Substitui os valores booleanos por texto
    df['valorado'] = df['valorado'].astype(str).replace({'True': 'Sim', 'False': 'Não'})


    # Aplica o mapeamento
    df['bem_por_tipo'] = df['bem_tipo_protecao'].map(mapa_bem_por_tipo)


    df['dt_cadastro'] = pd.to_datetime(df['dt_cadastro'], errors='coerce')
    df['dt_cadastro'] = df['dt_cadastro'].dt.strftime('%d/%m/%Y')


    df['no_logradouro'] = df['no_logradouro'].apply(
        lambda x: 'Não informado' if pd.isna(x) or (isinstance(x, str) and
x.strip().lower() == 'não informado') else x
    )

    df_datas = pd.read_csv('util/tb_forma_inscricao.csv', encoding='utf-8',
dtype={'id_protecao_bem': str})
    df_datas['dt_inscricao'] = pd.to_datetime(df_datas['dt_inscricao'], errors='coerce')

    # Cria o dicionário {id_protecao_bem: data_formatada}
    mapa_datas = df_datas.dropna(subset=['dt_inscricao']) \
        .drop_duplicates(subset=['id_protecao_bem']) \
        .set_index('id_protecao_bem')['dt_inscricao'] \
        .apply(lambda d: d.strftime('%d/%m/%Y')) \
        .to_dict()
```

```
    # Aplica no DataFrame principal
    df['data_inscricao'] = df['id_protecao_bem'].apply(buscar_datas_multiplas)


    # Aplica as funções no DataFrame
    df['ano'] = df['data_inscricao'].apply(extrair_ano)
    df['decada'] = df['data_inscricao'].apply(extrair_decada)


    #-----------Adiciona colunas classificacao e
natureza-------------------------------------

    # Lê os dados de apoio se ainda não tiver lido antes
    df_classif = pd.read_csv('util/tb_classificacao.csv', encoding='utf-8',
dtype={'id_classificacao': str, 'id_natureza': str})
    df_natureza = pd.read_csv('util/tb_natureza.csv', encoding='utf-8',
dtype={'id_natureza': str})

    # Cria os dicionários de mapeamento
    mapa_classificacao =
df_classif.set_index('id_classificacao')['ds_classificacao'].to_dict()
    mapa_id_natureza = df_classif.set_index('id_classificacao')['id_natureza'].to_dict()
    mapa_natureza = df_natureza.set_index('id_natureza')['ds_natureza'].to_dict()

    # Funções para mapear classificacao e natureza a partir de id_classificacao
    def mapear_classificacao(valor):
        if valor.isdigit() and valor in mapa_classificacao:
            return mapa_classificacao[valor]
        return np.nan  # se nao existir ou nao for válido

    def mapear_natureza(valor):
        if valor.isdigit() and valor in mapa_id_natureza:
            id_nat = mapa_id_natureza[valor]
            return mapa_natureza.get(id_nat, np.nan)
        return np.nan

    # Aplica os mapeamentos no seu DataFrame original
    df['classificacao'] =
df['id_classificacao'].fillna('').astype(str).apply(mapear_classificacao)
    df['natureza'] = df['id_classificacao'].fillna('').astype(str).apply(mapear_natureza)

    #----------------------Insere livros---------------

    #Lê o arquivo com as justificativas
    df_justificativas = pd.read_csv("util/tb_forma_inscricao.csv", dtype=str)

    #Limpa os IDs nos dois DataFrames para garantir merge correto
    df_justificativas['id_protecao_bem'] =
```

```
df_justificativas['id_protecao_bem'].str.replace('.0', '', regex=False).str.strip()
   df['id_protecao_bem'] = df['id_protecao_bem'].str.replace('.0', '',
regex=False).str.strip()




   #Aplica a extração de livros
   df_justificativas['livros'] =
df_justificativas['justificativa'].apply(extrair_livros_completo)

   #Agrupa por id_protecao_bem (caso haja múltiplas justificativas por bem)
   df_livros = df_justificativas.groupby('id_protecao_bem')['livros'] \
      .unique().apply(lambda x: '|'.join(sorted(set(x)))).reset_index()

   #Merge no df principal
   df = df.merge(df_livros, on='id_protecao_bem', how='left')

   #-------------------------Mapeia o
tipo-----------------------------------------------------


   # Garante que id_tipo_bem está como string para mapear corretamente
   df['id_tipo_bem'] = df['id_tipo_bem'].astype(str)

   # Cria a nova coluna 'tipo' com os valores mapeados
   df['tipo'] = df['id_tipo_bem'].map(mapa_tipo)


   #--------------------INSERE OS AGENTES---------------


   df_contato = pd.read_csv('util/tb_contato.csv', dtype=str)
   df_bem_contato = pd.read_csv('util/tb_bem_contato.csv', dtype=str)

   #Filtra apenas agentes jurídicos
   df_contato = df_contato[df_contato['fisica_juridica'] == '1']

   #Faz merge entre bem e contato, mantendo id_contato para juntar
   df_bem_contato = df_bem_contato.merge(
      df_contato[['id_contato', 'no_fisica_juridica', 'ds_email']],
      on='id_contato',
      how='left'
   )

   #Agrupa os dados por id_bem (agentes multivalorados com "|")
   df_agentes_agrupados = df_bem_contato.groupby('id_bem').agg({
      'id_contato': lambda x: '|'.join(sorted(set(x.dropna()))),
      'no_fisica_juridica': lambda x: '|'.join(sorted(set(x.dropna()))),
      'ds_email': lambda x: '|'.join(sorted(set(x.dropna())))
```

```
    }).reset_index()

    #Renomeia as colunas para o padrão final
    df_agentes_agrupados = df_agentes_agrupados.rename(columns={
        'id_contato': 'id_agente',
        'no_fisica_juridica': 'nome_agente'
    })

    # Adiciona tipo_agente
    df_agentes_agrupados['tipo_agente'] = 'Contato Bem Material'

    #Merge com o seu DataFrame principal (df)
    df = df.merge(df_agentes_agrupados, on='id_bem', how='left')

    #Preenche tipo_agente apenas quando houver agente
    df['tipo_agente'] = df['nome_agente'].apply(lambda x: 'Contato Bem Material' if
pd.notna(x) and x.strip() != '' else '')


    #Normaliza nome_agente: transforma em Title Case (letras maiúsculas só no
início das palavras)
    df['nome_agente'] = df['nome_agente'].apply(
        lambda x: x.title() if pd.notna(x) and isinstance(x, str) else x
    )
    #----------MURAL AGENTES-----------------------

    df_mural = pd.read_csv("util/mural_agentes_wordpress.csv", dtype=str)
    df_mural = df_mural.rename(columns={
        'natureza_organizacao': 'natureza_organizacao_agente',
        'atuacao': 'atuacao_agente'
    })
    df_mural['post_title'] = df_mural['post_title'].fillna('').astype(str).str.strip()

    #Prepara agentes no df
    df['nome_agente'] = df['nome_agente'].fillna('').astype(str).str.strip()

    #Lista de títulos disponíveis
    titulos_mural = df_mural['post_title'].tolist()



    # Aplica ao DataFrame
    df[['url_mural_agente', 'natureza_organizacao_agente', 'atuacao_agente']] =
df['nome_agente'].apply(aplicar_mapeamento_multivalorado)


    #----------------------INSERE AS ACOES---------------

    #Ações FISCALIS
    df_acoes = pd.read_csv("util/acao_fiscalizacao.csv", dtype=str)
```

```
   #Conta quantas vezes cada cod_bem aparece
   contagem_acoes =
df_acoes['cod_bem'].value_counts().rename_axis('co_iphan').reset_index(name='total
_acoes')

   #Adiciona a coluna tipo_acao
   contagem_acoes['tipo_acao'] = "Ação do processo institucional preservação:
Fiscalização"

   #Faz merge com o DataFrame principal
   df = df.merge(contagem_acoes, on='co_iphan', how='left')

   #Se quiser, preenche NaN com vazio para manter o padrão
   df['total_acoes'] = df['total_acoes'].fillna('')
   df['tipo_acao'] = df['tipo_acao'].fillna('')


   df['possui_agentes_acoes'] = df.apply(
      lambda row: 'Sim' if pd.notna(row.get('tipo_agente')) and row['tipo_agente'] != ''
                  or pd.notna(row.get('tipo_acao')) and row['tipo_acao'] != ''
               else '',
      axis=1
   )

   #-----------------Final----------------------------------------------

   #Tratamento de colunas
   df['id_bem'] = df['id_bem'].fillna('').astype(str).str.replace('.0', '', regex=False)
   df['id_bem_pai'] = df['id_bem'].fillna('').astype(str).str.replace('.0', '', regex=False)
   df['nu_cep'] = df['nu_cep'].fillna('').astype(str).str.replace('.0', '', regex=False)
   df['total_acoes'] = df['total_acoes'].fillna('').astype(str).str.replace('.0', '',
regex=False)
   #Lista de colunas desejadas
   colunas_finais = [
      "co_iphan", "id_bem", "bem_tipo_protecao", "bem_por_tipo", "id_protecao_bem",
      "id_bem_pai", "nome_bem", "no_logradouro", "nu_cep",
"local_especifico","tipo","natureza","classificacao",
      "tipo_estado_conservacao", "tipo_estado_preservacao", "ds_tipo_propriedade",
"valorado", "ponto",
      "longitude", "latitude", "url", "regiao", "uf_estado", "uf", "uf_iso",
"nome_municipio",
      "dt_cadastro", "data_inscricao", "ano",
"decada","livros","id_agente","nome_agente","ds_email","tipo_agente",

"url_mural_agente","natureza_organizacao_agente","atuacao_agente","tipo_acao","tota
l_acoes","possui_agentes_acoes"
   ]

   #Filtra o DataFrame para manter somente essas colunas
```

```
    df = df[colunas_finais]
    df = df.applymap(lambda x: str(x).replace("""", " ") if pd.notna(x) else x)

    #Exibe número total de linhas no DataFrame
    #print(f"Total de linhas no DataFrame: {len(df)}")
    return df  # DataFrame tratado

#----------Carrega os dados no armazém de dados----------
def carregando_dados_armazem():
    start_time = time.time()

    df_final = pd.read_csv("util/dados_sicg_tratados.csv")

    conn = psycopg2.connect(
        host="xxxxx",
        dbname="xxxxxxxx",
        user="xxxxxxx",
        password="xxxxxxx"
    )
    cursor = conn.cursor()
    conn.autocommit = False

    id_colecao = 2
    id_metadado_co_iphan = 25

    coluna_para_id_metadado = {
        "co_iphan": 25, "id_bem": 26, "bem_tipo_protecao": 27, "bem_por_tipo": 28,
        "id_protecao_bem": 29, "id_bem_pai": 30, "nome_bem": 31, "no_logradouro": 32,
        "nu_cep": 33, "local_especifico": 34, "tipo": 35, "natureza": 36,
        "classificacao": 37, "tipo_estado_conservacao": 38, "tipo_estado_preservacao": 39,
        "ds_tipo_propriedade": 40, "valorado": 41, "ponto": 42, "longitude": 43,
        "latitude": 44, "url": 45, "regiao": 46, "uf_estado": 47, "uf": 48,
        "uf_iso": 49, "nome_municipio": 50, "dt_cadastro": 51, "data_inscricao": 52,
        "ano": 53, "decada": 54, "livros": 55, "id_agente": 56, "nome_agente": 57,
        "ds_email": 58, "tipo_agente": 59, "url_mural_agente": 60,
        "natureza_organizacao_agente": 61, "atuacao_agente": 62,
        "tipo_acao": 63, "total_acoes": 64, "possui_agentes_acoes": 65
    }

    # Carrega co_iphan existentes
    cursor.execute("""
        SELECT i.id, v.value
        FROM obs_valor_metadado v
        JOIN obs_itens i ON v.id_valor = i.id
        WHERE v.id_metadado = %s AND i.id_colecao = %s
    """, (id_metadado_co_iphan, id_colecao))
    co_iphan_existentes = {valor.strip(): item_id for item_id, valor in cursor.fetchall()}

    # Carrega todos os valores já inseridos
    cursor.execute("SELECT id_valor, id_metadado FROM obs_valor_metadado")
```

```
    valores_existentes = {(id_valor, id_metadado) for id_valor, id_metadado in
cursor.fetchall()}

    itens_atualizados = set()
    insert_valores = []
    update_valores = []
    novos_itens = []

    for i, row in df_final.iterrows():
        co_iphan = str(row.get("co_iphan")).strip()
        if not co_iphan or co_iphan.lower() == 'nan':
            continue

        if co_iphan in co_iphan_existentes:
            id_item = co_iphan_existentes[co_iphan]
            item_ja_existente = True
        else:
            post_title = row.get("nome_bem") or "Item sem título"
            cursor.execute("""
                INSERT INTO obs_itens (id_colecao, post_title)
                VALUES (%s, %s)
                RETURNING id
            """, (id_colecao, post_title))
            id_item = cursor.fetchone()[0]
            co_iphan_existentes[co_iphan] = id_item
            item_ja_existente = False

            insert_valores.append((id_item, id_metadado_co_iphan, co_iphan))

        for col, id_metadado in coluna_para_id_metadado.items():
            if col == "co_iphan":
                continue

            valor = row.get(col)
            if pd.isna(valor) or str(valor).strip() == "":
                continue

            valor_str = str(valor).replace("""", " ").strip()
            chave = (id_item, id_metadado)

            if chave in valores_existentes:
                #update_valores.append((valor_str, id_item, id_metadado))
                cursor.execute("""
                    SELECT value FROM obs_valor_metadado
                    WHERE id_valor = %s AND id_metadado = %s
                """, (id_item, id_metadado))
                valor_atual = cursor.fetchone()

                if valor_atual and valor_atual[0] != valor_str:
                    update_valores.append((valor_str, id_item, id_metadado))
```

```
            if item_ja_existente:
                itens_atualizados.add(id_item)

        else:
            insert_valores.append((id_item, id_metadado, valor_str))
            valores_existentes.add(chave)

    # A cada 1000 linhas, faz o flush
    if i % 1000 == 0 and i > 0:
        if insert_valores:
            cursor.executemany("""
                INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
                VALUES (%s, %s, %s)
            """, insert_valores)
            insert_valores.clear()

        if update_valores:
            cursor.executemany("""
                UPDATE obs_valor_metadado
                SET value = %s
                WHERE id_valor = %s AND id_metadado = %s
            """, update_valores)
            update_valores.clear()

        conn.commit()
        print(f"Processadas {i} linhas...")

# Insere o que sobrou no final
if insert_valores:
    cursor.executemany("""
        INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
        VALUES (%s, %s, %s)
    """, insert_valores)

if update_valores:
    cursor.executemany("""
        UPDATE obs_valor_metadado
        SET value = %s
        WHERE id_valor = %s AND id_metadado = %s
    """, update_valores)

conn.commit()
cursor.close()
conn.close()

print("Processamento concluído.")
print(f"Total de itens atualizados: {len(itens_atualizados)}")
elapsed_seconds = round(time.time() - start_time, 2)
print(f"Tempo total de execução: {elapsed_seconds} segundos")
```

```
if __name__ == "__main__":
    # Etapa 1: Extração (realiza a extracao no inicio)

    # Etapa 2: Tratamento
    df_tratado = tratar_dados_sicg()
    df_tratado.to_csv("util/dados_sicg_tratados.csv", index=False)

    # Etapa 3: Carga
    carregando_dados_armazem()
```

Fonte: Elaborado pelos autores (2025).

i) View implementada para o Indicador de Bens Culturais

A view vw_indicador_bens foi implementada no armazém de dados para unificar e padronizar os dados de duas coleções distintas: 1) Bens culturais imateriais registrados e 2) Bens culturais materiais, ferroviários, arqueológicos, armazenadas no armazém. Por ser materializada, a view armazena os dados já processados, o que garante maior eficiência em consultas e visualizações, nos dashboards do Apache Superset. Ela consolida os metadados estruturados de forma dinâmica e multivalorada, tratando as diferenças entre as coleções e facilitando a análise posterior, especialmente em ferramentas como o Apache Superset. Principais pontos da view:

- Normalização de estruturas diferentes: Campos exclusivos de cada coleção são tratados com NULL onde não se aplicam.
- Os metadados multivalorados das coleções são expandidos com o objetivo de representar corretamente os dados nos dashboards. Para garantir uma contagem precisa das informações, utilizamos o metadado co_iphan, presente em ambas as coleções, como controle em operações de COUNT(DISTINCT) quando necessário.
    - Para a coleção 1, os campos uf, uf_estado, uf_iso, regiao e nome_municipio também são “expandidos” corretamente, mantendo a correspondência posicional.
    - Para a coleção 2, os campos ano, decada e data_inscricao são “expandidos” corretamente, mantendo a correspondência posicional.

A seguir, apresenta-se o código SQL usado para criação da view vw_indicador_bens, implementada no Armazém de Dados.

Quadro 5 - View vw_indicador_bens

```
CREATE OR REPLACE VIEW vw_indicador_bens AS
WITH colecao1 AS (
  SELECT
    i.id AS id_item,
    MAX(CASE WHEN m.nome = 'co_iphan'          THEN v.value END) AS co_iphan,
    MAX(CASE WHEN m.nome = 'uf'                THEN v.value END) AS uf,
    MAX(CASE WHEN m.nome = 'uf_estado'         THEN v.value END) AS uf_estado,
    MAX(CASE WHEN m.nome = 'uf_iso'            THEN v.value END) AS uf_iso,
    MAX(CASE WHEN m.nome = 'regiao'            THEN v.value END) AS regiao,
    MAX(CASE WHEN m.nome = 'nome_municipio'    THEN v.value END) AS
nome_municipio,
    MAX(CASE WHEN m.nome = 'url'               THEN v.value END) AS url,
    NULL::text AS url_mural_agente,
    MAX(CASE WHEN m.nome = 'nome_bem'          THEN v.value END) AS
nome_bem,
    NULL::text AS id_bem,
    MAX(CASE WHEN m.nome = 'livros'            THEN v.value END) AS livros,
    MAX(CASE WHEN m.nome = 'denominacao'       THEN v.value END) AS
denominacao,
    MAX(CASE WHEN m.nome = 'data_registro'     THEN v.value END) AS
data_registro,
    MAX(CASE WHEN m.nome = 'abrangencia_registro' THEN v.value END) AS
abrangencia_registro,
    MAX(CASE WHEN m.nome = 'nome_agente'       THEN v.value END) AS
nome_agente,
    MAX(CASE WHEN m.nome = 'ponto_geo'         THEN v.value END) AS ponto,
    MAX(CASE WHEN m.nome = 'longitude'         THEN v.value END) AS longitude,
    MAX(CASE WHEN m.nome = 'latitude'          THEN v.value END) AS latitude,
    MAX(CASE WHEN m.nome = 'bem_tipo_protecao'  THEN v.value END) AS
bem_tipo_protecao,
    MAX(CASE WHEN m.nome = 'bem_por_tipo'      THEN v.value END) AS
bem_por_tipo,
    MAX(CASE WHEN m.nome = 'classificacao'     THEN v.value END) AS
classificacao,
    MAX(CASE WHEN m.nome = 'ano'               THEN v.value END) AS ano,
    MAX(CASE WHEN m.nome = 'decada'            THEN v.value END) AS decada,
    MAX(CASE WHEN m.nome = 'mes_registro_nome'  THEN v.value END) AS
mes_registro_nome,
    MAX(CASE WHEN m.nome = 'tipo_agente'       THEN v.value END) AS
tipo_agente,
    MAX(CASE WHEN m.nome = 'ds_email'          THEN v.value END) AS ds_email,
    MAX(CASE WHEN m.nome = 'possui_agentes_acoes' THEN v.value END) AS
possui_agentes_acoes,
    NULL::text AS no_logradouro,
    NULL::text AS local_especifico,
    NULL::text AS tipo,
    NULL::text AS natureza,
    NULL::text AS tipo_estado_conservacao,
    NULL::text AS tipo_estado_preservacao,
```

```
    NULL::text AS tipo_propriedade,
    NULL::text AS valorado,
    NULL::text AS dt_cadastro,
    NULL::text AS natureza_organizacao_agente,
    NULL::text AS atuacao_agente,
    NULL::text AS total_acoes,
    NULL::text AS tipo_acao,
    'colecao_1' AS origem
  FROM obs_itens i
  LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
  LEFT JOIN obs_metadados m ON v.id_metadado = m.id
  WHERE i.id_colecao = 1
  GROUP BY i.id
),
colecao2 AS (
  SELECT
    i.id AS id_item,
    MAX(CASE WHEN m.nome = 'co_iphan' THEN v.value END) AS co_iphan,
    MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
    MAX(CASE WHEN m.nome = 'uf_estado' THEN v.value END) AS uf_estado,
    MAX(CASE WHEN m.nome = 'uf_iso' THEN v.value END) AS uf_iso,
    MAX(CASE WHEN m.nome = 'regiao' THEN v.value END) AS regiao,
    MAX(CASE WHEN m.nome = 'nome_municipio' THEN v.value END) AS
nome_municipio,
    MAX(CASE WHEN m.nome = 'url' THEN v.value END) AS url,
    MAX(CASE WHEN m.nome = 'url_mural_agente' THEN v.value END) AS
url_mural_agente,
    MAX(CASE WHEN m.nome = 'nome_bem' THEN v.value END) AS nome_bem,
    MAX(CASE WHEN m.nome = 'id_bem' THEN v.value END) AS id_bem,
    MAX(CASE WHEN m.nome = 'livros' THEN v.value END) AS livros,
    NULL::text AS denominacao,
    MAX(CASE WHEN m.nome = 'data_inscricao' THEN v.value END) AS
data_registro,
    NULL::text AS abrangencia_registro,
    MAX(CASE WHEN m.nome = 'nome_agente' THEN v.value END) AS
nome_agente,
    MAX(CASE WHEN m.nome = 'ponto' THEN v.value END) AS ponto,
    MAX(CASE WHEN m.nome = 'longitude' THEN v.value END) AS longitude,
    MAX(CASE WHEN m.nome = 'latitude' THEN v.value END) AS latitude,
    MAX(CASE WHEN m.nome = 'bem_tipo_protecao' THEN v.value END) AS
bem_tipo_protecao,
    MAX(CASE WHEN m.nome = 'bem_por_tipo' THEN v.value END) AS
bem_por_tipo,
    MAX(CASE WHEN m.nome = 'classificacao' THEN v.value END) AS classificacao,
    MAX(CASE WHEN m.nome = 'ano' THEN v.value END) AS ano,
    MAX(CASE WHEN m.nome = 'decada' THEN v.value END) AS decada,
    NULL::text AS mes_registro_nome,
    MAX(CASE WHEN m.nome = 'tipo_agente' THEN v.value END) AS tipo_agente,
    MAX(CASE WHEN m.nome = 'ds_email' THEN v.value END) AS ds_email,
    MAX(CASE WHEN m.nome = 'possui_agentes_acoes' THEN v.value END) AS
```

```
possui_agentes_acoes,
    MAX(CASE WHEN m.nome = 'no_logradouro' THEN v.value END) AS
no_logradouro,
    MAX(CASE WHEN m.nome = 'local_especifico' THEN v.value END) AS
local_especifico,
    MAX(CASE WHEN m.nome = 'tipo' THEN v.value END) AS tipo,
    MAX(CASE WHEN m.nome = 'natureza' THEN v.value END) AS natureza,
    MAX(CASE WHEN m.nome = 'tipo_estado_conservacao' THEN v.value END) AS
tipo_estado_conservacao,
    MAX(CASE WHEN m.nome = 'tipo_estado_preservacao' THEN v.value END) AS
tipo_estado_preservacao,
    MAX(CASE WHEN m.nome = 'ds_tipo_propriedade' THEN v.value END) AS
tipo_propriedade,
    MAX(CASE WHEN m.nome = 'valorado' THEN v.value END) AS valorado,
    MAX(CASE WHEN m.nome = 'dt_cadastro' THEN v.value END) AS dt_cadastro,
    MAX(CASE WHEN m.nome = 'natureza_organizacao_agente' THEN v.value END)
AS natureza_organizacao_agente,
    MAX(CASE WHEN m.nome = 'atuacao_agente' THEN v.value END) AS
atuacao_agente,
    MAX(CASE WHEN m.nome = 'total_acoes' THEN v.value END) AS total_acoes,
    MAX(CASE WHEN m.nome = 'tipo_acao' THEN v.value END) AS tipo_acao,
    'colecao_2' AS origem
  FROM obs_itens i
  LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
  LEFT JOIN obs_metadados m ON v.id_metadado = m.id
  WHERE i.id_colecao = 2
  GROUP BY i.id
),
uniao AS (
  SELECT * FROM colecao1
  UNION ALL
  SELECT * FROM colecao2
)
SELECT
  ROW_NUMBER() OVER () AS id,
  u.id_item,
  CASE WHEN u.origem = 'colecao_1' THEN uf_explodido ELSE u.uf END AS uf,
  CASE WHEN u.origem = 'colecao_1' THEN uf_estado_explodido ELSE u.uf_estado
END AS uf_estado,
  CASE WHEN u.origem = 'colecao_1' THEN uf_iso_explodido ELSE u.uf_iso END AS
uf_iso,
  CASE WHEN u.origem = 'colecao_1' THEN regiao_explodida ELSE u.regiao END AS
regiao,
  CASE WHEN u.origem = 'colecao_1' THEN municipio_explodido ELSE
u.nome_municipio END AS nome_municipio,
  u.co_iphan,
  u.url,
  u.url_mural_agente,
  u.nome_bem,
  u.id_bem,
```

```
  u.livros,
  u.denominacao,
  u.abrangencia_registro,
  u.nome_agente,
  u.ponto,
  u.longitude,
  u.latitude,
  u.bem_tipo_protecao,
  u.bem_por_tipo,
  u.classificacao,
  u.mes_registro_nome,
  u.tipo_agente,
  u.ds_email,
  u.possui_agentes_acoes,
  u.no_logradouro,
  u.local_especifico,
  u.tipo,
  u.natureza,
  u.tipo_estado_conservacao,
  u.tipo_estado_preservacao,
  u.tipo_propriedade,
  u.valorado,
  u.dt_cadastro,
  u.natureza_organizacao_agente,
  u.atuacao_agente,
  u.total_acoes,
  u.tipo_acao,
  u.origem,
  CASE WHEN u.origem = 'colecao_2' THEN e.ano ELSE u.ano END AS ano,
  CASE WHEN u.origem = 'colecao_2' THEN e.decada ELSE u.decada END AS decada,
  CASE WHEN u.origem = 'colecao_2' THEN e.data_inscricao ELSE u.data_registro
END AS data_inscricao
FROM uniao u
LEFT JOIN LATERAL (
  SELECT
    a.value AS ano,
    d.value AS decada,
    i.value AS data_inscricao
  FROM
    unnest(string_to_array(COALESCE(u.ano, ''), '|')) WITH ORDINALITY AS a(value,
ord)
  JOIN
    unnest(string_to_array(COALESCE(u.decada, ''), '|')) WITH ORDINALITY AS
d(value, ord) ON a.ord = d.ord
  JOIN
    unnest(string_to_array(COALESCE(u.data_registro, ''), '|')) WITH ORDINALITY
AS i(value, ord) ON a.ord = i.ord
) e ON u.origem = 'colecao_2'
LEFT JOIN LATERAL (
  SELECT
```

```
    uf.value AS uf_explodido,
    uf_estado.value AS uf_estado_explodido,
    uf_iso.value AS uf_iso_explodido,
    regiao.value AS regiao_explodida,
    municipio.value AS municipio_explodido
  FROM
    unnest(string_to_array(COALESCE(u.uf, ''), '|')) WITH ORDINALITY AS uf(value,
ord)
  JOIN unnest(string_to_array(COALESCE(u.uf_estado, ''), '|')) WITH ORDINALITY
AS uf_estado(value, ord) ON uf.ord = uf_estado.ord
  JOIN unnest(string_to_array(COALESCE(u.uf_iso, ''), '|')) WITH ORDINALITY AS
uf_iso(value, ord) ON uf.ord = uf_iso.ord
  JOIN unnest(string_to_array(COALESCE(u.regiao, ''), '|')) WITH ORDINALITY AS
regiao(value, ord) ON uf.ord = regiao.ord
  JOIN unnest(string_to_array(COALESCE(u.nome_municipio, ''), '|')) WITH
ORDINALITY AS municipio(value, ord) ON uf.ord = municipio.ord
) c1 ON u.origem = 'colecao_1';
```

Fonte: elaborado pelos autores (2025).

j) Consulta SQL implementada no Apache Superset para gerar o DATASET do Indicador de Bens Culturais

O select abaixo é responsável por recuperar os dados da view vw_indicador_bens, utilizada como base para visualizações no Superset. Além dos metadados consolidados, o select também trata:

- Geração de links clicáveis para os campos url e url_mural_agente.
    - url: convertido em um link HTML único.
    - url_mural_agente: sendo um campo multivalorado (valores separados por "|"), é transformado em múltiplos links HTML no mesmo campo, separados por |.

Esse select é usado diretamente na criação do Dataset no Superset, para os dashboards de indicadores. A seguir, apresenta-se a consulta SQL da view vw_indicador_bens no Superset.

Quadro 6 - Consulta Sql Da View Vw_indicador_bens

```
SELECT
  *,

  -- Remove o .0 de total_acoes
  TRIM(TRAILING '.0' FROM total_acoes) AS total_acoes_normalizado,

  -- Link da URL principal
```

```
  CASE
    WHEN url IS NOT NULL AND url != ''
    THEN CONCAT('<a href="', url, '" target="_blank">Acessar</a>')
    ELSE NULL
  END AS link_url,

  -- Link da URL mural agente (com separação por pipe, sem explodir a linha)
  CASE
    WHEN url_mural_agente IS NOT NULL AND url_mural_agente != ''
    THEN (
      SELECT string_agg('<a href="' || trim(valor) || '" target="_blank">Mural</a>', '
| ')
      FROM unnest(string_to_array(url_mural_agente, '|')) AS valor
    )
    ELSE NULL
  END AS link_url_mural_agente

FROM vw_indicador_bens;
```

Fonte: elaborado pelos autores (2025).

k) Implementação da POP-UP no Superset - Bens culturais

Os códigos JavaScript utilizados na implementação da POP-UP no mapa de Bens Culturais, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator

Quadro 7 - Configuração do JavaScript tooltip para o mapa de bens culturais

```
d => `
 <div>
  <strong>${d.object?.extraProps?.nome_bem}</strong>
 </div>
 <div>
  Agente: <strong>${d.object?.extraProps?.nome_agente ?
d.object.extraProps.nome_agente : ''}</strong>
 </div>
 <div>
  Tipo de Agente: <strong>${d.object?.extraProps?.tipo_agente ?
d.object.extraProps.tipo_agente : ''}</strong>
 </div>
 <div>
  Total de Ações do Processo Institucional de Preservação (Fiscalização):
<strong>${d.object?.extraProps?.total_acoes_normalizado ?
d.object.extraProps.total_acoes_normalizado : ''}</strong>
 </div>
 <div>
  Município: <strong>${d.object?.extraProps?.nome_municipio ?
```

```
d.object.extraProps.nome_municipio : ''}</strong>
  </div>
  <div>
   Estado UF: <strong>${d.object?.extraProps?.uf ? d.object.extraProps.uf :
''}</strong>
  </div>
  <div>
   Região: <strong>${d.object?.extraProps?.regiao ? d.object.extraProps.regiao :
''}</strong>
  </div>
  <div>
   Tipo do Bem: <strong>${d.object?.extraProps?.bem_por_tipo ?
d.object.extraProps.bem_por_tipo : ''}</strong>
  </div>
  <div>
   Tipo de Proteção: <strong>${d.object?.extraProps?.bem_tipo_protecao ?
d.object.extraProps.bem_tipo_protecao : ''}</strong>
  </div>
  <div>
   Ano de Inscrição/Registro: <strong>${d.object?.extraProps?.ano ?
d.object.extraProps.ano : ''}</strong>
  </div>
  <div>
   Clique no ponto e saiba mais
  </div>
`
```

Fonte: Elaborado pelos autores (2025).

- Configuração da POP-UP em: JavaScript onClick href

Quadro 8 - Configuração do JavaScript onClick para o mapa de bens culturais

```
d => d.object.extraProps.url
```

Fonte: Elaborado pelos autores (2025).

3.1.3.1.2 Indicador - Agentes

Foi desenvolvido o script de automação, para o indicador de Agentes, que inclui: 1) Instituições Parcerias, fonte: BCR, 2) Instituições Executoras, fonte: INRC 3) Contato de Bem Material, fonte: SICG e 4) Instituições Identificadas, fonte: SICG.

a) Script de Automação do Indicador de Agentes

Os processos descritos a seguir foram implementados como parte do fluxo de integração dos dados do indicador de agentes no armazém de dados.

b) Extração de dados de indicadores de agentes

Foram desenvolvidas quatro funções para realizar a extração de dados a partir de diferentes bancos relacionais: PostgreSQL, MySQL e SQL Server.

- PostgreSQL – Base de dados SICG

A função extrair_dados_postgresql() permite a extração direta de tabelas do banco SICG. As principais tabelas extraídas incluem: tb_bem_contato, tb_contato, tb_tipo_logradouro e tg_instituicao.

- MySQL – Base de dados INRC

A função extrair_dados_mysql() realiza a extração da coleção INRC: 2.1 - Projeto de Identificação.

c) Implementação do Tratamento

Após a extração o script realiza o tratamento dos dados extraídos. A seguir, apresentam-se os principais passos:

- Para os agentes: instituições executoras
  - ❖ Remove linhas com nome_projeto nulo ou vazio.
  - ❖ Elimina duplicatas com base em nome_projeto e nome_agente.
  - ❖ Separa a sigla da unidade federativa (UF) da coluna huf_estado, criando também as colunas uf_iso e regiao.
  - ❖ Define o tipo do agente como "Instituição Executora".
  - ❖ Normaliza os identificadores (id_agente) com prefixo INRC_.
  - ❖ Remove colunas irrelevantes para o carregamento.
  - ❖ Substitui aspas simples em textos por espaços.
  - ❖ Padroniza os campos ano_inicio e ano_finalizacao.
- Para os agentes: instituições parceiras
  - ❖ Agrupa os registros por nome_agente, consolidando colunas multivaloradas (ex.: nome_bem, uf_estado).

- ❖ Utiliza o algoritmo de similaridade textual (fuzz.ratio) para encontrar correspondências com instituições da base de referência.
- ❖ Normaliza os identificadores (id_agente) com prefixo INRC_.
- ❖ Extrai coordenadas de latitude e longitude a partir da coluna geometria.
- ❖ Reorganiza as colunas e remove as colunas não utilizadas.

- Para os agentes: contato bem material
  - ❖ Filtra apenas contatos do tipo "Jurídica" e remove entradas de teste.
  - ❖ Mapeia códigos categóricos para valores descritivos (ex.: setor público, área de atuação).
  - ❖ Realiza join com as tabelas auxiliares de município e UF para preencher informações de localização (uf, regiao, uf_iso).
  - ❖ Formata números de telefone e remove colunas desnecessárias.
  - ❖ Mapeia os bens vinculados ao contato, preenchendo nome, tipo e proteção do bem.
  - ❖ Normaliza os identificadores (id_agente) com prefixo SICG_ e organiza os dados para carregamento.

- Para os agentes: instituições identificadas
  - ❖ Elimina registros duplicados com base em similaridade textual de nomes.
  - ❖ Extrai latitude e longitude da coluna geometria.
  - ❖ Formata a geometria no padrão de visualização do Superset.
  - ❖ Realiza o mapeamento da coluna id_tipo_instituicao para coluna ds_tipo_instituicao, por exemplo: 1 para "Associação de detentores", 2 para "Instituição privada sem fins lucrativos (ONG, OSCIP, Cooperativas, etc)".
  - ❖ Normaliza os campos booleanos (ativo, st_participe, st_referencia) como "Sim" ou "Não".
  - ❖ Remove aspas simples.
  - ❖ Normaliza os identificadores (id_agente) com prefixo INSTITUICAO_ e organiza os dados para carregamento.

d) Inserção/Atualização no Armazém de Dados

Após o tratamento dos dados, é realizada a etapa de inserção ou atualização dos valores dos metadados no armazém de dados, utilizando conexão com banco do armazém PostgreSQL. Este processo garante que os dados tratados sejam devidamente sincronizados com o armazém de dados, mantendo atualizações incrementais, controle de duplicidade via id_agente e preservando os metadados associados a cada item das coleções.

Abaixo, apresenta-se o script completo desenvolvido para o indicador de agentes.

Quadro 9 - Script Completo - Agentes

```
#!/usr/bin/env python
# coding: utf-8

#Indicador Agentes - INRC, BCR e SICG

#Bibliotecas
import numpy as np
import pandas as pd
import pymysql
import re
from unidecode import unidecode
import psycopg2
import unicodedata
from fuzzywuzzy import process, fuzz
import pyodbc
import time
#--------------Extracao nas bases de dados-----------
#Executa uma query no MySQL e retorna um DataFrame
def extrair_dados_mysql(
    query: str,
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 3306
) -> pd.DataFrame:

    conn = None
    try:
        conn = pymysql.connect(
            host=host,
            user=user,
            password=password,
            database=database,
            port=port
        )
        df = pd.read_sql(query, conn)
```

```
        return df
    finally:
        if conn:
            conn.close()


#Executa uma query no SQLserver e retorna um DataFrame
def extrair_dados_sqlserver(
    query: str,
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 1433
) -> pd.DataFrame:
    """Executa uma query no SQL Server e retorna um DataFrame."""
    conn = None
    try:
        conn_str = (
            f"DRIVER={{ODBC Driver 17 for SQL Server}};"
            f"SERVER={host},{port};"
            f"DATABASE={database};"
            f"UID={user};"
            f"PWD={password}"
        )
        conn = pyodbc.connect(conn_str)
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()

#Executa uma query no PostgreSQL e retorna um DataFrame
def extrair_dados_postgresql(query: str, host: str, user: str, password: str, database:
str, port: int = 5432) -> pd.DataFrame:
    import psycopg2
    conn = None
    try:
        conn = psycopg2.connect(
            host=host,
            user=user,
            password=password,
            dbname=database,
            port=port
        )
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()
```

```
#EXTRACAO da Coleção INRC: 2.1 - Projeto de Identificação
query_inrc = """
SELECT
  p.ID,
  p.post_title,
  p.post_name AS item_slug,
  '2-1-projetos' AS colecao_slug,
  CONCAT('https://inrc.iphan.gov.br/2-1-projetos/', p.post_name, '/') AS url,

  MAX(CASE WHEN pm.meta_key = '37722' THEN pm.meta_value END) AS
nome_projeto,
  MAX(CASE WHEN pm.meta_key = '83787' THEN pm.meta_value END) AS
nome_agente,
  MAX(CASE WHEN pm.meta_key = '83790' THEN pm.meta_value END) AS
ano_inicio,
  MAX(CASE WHEN pm.meta_key = '83793' THEN pm.meta_value END) AS
ano_finalizacao,
  MAX(CASE WHEN pm.meta_key = '83803' THEN pm.meta_value END) AS
valor_orcamentario,

  GROUP_CONCAT(DISTINCT t1.name ORDER BY t1.name ASC SEPARATOR ', ') AS
regiao,
  GROUP_CONCAT(DISTINCT t2.name ORDER BY t2.name ASC SEPARATOR ', ') AS
uf_estado

FROM
  wp_posts p
  INNER JOIN wp_postmeta pm ON p.ID = pm.post_id
  LEFT JOIN wp_term_relationships tr1 ON p.ID = tr1.object_id
  LEFT JOIN wp_term_taxonomy tt1 ON tr1.term_taxonomy_id =
tt1.term_taxonomy_id
    AND tt1.taxonomy = 'tnc_tax_83776'
  LEFT JOIN wp_terms t1 ON tt1.term_id = t1.term_id
  LEFT JOIN wp_term_relationships tr2 ON p.ID = tr2.object_id
  LEFT JOIN wp_term_taxonomy tt2 ON tr2.term_taxonomy_id =
tt2.term_taxonomy_id
    AND tt2.taxonomy = 'tnc_tax_7447'
  LEFT JOIN wp_terms t2 ON tt2.term_id = t2.term_id

WHERE
  p.post_type = 'tnc_col_37719_item'
  AND p.post_status = 'publish'

GROUP BY
  p.ID, p.post_title, p.post_name

ORDER BY
  p.post_title ASC;
```

```
"""

df = extrair_dados_mysql(
   query=query_inrc,
   host="xxxxxxx",
   user="xxxxxxx",
   password="xxxxxxxx",
   database="xxxxxxxx"
)

#Salva
df.to_csv("util/base_inrc_projeto_identificacao.csv", index=False)


#-------------------------------------------------------------

#EXTRACAO TABELA tb_bem_contato
query_tb_bem_contato = "SELECT * FROM tb_bem_contato;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tb_bem_contato,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tb_bem_contato.csv", index=False)


#EXTRACAO TABELA tb_contato
query_tb_contato = "SELECT * FROM tb_contato;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tb_contato,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tb_contato.csv", index=False)

#EXTRACAO TABELA tb_contato
query_tb_tipo_logradouro = "SELECT * FROM tb_tipo_logradouro;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tb_tipo_logradouro,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
```

```
)
df.to_csv("util/tb_tipo_logradouro.csv", index=False)


#EXTRACAO TABELA tb_contato
query_tg_instituicao = "SELECT * FROM tg_instituicao.csv;"
# Executa extração
df = extrair_dados_postgresql(
   query=query_tg_instituicao,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/tg_instituicao.csv", index=False)


#---------------Tratamento----------------------
#------------Instituicoes executoras-------------
def tratar_dados_agentes_instituicao_executora() -> pd.DataFrame:

   df = pd.read_csv("util/base_inrc_projeto_identificacao.csv")

   #Garante que linhas onde 'post_title' está vazio ou nulo sejam removidas
   df = df[df['nome_projeto'].notnull() & (df['nome_projeto'].astype(str).str.strip() !=
"")]

   #Remove duplicatas com base em 'nome_projeto' + 'instituição_executora'
   df = df.drop_duplicates(subset=["nome_projeto", "nome_agente"])

   df['uf_estado'] = df['uf_estado'].astype(str)

   # Extrai a sigla (UF) para uma nova coluna
   df['uf'] = df['uf_estado'].str.extract(r'-\s*([A-Z]{2})$')

   # Remove o sufixo " - UF" da coluna uf_estado
   df['uf_estado'] = df['uf_estado'].str.replace(r'\s*-\s*[A-Z]{2}$', '', regex=True)

   # Extrai a sigla da UF da coluna 'uf_iso' (ex: "BR-AP" -> "AP")
   df['uf_iso'] = 'BR-' + df['uf']
   # Remove 'nan' caso tenha virado string durante o processo
   df['uf_estado'] = df['uf_estado'].replace('nan', '')
   df['regiao'] = df['regiao'].astype(str).str.replace(r'^Região\s+', '', regex=True)
   df['tipo_agente'] = 'Instituição Executora'

   df.rename(columns={'ID': 'id_agente'}, inplace=True)
   df['id_agente'] = 'INRC_' + df['id_agente'].astype(str).str.strip()
   df = df.drop(columns=['post_title'])
   df = df.drop(columns=['colecao_slug'])
   df = df.drop(columns=['item_slug'])
```

```
    #Remove as aspas simples: percorre todos os elementos do DataFrame, e se o
valor for uma string, ele substitui por espaco
    df = df.applymap(lambda x: x.replace("'", " ") if isinstance(x, str) else x)
    df.loc[:, 'ano_inicio'] = df['ano_inicio'].fillna('').astype(str).str.replace('.0', '',
regex=False)
    df.loc[:, 'ano_finalizacao'] =
df['ano_finalizacao'].fillna('').astype(str).str.replace('.0', '', regex=False)

    return df


#----------Instituicoes parceiras-----------------
def slugify(text):
    # Remove acentuação
    text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8')
    # Minúsculas
    text = text.lower()
    # Substitui caracteres não alfanuméricos por "_"
    text = re.sub(r'[^a-z0-9]+', '_', text)
    # Remove underscores duplicados ou nas bordas
    text = re.sub(r'_+', '_', text).strip('_')
    return text

def tratar_dados_agentes_instituicao_parceira() -> pd.DataFrame:

    df = pd.read_csv("../1-indicador-bens/bcr/util/dados_bcr_tratados.csv")

    # Agrupa pelo nome_agente
    grupos = []

    # Identifica colunas que serão agrupadas como multivalores
    colunas_multivaloradas = ["nome_bem", "co_iphan", "nome_municipio",
"uf_estado", "uf", "uf_iso", "regiao"]

    # Colunas fixas (que devem ter valor único por agente)
    colunas_fixas = ["bem_tipo_protecao", "bem_por_tipo", "tipo_agente"]

    # Remover linhas com nome_agente vazio
    df = df[df["nome_agente"].notna() & (df["nome_agente"].str.strip() != "")]

    # Agrupar por nome_agente
    agrupar = df.groupby("nome_agente")

    for nome_agente, grupo in agrupar:
        nova_linha = {"nome_agente": nome_agente}

        # Campos multivalorados: junta valores únicos com "; "
        for col in colunas_multivaloradas:
```

```
        valores_unicos = grupo[col].dropna().astype(str).unique()
        nova_linha[col] = '; '.join(sorted(valores_unicos))

    # Campos fixos: assume que todos têm o mesmo valor (valida)
    for col in colunas_fixas:
        valores_unicos = grupo[col].dropna().unique()
        if len(valores_unicos) == 1:
            nova_linha[col] = valores_unicos[0]
        else:
            nova_linha[col] = "CONFLITO"

    grupos.append(nova_linha)

# Cria novo DataFrame
df_agentes = pd.DataFrame(grupos)
df_tg = pd.read_csv("util/tg_instituicao.csv")

# Resultado final
resultado = []

    # Para cada agente, procurar instituição com ≥70% de similaridade
for _, row in df_agentes.iterrows():
    nome_agente = str(row['nome_agente']).strip()
    melhor_score = 0
    melhor_linha = None

    for _, linha_tg in df_tg.iterrows():
        nome_tg = str(linha_tg['no_instituicao']).strip()
        score = fuzz.ratio(nome_agente.lower(), nome_tg.lower())
        if score >= 70 and score > melhor_score:
            melhor_score = score
            melhor_linha = linha_tg

    nova_linha = row.copy()

    if melhor_linha is not None:
        for col in ["localizacao_especifica", "observacao", "geometria", "ativo"]:
            nova_linha[col] = melhor_linha.get(col, None)
    else:
        for col in ["localizacao_especifica", "observacao", "geometria", "ativo"]:
            nova_linha[col] = None

    resultado.append(nova_linha)

# DataFrame final
df = pd.DataFrame(resultado)

# Criar id_agente baseado no nome_agente
df['id_agente'] = 'BCR_' + df['nome_agente'].apply(slugify)
```

```
    #Função extrai longitude/latitude e também formatar como lista string
    def tratar_ponto(ponto):
        match = re.match(r'POINT\s*\(\s*([-\d\.]+)\s+([-\d\.]+)\s*\)', str(ponto))
        if match:
            lon = float(match.group(1))
            lat = float(match.group(2))
            return f"[{lon}, {lat}]", lon, lat
        return "", None, None

    # Aplica a função e cria as colunas
    # Aplica a função e cria as colunas
    df['ponto'], df['longitude'], df['latitude'] = zip(*df['geometria'].map(tratar_ponto))

    # Reorganiza as colunas para colocar 'id_agente' primeiro
    colunas = ['id_agente'] + [col for col in df.columns if col != 'id_agente']
    df = df[colunas]
    df = df.drop(columns=['geometria'])

    return df


#-----------------Agente Contato----------------
def tratar_dados_agentes_contato() -> pd.DataFrame:
    df = pd.read_csv("util/tb_contato.csv")
    df_bem = pd.read_csv("util/tb_bem_contato.csv")

    # Normaliza e remove linhas de teste
    if 'no_fisica_juridica' in df.columns:
        df['no_fisica_juridica'] = df['no_fisica_juridica'].astype(str).str.title()
        df = df[~df['no_fisica_juridica'].str.contains('teste', case=False, na=False)]

    df_filtrado = df[df["fisica_juridica"] == 1].copy()
    df_filtrado["tipo_fisica_juridica"] = "Jurídica"

    # Mapeia valores categóricos
    mapeamentos = {
        "tipo_setor": {1: "Público", 2: "Privado", 3: "Misto"},
        "tipo_setor_publico": {1: "Federal", 2: "Estadual", 3: "Distrital", 4: "Municipal"},
        "area_atuacao": {1: "Local", 2: "Regional", 3: "Nacional"}
    }

    for coluna, mapa in mapeamentos.items():
        if coluna in df_filtrado.columns:
            df_filtrado[coluna] = df_filtrado[coluna].map(mapa)

    # Substitui nulos
    for coluna in ["tipo_setor", "tipo_setor_publico", "area_atuacao"]:
        if coluna in df_filtrado.columns:
            df_filtrado[coluna] = df_filtrado[coluna].fillna("Não informado")
```

```
    # Remove colunas inúteis
    colunas_remover = [
        "nu_passaporte", "outros_nomes", "sexo", "ocupacao", "fisica_juridica",
        "tipo_contato", "cd_rg", "telefone_contato", "nu_cep_correspondencia",
        "id_munincipio_correspondencia", "id_tipo_logradouro_correspondencia",
        "de_endereco_correspondencia", "nu_endereco_correspondencia",
        "de_complemento_correspondencia", "no_bairro_correspondencia",
        "de_observavao_correspondencia"
    ]
    df_filtrado.drop(columns=[col for col in colunas_remover if col in
df_filtrado.columns], inplace=True)

    # Agrupa bens por contato
    df_bem_agrupado = (
        df_bem.groupby("id_contato")["id_bem"]
        .apply(lambda x: "|".join(str(int(b)) for b in x.dropna().unique()))
        .reset_index()
    )

    df_base = pd.merge(df_filtrado, df_bem_agrupado, on="id_contato", how="left")

    if 'cpfcnpj' in df_base.columns:
        df_base['cpfcnpj'] = df_base['cpfcnpj'].fillna('').astype(str).str.replace('.0', '',
regex=False)

    # UF e região
    df_mun = pd.read_csv("util/tg_municipio.csv")[["id_municipio", "nome", "id_uf"]]
    df_uf = pd.read_csv("util/tg_uf.csv")[["id_uf", "nome"]]
    df_uf.rename(columns={"nome": "uf_estado"}, inplace=True)

    nome_para_sigla = {
        "Acre": "AC", "Alagoas": "AL", "Amazonas": "AM", "Amapá": "AP", "Bahia": "BA",
        "Ceará": "CE", "Espírito Santo": "ES", "Goiás": "GO", "Maranhão": "MA",
        "Minas Gerais": "MG", "Mato Grosso do Sul": "MS", "Mato Grosso": "MT",
        "Pará": "PA", "Paraíba": "PB", "Paraná": "PR", "Pernambuco": "PE",
        "Piauí": "PI", "Rio de Janeiro": "RJ", "Rio Grande do Norte": "RN",
        "Rio Grande do Sul": "RS", "Rondônia": "RO", "Roraima": "RR",
        "Santa Catarina": "SC", "São Paulo": "SP", "Sergipe": "SE",
        "Tocantins": "TO", "Distrito Federal": "DF"
    }
    df_uf["uf"] = df_uf["uf_estado"].map(nome_para_sigla)

    df_mun.rename(columns={"nome": "nome_municipio"}, inplace=True)
    df_mun_uf = pd.merge(df_mun, df_uf, on="id_uf", how="left")
    df_merge = pd.merge(df_base, df_mun_uf, on="id_municipio", how="left")

    uf_para_regiao = {
        'AC': 'Norte', 'AP': 'Norte', 'AM': 'Norte', 'PA': 'Norte', 'RO': 'Norte', 'RR': 'Norte', 'TO':
'Norte',
        'AL': 'Nordeste', 'BA': 'Nordeste', 'CE': 'Nordeste', 'MA': 'Nordeste', 'PB': 'Nordeste',
```

```
    'PE': 'Nordeste', 'PI': 'Nordeste', 'RN': 'Nordeste', 'SE': 'Nordeste',
    'DF': 'Centro-Oeste', 'GO': 'Centro-Oeste', 'MT': 'Centro-Oeste', 'MS':
'Centro-Oeste',
    'ES': 'Sudeste', 'MG': 'Sudeste', 'RJ': 'Sudeste', 'SP': 'Sudeste',
    'PR': 'Sul', 'RS': 'Sul', 'SC': 'Sul'
  }

  df_merge["uf_iso"] = df_merge["uf"].map(lambda x: f"BR-{x}" if pd.notna(x) else
None)
  df_merge["regiao"] = df_merge["uf"].map(uf_para_regiao)
  df_merge["nome_municipio"] = df_merge["nome_municipio"].fillna("") + "-" +
df_merge["uf"].fillna("")
  df_merge["tipo_agente"] = "Contato Bem Material"

  colunas_originais = df_base.columns.tolist()
  colunas_adicionais = ["nome_municipio", "uf_estado", "uf", "uf_iso", "regiao",
"tipo_agente"]
  df = df_merge[colunas_originais + colunas_adicionais].copy()

  if 'area_atuacao' in df.columns:
    df.drop('area_atuacao', axis=1, inplace=True)

  # Telefones
  def formatar_telefone(numero):
    if pd.isna(numero):
      return None
    numero = re.sub(r'[^\d]', '', str(numero))
    if len(numero) == 10:
      return f"({numero[:2]}) {numero[2:6]}-{numero[6:]}"
    elif len(numero) == 11 and numero[2] == '9':
      return f"({numero[:2]}) {numero[2:7]}-{numero[7:]}"
    else:
      return numero

  for coluna in ['telefone_fixo', 'telefone_movel', 'telefone_alternativo']:
    if coluna in df.columns:
      df[coluna] = df[coluna].apply(formatar_telefone)

  # Tipo de logradouro
  if "id_tipo_logradouro" in df.columns:
    df_tipo = pd.read_csv("util/tb_tipo_logradouro.csv")
    idx = df.columns.get_loc("id_tipo_logradouro")
    df = df.merge(df_tipo, on="id_tipo_logradouro", how="left")
    df.insert(idx, "tipo_logradouro", df["ds_tipo_logradouro"])
    df.drop(columns=["id_tipo_logradouro", "ds_tipo_logradouro"], inplace=True)

  # Renomeia nome do agente
  df.rename(columns={"no_fisica_juridica": "nome_agente"}, inplace=True)

  # Nome, tipo e proteção dos bens
```

```
    df_bens = pd.read_csv("../1-indicador-bens/sicg/util/dados_sicg_tratados.csv",
dtype={"id_bem": str})
    df_bens['nome_bem'] = df_bens['nome_bem'].astype(str).str.replace('"',
'').str.strip()
    df_bens['bem_por_tipo'] = df_bens['bem_por_tipo'].astype(str).str.strip()
    df_bens['bem_tipo_protecao'] =
df_bens['bem_tipo_protecao'].astype(str).str.strip()

    mapa_nome_bem = df_bens.set_index('id_bem')['nome_bem'].to_dict()
    mapa_bem_por_tipo = df_bens.set_index('id_bem')['bem_por_tipo'].to_dict()
    mapa_bem_tipo_protecao =
df_bens.set_index('id_bem')['bem_tipo_protecao'].to_dict()

    def mapear_bem_por_campo(campo_ids, mapa):
        if pd.isna(campo_ids) or campo_ids == '':
            return ''
        ids = re.split(r'[|/]', str(campo_ids))
        valores = []
        for b in ids:
            b_clean = str(b).strip()
            if not b_clean:
                continue
            valor = mapa.get(b_clean)
            if valor and str(valor).lower() != 'nan':
                valores.append(valor)
        return '|'.join(valores).strip() if valores else ''


    df['nome_bem'] = df['id_bem'].apply(lambda x: mapear_bem_por_campo(x,
mapa_nome_bem))
    df['bem_por_tipo'] = df['id_bem'].apply(lambda x: mapear_bem_por_campo(x,
mapa_bem_por_tipo))
    df['bem_tipo_protecao'] = df['id_bem'].apply(lambda x:
mapear_bem_por_campo(x, mapa_bem_tipo_protecao))
    df['id_bem'] = df['id_bem'].str.replace('/', '|')

    # Final ajustes
    if 'cpfcnpj' in df.columns:
        df['cpfcnpj'] = df['cpfcnpj'].fillna('').astype(str).str.replace('.0', '', regex=False)
    if 'nu_cep' in df.columns:
        df['nu_cep'] = df['nu_cep'].fillna('').astype(str).str.replace('.0', '', regex=False)

    df = df.applymap(lambda x: str(x).replace("""", " ") if pd.notna(x) else x)


    #Cria a coluna id_agente no formato desejado
    df['id_agente'] = 'SICG_' + df['id_contato'].astype(str)

    #Opcional: reorganiza para colocar id_agente como primeira coluna
    colunas = ['id_agente'] + [col for col in df.columns if col != 'id_agente']
```

```
    df = df[colunas]

    return df


#-----------------Agente Contato----------------
def tratar_dados_agentes_instituicao_identificada() -> pd.DataFrame:

    df = pd.read_csv("util/tg_instituicao.csv")

    # Remove as colunas indesejadas, se existirem
    df = df.drop(columns=["rev", "revtype"], errors="ignore")

    df = df.rename(columns={"no_instituicao": "nome_agente"})

    # Lista de instituições a remover da coluna 'nome_agente'
    instituicoes_para_remover = [
        "OPAN (Operação Amazônia Nativa)",
        "Irmandade Carimbó de São Benedito",
        "IPAC - Instituto do Patrimônio Artístico Cultural da Bahia",
        "Irmandade do Carimbó de São Benedito",
        "Fundação Museu do Homem Americano – FUMDAHM",
        "Universidade Federal do Rio Grande do Sul",
        "Prefeitura de Ouro Preto (MG)",
        "OPAN Operação Amazônia Nativa",
        "Irmandade de São Benedito",
        "SUPERINTENDÊNCIA DO IPHAN NO ESTADO DO RIO GRANDE DO SUL",
        "Fundação Museu do Homem Americano"
    ]

    # Remove as linhas com essas instituições
    df = df[~df["nome_agente"].isin(instituicoes_para_remover)]


    #Função para normalizar: remove acentos e converte para minúsculas
    def normalizar(texto):
        if pd.isna(texto):
            return ""
        texto = str(texto).lower().strip()
        texto = unicodedata.normalize('NFKD', texto).encode('ASCII',
'ignore').decode('utf-8')
        return texto

    #Função para remover duplicatas baseadas em similaridade de nomes
    def remover_nomes_similares(df, coluna='nome_agente', limiar=90):
        nomes_mantidos = []
        indices_mantidos = []

        for idx, nome in df[coluna].dropna().items():
            nome_norm = normalizar(nome)
```

```
        duplicado = False

        for mantido in nomes_mantidos:
            score = fuzz.ratio(nome_norm, normalizar(mantido))
            if score >= limiar:
                duplicado = True
                break

        if not duplicado:
            nomes_mantidos.append(nome)
            indices_mantidos.append(idx)

    return df.loc[indices_mantidos].reset_index(drop=True)

df = remover_nomes_similares(df, coluna='nome_agente', limiar=90)



# Reorganiza as colunas para colocar os IDs no início (opcional)
df["id_agente"] = "INSTITUICAO_" + df["id_instituicao"].astype(str)

# Opcional: colocar id_agente como primeira coluna
colunas_ordenadas = ["id_agente"] + [col for col in df.columns if col != "id_agente"]
df = df[colunas_ordenadas]


#Extrair latitude e longitude da coluna geometria
#Substituir id_tipo_instituicao pelo seu significado (ds_tipo_instituicao)


# Função para extrair latitude e longitude da string 'geometria'
def extrair_coords(geometria):
    if pd.isnull(geometria):
        return pd.NA, pd.NA
    match = re.match(r'POINT\s*\((-?\d+\.\d+)\s+(-?\d+\.\d+)\)', geometria)
    if match:
        lon, lat = match.groups()
        return float(lat), float(lon)
    return pd.NA, pd.NA

# Aplica a função e cria as colunas 'latitude' e 'longitude'
df[['latitude', 'longitude']] = df['geometria'].apply(lambda x:
pd.Series(extrair_coords(x)))

# Dicionário de mapeamento de id_tipo_instituicao para descrição
mapa_tipos = {
    1: "Associação de detentores",
    2: "Instituição privada sem fins lucrativos (ONG, OSCIP, Cooperativas, etc)",
    3: "Instituição privada com fins lucrativos",
    4: "Fundação universitária",
```

```
    5: "Instituição religiosa",
    6: "Instituição pública federal",
    7: "Instituição pública estadual",
    8: "Instituição pública municipal",
    9: "Grupos de detentores",
    10: "Outra"
  }


  # Função para formatar geometria no formato [[lon, lat]]
  def formatar_para_superset(geometria):
    if pd.isnull(geometria):
      return pd.NA
    match = re.match(r'POINT\s*\((-?\d+\.\d+)\s+(-?\d+\.\d+)\)', geometria)
    if match:
      lon, lat = match.groups()
      return f"[{float(lon)}, {float(lat)}]"
    return pd.NA

  # Aplica o formato desejado para Superset
  df["geometria"] = df["geometria"].apply(formatar_para_superset)

  for col in ["ativo", "st_participe", "st_referencia"]:
    df[col] = df[col].astype(str).str.strip().str.lower().map({
      "true": "Sim",
      "false": "Não"
    }).fillna("Não")

  df = df.applymap(lambda x: str(x).replace("""", " ") if pd.notna(x) else x)
  # Cria nova coluna com a descrição do tipo de instituição


  df["id_tipo_instituicao"] = pd.to_numeric(df["id_tipo_instituicao"],
errors="coerce").astype("Int64")

  df["ds_tipo_instituicao"] = df["id_tipo_instituicao"].map(mapa_tipos)

  df.loc[:, 'id_instituicao'] = df['id_instituicao'].fillna('').astype(str).str.replace('.0', '',
regex=False)
  df = df.drop(columns=["id_tipo_instituicao"])

  df["tipo_agente"] = "Instituição Identificada"


  # Remove duplicados com base em 'id_agente', mantendo apenas o primeiro
  df = df.drop_duplicates(subset=["id_agente"], keep='first')

  return df

#----------Carrega os dados no armazém de dados----------
```

```
def carregando_dados_armazem(csv_path, id_colecao, id_metadado_chave,
nome_coluna_chave, coluna_para_id_metadado):
    start_time = time.time()
    df_final = pd.read_csv(csv_path)

    conn = psycopg2.connect(
        host="xxxxxx",
        dbname="xxxxxxx",
        user="xxxxxx",
        password="xxxxxx"
    )
    cursor = conn.cursor()
    conn.autocommit = False

    # Carrega itens existentes pela chave
    cursor.execute("""
        SELECT i.id, v.value
        FROM obs_valor_metadado v
        JOIN obs_itens i ON v.id_valor = i.id
        WHERE v.id_metadado = %s AND i.id_colecao = %s
    """, (id_metadado_chave, id_colecao))
    chaves_existentes = {valor.strip(): item_id for item_id, valor in cursor.fetchall()}

    cursor.execute("SELECT id_valor, id_metadado FROM obs_valor_metadado")
    valores_existentes = {(id_valor, id_metadado) for id_valor, id_metadado in
cursor.fetchall()}

    itens_atualizados = set()
    insert_valores = []
    update_valores = []

    for i, row in df_final.iterrows():
        chave_valor = str(row.get(nome_coluna_chave)).strip()
        if not chave_valor or chave_valor.lower() == "nan":
            continue

        if chave_valor in chaves_existentes:
            id_item = chaves_existentes[chave_valor]
            item_ja_existente = True
        else:
            post_title = row.get("nome_projeto") or row.get("nome_agente") or "Item sem
título"
            cursor.execute("""
                INSERT INTO obs_itens (id_colecao, post_title)
                VALUES (%s, %s)
                RETURNING id
            """, (id_colecao, post_title))
            id_item = cursor.fetchone()[0]
            chaves_existentes[chave_valor] = id_item
            item_ja_existente = False
```

```
            insert_valores.append((id_item, id_metadado_chave, chave_valor))

        for col, id_metadado in coluna_para_id_metadado.items():
            if col == nome_coluna_chave:
                continue

            valor = row.get(col)
            if pd.isna(valor) or str(valor).strip().lower() == "nan" or str(valor).strip() == "":
                continue

            valor_str = str(valor).replace("'", " ").strip()
            chave = (id_item, id_metadado)

            if chave in valores_existentes:
                cursor.execute("""
                    SELECT value FROM obs_valor_metadado
                    WHERE id_valor = %s AND id_metadado = %s
                """, (id_item, id_metadado))
                valor_atual = cursor.fetchone()

                if valor_atual and valor_atual[0] != valor_str:
                    update_valores.append((valor_str, id_item, id_metadado))
                    if item_ja_existente:
                        itens_atualizados.add(id_item)
            else:
                insert_valores.append((id_item, id_metadado, valor_str))
                valores_existentes.add(chave)

        if i % 1000 == 0 and i > 0:
            if insert_valores:
                cursor.executemany("""
                    INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
                    VALUES (%s, %s, %s)
                """, insert_valores)
                insert_valores.clear()

            if update_valores:
                cursor.executemany("""
                    UPDATE obs_valor_metadado
                    SET value = %s
                    WHERE id_valor = %s AND id_metadado = %s
                """, update_valores)
                update_valores.clear()

            conn.commit()
            print(f"Processadas {i} linhas...")

    if insert_valores:
        cursor.executemany("""
```

```
        INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
        VALUES (%s, %s, %s)
    """, insert_valores)

  if update_valores:
    cursor.executemany("""
      UPDATE obs_valor_metadado
      SET value = %s
      WHERE id_valor = %s AND id_metadado = %s
    """, update_valores)

  conn.commit()
  cursor.close()
  conn.close()

  print("Processamento concluído.")
  print(f"Total de itens atualizados: {len(itens_atualizados)}")
  print(f"Tempo total de execução: {round(time.time() - start_time, 2)} segundos")



if __name__ == "__main__":
  # Etapa 1: Extração (realiza a extracao no inicio)

  # Etapa 2: Tratamento
  #Agentes contato - SICG
  df_tratado_contato = tratar_dados_agentes_contato()
  df_tratado_contato.to_csv('util/dados_agentes_tratados_contato.csv', index=False,
na_rep='')
  #Agentes instituicao executora - INRC
  df_tratado_executora = tratar_dados_agentes_instituicao_executora()
  df_tratado_executora.to_csv('util/dados_agentes_tratados_inrc.csv', index=False)
  #Agentes instituicao parceira - BCR
  df_tratado_parceira = tratar_dados_agentes_instituicao_parceira()
  df_tratado_parceira.to_csv('util/dados_agentes_tratados_bcr.csv', index=False)
  #Agentes instituicao identificada - SICG
  df_tratado_identificada = tratar_dados_agentes_instituicao_identificada()
  df_tratado_identificada.to_csv('util/dados_agentes_tratados_sicg_identificada.csv',
index=False)

  # Etapa 3: Carga
  # Instituicao executora - INRC
  coluna_para_id_executora = {
    "id_agente": 181,
    "url": 182,
    "nome_projeto": 183,
    "nome_agente": 184,
    "ano_inicio": 185,
    "ano_finalizacao": 186,
```

```
    "valor_orcamentario": 187,
    "regiao": 188,
    "uf_estado": 189,
    "uf": 190,
    "uf_iso": 191,
    "tipo_agente": 192
}

carregando_dados_armazem(
    csv_path="util/dados_agentes_tratados_inrc.csv",
    id_colecao=16,
    id_metadado_chave=181,
    nome_coluna_chave="id_agente",
    coluna_para_id_metadado=coluna_para_id_executora
)

# Instituicao parceira - BCR
coluna_para_id_parceira = {
    "id_agente": 193,
    "nome_agente": 194,
    "nome_bem": 195,
    "co_iphan": 196,
    "nome_municipio": 197,
    "uf_estado": 198,
    "uf": 199,
    "uf_iso": 200,
    "regiao": 201,
    "bem_tipo_protecao": 202,
    "bem_por_tipo": 203,
    "tipo_agente": 204,
    "localizacao_especifica": 205,
    "observacao": 206,
    "ativo": 207,
    "ponto": 208,
    "longitude": 209,
    "latitude": 210
}

carregando_dados_armazem(
    csv_path="util/dados_agentes_tratados_inrc.csv",
    id_colecao=17,
    id_metadado_chave=193,
    nome_coluna_chave="id_agente",
    coluna_para_id_metadado=coluna_para_id_parceira
)

# Contato bem - SICG
coluna_para_id_contato = {
    "id_agente": 211,
    "id_contato": 212,
```

```
    "id_pais": 213,
    "id_municipio": 214,
    "cpfcnpj": 215,
    "outra_identificacao": 216,
    "nome_agente": 217,
    "ds_localidade": 218,
    "nu_cep": 219,
    "no_logradouro": 220,
    "nu_logradouro": 221,
    "ds_complemento": 222,
    "no_bairro": 223,
    "tipo_logradouro": 224,
    "local_especifico": 225,
    "ds_email": 226,
    "sg_empresa": 227,
    "no_contato": 228,
    "tipo_setor": 229,
    "tipo_setor_publico": 230,
    "tipo_setor_privado": 231,
    "telefone_fixo": 232,
    "telefone_movel": 233,
    "telefone_alternativo": 234,
    "tipo_fisica_juridica": 235,
    "id_bem": 236,
    "nome_municipio": 237,
    "uf_estado": 238,
    "uf": 239,
    "uf_iso": 240,
    "regiao": 241,
    "tipo_agente": 242,
    "nome_bem": 243,
    "bem_por_tipo": 244,
    "bem_tipo_protecao": 245
}

carregando_dados_armazem(
    csv_path="util/dados_agentes_tratados_contato.csv",
    id_colecao=18,
    id_metadado_chave=211,
    nome_coluna_chave="id_agente",
    coluna_para_id_metadado=coluna_para_id_contato
)

# Instituicao identificada - SICG
coluna_para_id_identificada = {
    "id_agente": 246,
    "id_instituicao": 247,
    "st_participe": 248,
    "st_referencia": 249,
    "nome_agente": 250,
```

```
    "localizacao_especifica": 251,
    "observacao": 252,
    "geometria": 253,
    "ativo": 254,
    "latitude": 255,
    "longitude": 256,
    "ds_tipo_instituicao": 257,
    "tipo_agente": 258
  }

  carregando_dados_armazem(
    csv_path="util/dados_agentes_tratados_sicg_identificada.csv",
    id_colecao=19,
    id_metadado_chave=246,
    nome_coluna_chave="id_agente",
    coluna_para_id_metadado=coluna_para_id_identificada
  )
```

Fonte: elaborado pelos autores (2025).

e) View Implementada para o Indicador de Agentes

A seguir, apresenta-se o código SQL usado para criação da view vw_indicador_agentes, implementada no armazém de dados.

Quadro 10 - View vw_indicador_agentes

```
CREATE OR REPLACE VIEW obs_view_colecao_16 AS
SELECT
  i.id AS id_item,
  MAX(CASE WHEN m.nome = 'id_agente' THEN v.value END) AS id_agente,
  MAX(CASE WHEN m.nome = 'url' THEN v.value END) AS url,
  MAX(CASE WHEN m.nome = 'nome_projeto' THEN v.value END) AS nome_projeto,
  MAX(CASE WHEN m.nome = 'nome_agente' THEN v.value END) AS nome_agente,
  MAX(CASE WHEN m.nome = 'ano_inicio' THEN v.value END) AS ano_inicio,
  MAX(CASE WHEN m.nome = 'ano_finalizacao' THEN v.value END) AS
ano_finalizacao,
  MAX(CASE WHEN m.nome = 'regiao' THEN v.value END) AS regiao,
  MAX(CASE WHEN m.nome = 'uf_estado' THEN v.value END) AS uf_estado,
  MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
  MAX(CASE WHEN m.nome = 'uf_iso' THEN v.value END) AS uf_iso,
  MAX(CASE WHEN m.nome = 'tipo_agente' THEN v.value END) AS tipo_agente
FROM obs_itens i
JOIN obs_valor_metadado v ON v.id_valor = i.id
JOIN obs_metadados m ON m.id = v.id_metadado
WHERE i.id_colecao = 16
GROUP BY i.id;

CREATE OR REPLACE VIEW obs_view_colecao_17 AS
```

```
SELECT
  i.id AS id_item,
  MAX(CASE WHEN m.nome = 'id_agente' THEN v.value END) AS id_agente,
  MAX(CASE WHEN m.nome = 'nome_agente' THEN v.value END) AS nome_agente,
  MAX(CASE WHEN m.nome = 'nome_bem' THEN v.value END) AS nome_bem,
  MAX(CASE WHEN m.nome = 'co_iphan' THEN v.value END) AS co_iphan,
  MAX(CASE WHEN m.nome = 'nome_municipio' THEN v.value END) AS
nome_municipio,
  MAX(CASE WHEN m.nome = 'uf_estado' THEN v.value END) AS uf_estado,
  MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
  MAX(CASE WHEN m.nome = 'uf_iso' THEN v.value END) AS uf_iso,
  MAX(CASE WHEN m.nome = 'regiao' THEN v.value END) AS regiao,
  MAX(CASE WHEN m.nome = 'bem_tipo_protecao' THEN v.value END) AS
bem_tipo_protecao,
  MAX(CASE WHEN m.nome = 'bem_por_tipo' THEN v.value END) AS bem_por_tipo,
  MAX(CASE WHEN m.nome = 'tipo_agente' THEN v.value END) AS tipo_agente,
  MAX(CASE WHEN m.nome = 'localizacao_especifica' THEN v.value END) AS
localizacao_especifica,
  MAX(CASE WHEN m.nome = 'observacao' THEN v.value END) AS observacao,
  MAX(CASE WHEN m.nome = 'ativo' THEN v.value END) AS ativo,
  MAX(CASE WHEN m.nome = 'ponto' THEN v.value END) AS ponto,
  MAX(CASE WHEN m.nome = 'longitude' THEN v.value END) AS longitude,
  MAX(CASE WHEN m.nome = 'latitude' THEN v.value END) AS latitude
FROM obs_itens i
JOIN obs_valor_metadado v ON v.id_valor = i.id
JOIN obs_metadados m ON m.id = v.id_metadado
WHERE i.id_colecao = 17
GROUP BY i.id;



CREATE OR REPLACE VIEW obs_view_colecao_18 AS
SELECT
  i.id AS id_item,
  MAX(CASE WHEN m.nome = 'id_agente' THEN v.value END) AS id_agente,
  MAX(CASE WHEN m.nome = 'id_bem' THEN v.value END) AS id_bem,
  MAX(CASE WHEN m.nome = 'cpfcnpj' THEN v.value END) AS cpfcnpj,
  MAX(CASE WHEN m.nome = 'nome_agente' THEN v.value END) AS nome_agente,
  MAX(CASE WHEN m.nome = 'ds_email' THEN v.value END) AS ds_email,
  MAX(CASE WHEN m.nome = 'regiao' THEN v.value END) AS regiao,
  MAX(CASE WHEN m.nome = 'uf_estado' THEN v.value END) AS uf_estado,
  MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
  MAX(CASE WHEN m.nome = 'uf_iso' THEN v.value END) AS uf_iso,
  MAX(CASE WHEN m.nome = 'nome_municipio' THEN v.value END) AS
nome_municipio,
  MAX(CASE WHEN m.nome = 'tipo_agente' THEN v.value END) AS tipo_agente,
  MAX(CASE WHEN m.nome = 'tipo_setor' THEN v.value END) AS tipo_setor,
  MAX(CASE WHEN m.nome = 'tipo_setor_publico' THEN v.value END) AS
tipo_setor_publico,
  MAX(CASE WHEN m.nome = 'tipo_setor_privado' THEN v.value END) AS
```

```
tipo_setor_privado,
   MAX(CASE WHEN m.nome = 'nome_bem' THEN v.value END) AS nome_bem,
   MAX(CASE WHEN m.nome = 'bem_por_tipo' THEN v.value END) AS bem_por_tipo,
   MAX(CASE WHEN m.nome = 'bem_tipo_protecao' THEN v.value END) AS
bem_tipo_protecao
FROM obs_itens i
JOIN obs_valor_metadado v ON v.id_valor = i.id
JOIN obs_metadados m ON m.id = v.id_metadado
WHERE i.id_colecao = 18
GROUP BY i.id;


CREATE OR REPLACE VIEW obs_view_colecao_19 AS
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'id_agente' THEN v.value END) AS id_agente,
   MAX(CASE WHEN m.nome = 'id_instituicao' THEN v.value END) AS id_instituicao,
   MAX(CASE WHEN m.nome = 'nome_agente' THEN v.value END) AS nome_agente,
   MAX(CASE WHEN m.nome = 'localizacao_especifica' THEN v.value END) AS
localizacao_especifica,
   MAX(CASE WHEN m.nome = 'geometria' THEN v.value END) AS geometria,
   MAX(CASE WHEN m.nome = 'ativo' THEN v.value END) AS ativo,
   MAX(CASE WHEN m.nome = 'latitude' THEN v.value END) AS latitude,
   MAX(CASE WHEN m.nome = 'longitude' THEN v.value END) AS longitude,
   MAX(CASE WHEN m.nome = 'ds_tipo_instituicao' THEN v.value END) AS
ds_tipo_instituicao,
   MAX(CASE WHEN m.nome = 'tipo_agente' THEN v.value END) AS tipo_agente
FROM obs_itens i
JOIN obs_valor_metadado v ON v.id_valor = i.id
JOIN obs_metadados m ON m.id = v.id_metadado
WHERE i.id_colecao = 19
GROUP BY i.id;
```

```
CREATE MATERIALIZED VIEW vw_indicador_agentes AS

-- COLEÇÃO 16
SELECT
   id_agente,
   nome_agente,
   tipo_agente,
   NULL::TEXT AS nome_bem,
   NULL::TEXT AS co_iphan,
   NULL::TEXT AS nome_municipio,
   uf_estado,
   uf,
   uf_iso,
   regiao,
```

```
    NULL::TEXT AS bem_tipo_protecao,
    NULL::TEXT AS bem_por_tipo,
    NULL::TEXT AS localizacao_especifica,
    NULL::TEXT AS observacao,
    NULL::TEXT AS ativo,
    NULL::TEXT AS ponto,
    NULL::TEXT AS longitude,
    NULL::TEXT AS latitude,
    NULL::TEXT AS id_bem,
    NULL::TEXT AS id_instituicao,
    nome_projeto,
    ano_inicio,
    ano_finalizacao,
    url,
    NULL::TEXT AS ds_email,
    NULL::TEXT AS tipo_setor,
    NULL::TEXT AS tipo_setor_publico,
    NULL::TEXT AS tipo_setor_privado,
    NULL::TEXT AS telefone_fixo,
    NULL::TEXT AS telefone_movel,
    NULL::TEXT AS geometria,
    NULL::TEXT AS ds_tipo_instituicao
FROM obs_view_colecao_16

UNION ALL

-- COLEÇÃO 17
SELECT
    id_agente,
    nome_agente,
    tipo_agente,
    nome_bem,
    co_iphan,
    nome_municipio,
    uf_estado,
    uf,
    uf_iso,
    regiao,
    bem_tipo_protecao,
    bem_por_tipo,
    localizacao_especifica,
    observacao,
    ativo,
    ponto,
    longitude,
    latitude,
    NULL::TEXT AS id_bem,
    NULL::TEXT AS id_instituicao,
    NULL::TEXT AS nome_projeto,
    NULL::TEXT AS ano_inicio,
```

```
    NULL::TEXT AS ano_finalizacao,
    NULL::TEXT AS url,
    NULL::TEXT AS ds_email,
    NULL::TEXT AS tipo_setor,
    NULL::TEXT AS tipo_setor_publico,
    NULL::TEXT AS tipo_setor_privado,
    NULL::TEXT AS telefone_fixo,
    NULL::TEXT AS telefone_movel,
    NULL::TEXT AS geometria,
    NULL::TEXT AS ds_tipo_instituicao
FROM obs_view_colecao_17

UNION ALL

-- COLEÇÃO 18
SELECT
    id_agente,
    nome_agente,
    tipo_agente,
    nome_bem,
    NULL::TEXT AS co_iphan,
    nome_municipio,
    uf_estado,
    uf,
    uf_iso,
    regiao,
    bem_tipo_protecao,
    bem_por_tipo,
    NULL::TEXT AS localizacao_especifica,
    NULL::TEXT AS observacao,
    NULL::TEXT AS ativo,
    NULL::TEXT AS ponto,
    NULL::TEXT AS longitude,
    NULL::TEXT AS latitude,
    id_bem,
    NULL::TEXT AS id_instituicao,
    NULL::TEXT AS nome_projeto,
    NULL::TEXT AS ano_inicio,
    NULL::TEXT AS ano_finalizacao,
    NULL::TEXT AS url,
    ds_email,
    NULL::TEXT AS tipo_setor,
    NULL::TEXT AS tipo_setor_publico,
    NULL::TEXT AS tipo_setor_privado,
    NULL::TEXT AS telefone_fixo,
    NULL::TEXT AS telefone_movel,
    NULL::TEXT AS geometria,
    NULL::TEXT AS ds_tipo_instituicao
FROM obs_view_colecao_18
```

```
UNION ALL

-- COLEÇÃO 19
SELECT
   id_agente,
   nome_agente,
   tipo_agente,
   NULL::TEXT AS nome_bem,
   NULL::TEXT AS co_iphan,
   NULL::TEXT AS nome_municipio,
   NULL::TEXT AS uf_estado,
   NULL::TEXT AS uf,
   NULL::TEXT AS uf_iso,
   NULL::TEXT AS regiao,
   NULL::TEXT AS bem_tipo_protecao,
   NULL::TEXT AS bem_por_tipo,
   localizacao_especifica,
   NULL::TEXT AS observacao,
   ativo,
   NULL::TEXT AS ponto,
   longitude,
   latitude,
   NULL::TEXT AS id_bem,
   id_instituicao,
   NULL::TEXT AS nome_projeto,
   NULL::TEXT AS ano_inicio,
   NULL::TEXT AS ano_finalizacao,
   NULL::TEXT AS url,
   NULL::TEXT AS ds_email,
   NULL::TEXT AS tipo_setor,
   NULL::TEXT AS tipo_setor_publico,
   NULL::TEXT AS tipo_setor_privado,
   NULL::TEXT AS telefone_fixo,
   NULL::TEXT AS telefone_movel,
   geometria,
   ds_tipo_instituicao
FROM obs_view_colecao_19;
```

Fonte: elaborado pelos autores (2025).

f) Consulta SQL implementada no Apache Superset para gerar o DATASET do Indicador de Agentes

O select abaixo é responsável por recuperar os registros da view vw_indicador_agentes, utilizada como base para visualizações no Superset. Além dos metadados consolidados, o select também trata:

- Geração do link clicáveis para o campo url convertido em um link HTML único.

Esse select é usado diretamente na criação do dataset no Superset, para o dashboard de agentes. Para garantir uma contagem precisa das informações, utilizamos o metadado id_agente, presente em ambas as ações, como controle em operações de COUNT(DISTINCT) quando necessário. A seguir, apresenta-se a consulta SQL da view vw_indicador_agentes no Superset.

Quadro 11 - Consulta Sql Da View Vw_indicador_agentes

```
SELECT
  *,
  REGEXP_REPLACE(ano_finalizacao::TEXT, '\.0$', '') AS
ano_finalizacao_normalizado,
  REGEXP_REPLACE(ano_inicio::TEXT, '\.0$', '') AS ano_inicio_normalizado,

  CASE
    WHEN url IS NOT NULL AND url != ''
    THEN CONCAT('<a href="', url, '" target="_blank">Acessar</a>')
    ELSE NULL
  END AS link_url
FROM vw_indicador_agentes;
```

Fonte: elaborado pelos autores (2025).

3.1.3.1.3 Indicador - Ações

O script de automação para o indicador Ações, inclui as ações do processo institucional preservação (fiscalização), fonte: FISCALIS e as ações processo institucional de identificação, fonte: INRC.

a) Script de Automação do Indicador de Ações

Os processos descritos a seguir foram implementados como parte do fluxo de integração dos dados do indicador de ações no armazém de dados.

b) Extração de dados do indicador Ações

Foram desenvolvidas funções para realizar a extração de dados a partir das bases de dados Mysql e SQLserver.

- MySQL – Base INRC

A função extrair_dados_mysql() realiza a extração das ações do INRC.

- SQL Server – Base FISCALIS

A função extrair_dados_sqlserver(), extrai dados dos FISCALIS, tabelas: documentos_sei e fiscalizacao_bem.

c) Implementação do Tratamento

Após a extração, o script realiza o tratamento dos dados das ações do processo institucional preservação (fiscalização), tabela do banco de dados fiscalizacao_bem do FISCALIS e das ações processo institucional de identificação: INRC, coleção 2.1 - Projeto de Identificação do INRC.

- **Tratamento das ações do processo institucional preservação (fiscalização)**
  - Mapeamento de colunas binárias: converte valores 1/0 das colunas tem_intervencao, irregularidade_encontrada, existe_dano, obra_autorizada, obra_em_andamento, obra_em_conformidade, responsavel_identificado, perda_autenticidade e perda_integridade para 'Sim' e 'Não'.
  - Mapeamento de classificações:
    - A coluna id_tipo_estado_preservacao é mapeada para os valores legíveis: Íntegro, Pouco Alterado, Muito Alterado, Descaracterizado.
    - A coluna id_tipo_estado_conservacao é mapeada para: Bom, Regular, Ruim, Péssimo.

      Ambas são renomeadas respectivamente para tipo_estado_preservacao e tipo_estado_conservacao.
  - Remoção de colunas irrelevantes ou nulas: são excluídas as colunas indicador_sem_imovel, bem_desaparecido, id_justificativa_desaparecimento e ordem.
  - Filtragem de registros inválidos:
    - Linhas com codigo_iphan vazio ou nulo são removidas.
    - Linhas com id_fiscalizacao vazio, nulo ou com valor 'nan' (como string) também são excluídas.

- Integração com documentos SEI: Os documentos vinculados às fiscalizações são agregados a partir da tabela documentos_sei.csv, usando a coluna id_fiscalizacao. São incluídas as colunas numero_documento e data_criacao.
- Integração com dados do SICG: São incorporadas informações complementares sobre os bens a partir dos dados tratados do SICG, por meio do codigo_iphan. As colunas incluídas estão listadas em colunas_desejadas, como: nome_bem, regiao, uf, nome_agente, entre outras.
- Adição da coluna status_bem_sicg:
  Informa se o bem foi encontrado no SICG. Valores possíveis: 'Bem encontrado no SICG' ou 'Bem não encontrado ou inativo'.
- Padronização de IDs e limpeza de formatos: Alguns identificadores numéricos são convertidos para texto e padronizados para evitar que sufixos como .0 sejam mantidos. Também é feita a substituição de aspas simples (') por espaços, para evitar erros em inserções SQL.
- Coluna adicional: A coluna tipo_acao é adicionada com valor fixo: "Ação do Processo Institucional Preservação: Fiscalização".

- **Tratamento das ações do processo institucional de identificação: INRC**
  - Remoção de linhas com nome_projeto vazio ou nulo: Garante que apenas registros válidos de projetos sejam mantidos.
  - Eliminação de duplicatas com base nas colunas nome_projeto e instituicao_executora: Evita que projetos repetidos com a mesma instituição.
  - Padronização de campos numéricos: As colunas ano_inicio e ano_finalizacao são convertidas para texto, e eventuais sufixos .0 (de números float) são removidos, garantindo formato limpo de ano (ex: "2015.0" para "2015").
  - Tratamento da coluna uf_estado e extração da sigla da UF: A coluna uf_estado, originalmente no formato "Estado - UF" (ex: "Amapá - AP"), é separada:
    - A parte do nome do estado é mantida em uf_estado (ex: "Amapá").
    - A sigla da UF é extraída para uma nova coluna uf (ex: "AP").
    - A sigla também é usada para gerar a coluna uf_iso, no formato "BR-UF" (ex: "BR-AP").

- Padronização da coluna regiao: A palavra "Região" é removida do início da string, mantendo apenas o nome da região (ex: "Região Nordeste" para "Nordeste").
- Inclusão de metadado tipo_acao: A coluna tipo_acao é adicionada com valor fixo: "Ação do processo institucional de identificação: INRC".
- Identificador único: A coluna ID é renomeada para id_controle, os valores dessa coluna recebem o prefixo "INRC_", garantindo unicidade (ex: 123 para INRC_123).
- Tratamento de caracteres: Aspas simples (') são substituídas por espaço para evitar erros de inserção em SQL.

d) Inserção/Atualização no Armazém de Dados

Após o tratamento dos dados, é realizada a etapa de inserção ou atualização dos valores dos metadados no armazém de dados, utilizando conexão com banco do armazém PostgreSQL. Este processo garante que os dados tratados sejam devidamente sincronizados com o armazém de dados, mantendo atualizações incrementais, controle de duplicidade via id_controle e preservando os metadados associados a cada item da coleção.

Abaixo, apresenta-se o script completo desenvolvido para o indicador de ações.

Quadro 12 - Script Completo - Ações

```
#!/usr/bin/env python
# coding: utf-8

#Indicador Ações - FISCALIS E INRC

#Bibliotecas
import numpy as np
import pandas as pd
import pymysql
import re
from unidecode import unidecode
import psycopg2
from fuzzywuzzy import process, fuzz
import pyodbc
import time
#--------------Extracao nas bases de dados-----------

Executa uma query no MySQL e retorna um DataFrame
def extrair_dados_mysql(
   query: str,
```

```
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 3306
) -> pd.DataFrame:

    conn = None
    try:
        conn = pymysql.connect(
            host=host,
            user=user,
            password=password,
            database=database,
            port=port
        )
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()


#Executa uma query no SQLserver e retorna um DataFrame
def extrair_dados_sqlserver(
    query: str,
    host: str,
    user: str,
    password: str,
    database: str,
    port: int = 1433
) -> pd.DataFrame:
    """Executa uma query no SQL Server e retorna um DataFrame."""
    conn = None
    try:
        conn_str = (
            f"DRIVER={{ODBC Driver 17 for SQL Server}};"
            f"SERVER={host},{port};"
            f"DATABASE={database};"
            f"UID={user};"
            f"PWD={password}"
        )
        conn = pyodbc.connect(conn_str)
        df = pd.read_sql(query, conn)
        return df
    finally:
        if conn:
            conn.close()
```

```
#EXTRACAO TABELA fiscalizacao_bem Banco de dados SQLserver FISCALIS
query_fiscalizacao_bem = "SELECT * FROM fiscalizacao_bem;"
# Executa extração
df = extrair_dados_sqlserver(
   query=query_fiscalizacao_bem,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/fiscalizacao_bem.csv", index=False)

#EXTRACAO TABELA documentos_sei Banco de dados SQLserver FISCALIS
query_documentos_sei = "SELECT * FROM documentos_sei;"
# Executa extração
df = extrair_dados_sqlserver(
   query=query_documentos_sei,
   host="xxxxxx",
   user="xxxxxx",
   password="xxxxxx",
   database="xxxxxx"
)
df.to_csv("util/documentos_sei.csv", index=False)

#EXTRACAO da Coleção INRC: 2.1 - Projeto de Identificação
query_inrc = """
SELECT
  p.ID,
  p.post_title,
  p.post_name AS item_slug,
  '2-1-projetos' AS colecao_slug,
  CONCAT('https://inrc.iphan.gov.br/2-1-projetos/', p.post_name, '/') AS url,

  MAX(CASE WHEN pm.meta_key = '37722' THEN pm.meta_value END) AS
nome_projeto,
  MAX(CASE WHEN pm.meta_key = '83787' THEN pm.meta_value END) AS
instituicao_executora,
  MAX(CASE WHEN pm.meta_key = '83790' THEN pm.meta_value END) AS
ano_inicio,
  MAX(CASE WHEN pm.meta_key = '83793' THEN pm.meta_value END) AS
ano_finalizacao,
  MAX(CASE WHEN pm.meta_key = '83803' THEN pm.meta_value END) AS
valor_orcamentario,

  GROUP_CONCAT(DISTINCT t1.name ORDER BY t1.name ASC SEPARATOR ', ') AS
regiao,
  GROUP_CONCAT(DISTINCT t2.name ORDER BY t2.name ASC SEPARATOR ', ') AS
uf_estado

FROM
```

```
  wp_posts p
  INNER JOIN wp_postmeta pm ON p.ID = pm.post_id
  LEFT JOIN wp_term_relationships tr1 ON p.ID = tr1.object_id
  LEFT JOIN wp_term_taxonomy tt1 ON tr1.term_taxonomy_id =
tt1.term_taxonomy_id
    AND tt1.taxonomy = 'tnc_tax_83776'
  LEFT JOIN wp_terms t1 ON tt1.term_id = t1.term_id
  LEFT JOIN wp_term_relationships tr2 ON p.ID = tr2.object_id
  LEFT JOIN wp_term_taxonomy tt2 ON tr2.term_taxonomy_id =
tt2.term_taxonomy_id
    AND tt2.taxonomy = 'tnc_tax_7447'
  LEFT JOIN wp_terms t2 ON tt2.term_id = t2.term_id

WHERE
  p.post_type = 'tnc_col_37719_item'
  AND p.post_status = 'publish'

GROUP BY
  p.ID, p.post_title, p.post_name

ORDER BY
  p.post_title ASC;

"""

df = extrair_dados_mysql(
    query=query_inrc,
    host="xxxxxx",
    user="xxxxxxx",
    password="xxxxxxx",
    database="xxxxxxx"
)

#Salva
df.to_csv("util/base_inrc_projeto_identificacao.csv", index=False)


#---------------Tratamento----------------------
def tratar_dados_acoes_fiscalis() -> pd.DataFrame:
    #Carrega os dados
    df = pd.read_csv('util/fiscalizacao_bem.csv')
    documentos_df = pd.read_csv('util/documentos_sei.csv')
    sicg_df = pd.read_csv('../1-indicador-bens/sicg/util/dados_sicg_tratados.csv')

    #Lista de colunas com 1/0 para mapear para 'Sim'/'Não'
    colunas_binarias = [
        'tem_intervencao', 'irregularidade_encontrada', 'existe_dano',
        'obra_autorizada', 'obra_em_andamento', 'obra_em_conformidade',
        'responsavel_identificado', 'perda_autenticidade', 'perda_integridade'
    ]
```

```
    mapeamento_binario = {1: 'Sim', 0: 'Não'}
    for col in colunas_binarias:
        if col in df.columns:
            df[col] = df[col].map(mapeamento_binario)

    #Mapear e renomear estado de preservação
    mapa_preservacao = {
        0: 'Íntegro', 1: 'Pouco Alterado', 2: 'Muito Alterado', 3: 'Descaracterizado'
    }
    if 'id_tipo_estado_preservacao' in df.columns:
        df['id_tipo_estado_preservacao'] =
df['id_tipo_estado_preservacao'].map(mapa_preservacao)
        df.rename(columns={'id_tipo_estado_preservacao': 'tipo_estado_preservacao'},
inplace=True)

    #Mapear e renomear estado de conservação
    mapa_conservacao = {
        0: 'Bom', 1: 'Regular', 2: 'Ruim', 3: 'Péssimo'
    }
    if 'id_tipo_estado_conservacao' in df.columns:
        df['id_tipo_estado_conservacao'] =
df['id_tipo_estado_conservacao'].map(mapa_conservacao)
        df.rename(columns={'id_tipo_estado_conservacao': 'tipo_estado_conservacao'},
inplace=True)

    #Corrigir: estava faltando vírgula entre colunas!
    colunas_para_remover = [
        'indicador_sem_imovel', 'bem_desaparecido', 'id_justificativa_desaparecimento',
'ordem'
    ]
    df = df.drop(columns=[col for col in colunas_para_remover if col in df.columns])

    #Limpeza de código IPHAN
    df['codigo_iphan'] = df['codigo_iphan'].astype(str).str.strip()
    df = df[df['codigo_iphan'].notna() & (df['codigo_iphan'] != '')]

    #Remoção correta de linhas com id_fiscalizacao nulo, vazio ou 'nan'
    df = df[df['id_fiscalizacao'].notna()]
    df['id_fiscalizacao'] = df['id_fiscalizacao'].astype(str).str.strip()
    df = df[(df['id_fiscalizacao'] != '') & (df['id_fiscalizacao'].str.lower() != 'nan')]


    #Padronizar tipo da coluna co_iphan para string
    sicg_df['co_iphan'] = sicg_df['co_iphan'].astype(str).str.strip()

    #Normalização da data
    def formatar_data(data):
        try:
            return pd.to_datetime(data).strftime('%d/%m/%Y')
        except:
```

```
        return ''
    documentos_df['data_criacao'] =
documentos_df['data_criacao'].apply(formatar_data)


    #Garantir que as chaves de merge estão no mesmo tipo (string)
    df['id_fiscalizacao'] = pd.to_numeric(df['id_fiscalizacao'],
errors='coerce').fillna(0).astype(int).astype(str)
    documentos_df['id_fiscalizacao'] =
pd.to_numeric(documentos_df['id_fiscalizacao'],
errors='coerce').fillna(0).astype(int).astype(str)


    #Agrupar documentos por id_fiscalizacao
    grupos = documentos_df.groupby('id_fiscalizacao').agg({
        'numero_documento': lambda x: '|'.join(map(str, x)),
        'data_criacao': lambda x: '|'.join(map(str, x))
    }).reset_index()
    df = pd.merge(df, grupos, on='id_fiscalizacao', how='left')

    #Verificação de correspondência com SICG
    codigos_iphan = set(df['codigo_iphan'])
    codigos_sicg = set(sicg_df['co_iphan'])
    codigos_nao_encontrados = codigos_iphan - codigos_sicg
    df['status_bem_sicg'] = df['codigo_iphan'].apply(
        lambda x: 'Bem não encontrado ou inativo' if x in codigos_nao_encontrados else
'Bem encontrado no SICG'
    )

    #Selecionar colunas desejadas dos dados tratados do sicg
    colunas_desejadas = [
        'bem_por_tipo', 'bem_tipo_protecao', 'id_bem', 'nome_bem', 'uf_estado', 'uf',
        'nome_municipio', 'uf_iso', 'regiao', 'ponto', 'longitude', 'latitude',
        'url', 'nome_agente', 'tipo_agente', 'total_acoes'
    ]
    sicg_df = sicg_df[[col for col in sicg_df.columns if col in colunas_desejadas or col
== 'co_iphan']]

    #Merge com SICG
    df = pd.merge(df, sicg_df, left_on='codigo_iphan', right_on='co_iphan', how='left')


    #Adiciona tipo_acao fixo
    df['tipo_acao'] = 'Ação do Processo Institucional Preservação: Fiscalização'

    # Renomeia 'id' para 'id_controle' e adiciona prefixo FISCALIS_
    if 'id' in df.columns:
        df.rename(columns={'id': 'id_controle'}, inplace=True)
        df['id_controle'] = 'FISCALIS_' + df['id_controle'].astype(str).str.strip()
```

```
    #Padronizações finais
    colunas_padronizar = [
        'id_fiscalizacao', 'id_setor', 'id_tipo_propriedade', 'id_tipo_regime_ocupacao',
        'bem_entorno', 'id_controle', 'id_bem', 'total_acoes',
'id_tipo_intervencao_irregularidade'
    ]
    for col in colunas_padronizar:
        if col in df.columns:
            df[col] = df[col].fillna('').astype(str).str.replace('.0', '', regex=False)

    df = df.applymap(lambda x: str(x).replace("""", " ") if pd.notna(x) else x)

    df = df.drop(columns=[
        'co_iphan',
        'id_tipo_regime_ocupacao',
        'id_tipo_propriedade',
        'id_tipo_intervencao_irregularidade',
        'id_fiscalizacao_pai',
        'perda_autenticidade',
        'perda_integridade',
        'justificativa_perda_autenticidade',
        'justificativa_perda_integridade',
        'justificativa_sem_imovel',
        'nome_sem_imovel'
    ])

    return df

def tratar_dados_acoes_incr() -> pd.DataFrame:


    df = pd.read_csv("util/base_inrc_projeto_identificacao.csv")

    #Garante que linhas onde 'post_title' está vazio ou nulo sejam removidas
    df = df[df['nome_projeto'].notnull() & (df['nome_projeto'].astype(str).str.strip() !=
"")]

    #Remove duplicatas com base em 'nome_projeto' + 'instituição_executora'
    df = df.drop_duplicates(subset=["nome_projeto", "instituicao_executora"])

    df.loc[:, 'ano_inicio'] = df['ano_inicio'].fillna('').astype(str).str.replace('.0', '',
regex=False)
    df.loc[:, 'ano_finalizacao'] =
df['ano_finalizacao'].fillna('').astype(str).str.replace('.0', '', regex=False)

    df['uf_estado'] = df['uf_estado'].astype(str)

    # Extrai a sigla (UF) para uma nova coluna
    df['uf'] = df['uf_estado'].str.extract(r'-\s*([A-Z]{2})$')
```

```
    # Remove o sufixo " - UF" da coluna uf_estado
    df['uf_estado'] = df['uf_estado'].str.replace(r'\s*-\s*[A-Z]{2}$', '', regex=True)

    # Extrai a sigla da UF da coluna 'uf_iso' (ex: "BR-AP" -> "AP")
    df['uf_iso'] = 'BR-' + df['uf']
    # Remove 'nan' caso tenha virado string durante o processo
    df['uf_estado'] = df['uf_estado'].replace('nan', '')
    df['regiao'] = df['regiao'].astype(str).str.replace(r'^Região\s+', '', regex=True)
    df['tipo_acao'] = 'Ação do processo institucional de identificação: INRC'

    df.rename(columns={'ID': 'id_controle'}, inplace=True)
    df['id_controle'] = 'INRC_' + df['id_controle'].astype(str).str.strip()
    df = df.drop(columns=['post_title'])
    df = df.drop(columns=['colecao_slug'])
    df = df.drop(columns=['item_slug'])

    #Remove as aspas simples: percorre todos os elementos do DataFrame, e se o
valor for uma string, ele substitui por espaco
    df = df.applymap(lambda x: x.replace("'", " ") if isinstance(x, str) else x)

    return df

#----------Carrega os dados no armazém de dados----------
def carregando_dados_armazem():
    start_time = time.time()

    df_final = pd.read_csv("util/dados_acoes_tratados.csv")

    conn = psycopg2.connect(
        host="xxxxxx",
        dbname="xxxxxxx",
        user="xxxxxxxx",
        password="xxxxxx"
    )
    cursor = conn.cursor()
    conn.autocommit = False

    id_colecao = 15
    id_metadado_id_controle = 136

    coluna_para_id_metadado = {
        "id_controle": 136,
        "id_fiscalizacao": 137,
        "codigo_iphan": 138,
        "id_setor": 139,
        "bem_entorno": 140,
        "tipo_estado_conservacao": 141,
        "tipo_estado_preservacao": 142,
        "tem_intervencao": 143,
```

```
        "irregularidade_encontrada": 144,
        "descricao_irregularidade": 145,
        "obra_em_andamento": 146,
        "obra_autorizada": 147,
        "obra_em_conformidade": 148,
        "existe_dano": 149,
        "tipo_dano": 150,
        "responsavel_identificado": 151,
        "proprietario_identificado": 152,
        "descricao_estado_atual_bem": 153,
        "dano": 154,
        "constatacao": 155,
        "numero_documento": 156,
        "data_criacao": 157,
        "status_bem_sicg": 158,
        "id_bem": 159,
        "bem_tipo_protecao": 160,
        "bem_por_tipo": 161,
        "nome_bem": 162,
        "ponto": 163,
        "longitude": 164,
        "latitude": 165,
        "url": 166,
        "regiao": 167,
        "uf_estado": 168,
        "uf": 169,
        "uf_iso": 170,
        "nome_municipio": 171,
        "nome_agente": 172,
        "tipo_agente": 173,
        "total_acoes": 174,
        "tipo_acao": 175,
        "nome_projeto": 176,
        "instituicao_executora": 177,
        "ano_inicio": 178,
        "ano_finalizacao": 179,
        "valor_orcamentario": 180
    }

    # Carrega os id_controle existentes
    cursor.execute("""
        SELECT i.id, v.value
        FROM obs_valor_metadado v
        JOIN obs_itens i ON v.id_valor = i.id
        WHERE v.id_metadado = %s AND i.id_colecao = %s
    """, (id_metadado_id_controle, id_colecao))
    id_controle_existentes = {valor.strip(): item_id for item_id, valor in
cursor.fetchall()}

    cursor.execute("SELECT id_valor, id_metadado FROM obs_valor_metadado")
```

```
    valores_existentes = {(id_valor, id_metadado) for id_valor, id_metadado in
cursor.fetchall()}

    insert_valores = []
    update_valores = []
    itens_atualizados = set()

    for i, row in df_final.iterrows():
        id_controle = str(row.get("id_controle")).strip()
        if not id_controle or id_controle.lower() == 'nan':
            continue

        if id_controle in id_controle_existentes:
            id_item = id_controle_existentes[id_controle]
            item_ja_existente = True
        else:
             post_title = row.get("nome_bem") or "Item sem título"
            cursor.execute("""
                INSERT INTO obs_itens (id_colecao, post_title)
                VALUES (%s, %s)
                RETURNING id
            """, (id_colecao, post_title))
            id_item = cursor.fetchone()[0]
            id_controle_existentes[id_controle] = id_item
            item_ja_existente = False

            insert_valores.append((id_item, id_metadado_id_controle, id_controle))

        for col, id_metadado in coluna_para_id_metadado.items():
            if col == "id_controle":
                continue

             valor = row.get(col)
            if pd.isna(valor) or str(valor).strip() == "":
                continue

            valor_str = str(valor).replace("""", " ").strip()
            chave = (id_item, id_metadado)

            if chave in valores_existentes:
                cursor.execute("""
                    SELECT value FROM obs_valor_metadado
                    WHERE id_valor = %s AND id_metadado = %s
                """, (id_item, id_metadado))
                valor_atual = cursor.fetchone()

                if valor_atual and valor_atual[0] != valor_str:
                    update_valores.append((valor_str, id_item, id_metadado))
                    if item_ja_existente:
                        itens_atualizados.add(id_item)
```

```
            else:
                insert_valores.append((id_item, id_metadado, valor_str))
                valores_existentes.add(chave)

        if i % 1000 == 0 and i > 0:
            if insert_valores:
                cursor.executemany("""
                    INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
                    VALUES (%s, %s, %s)
                """, insert_valores)
                insert_valores.clear()

            if update_valores:
                cursor.executemany("""
                    UPDATE obs_valor_metadado
                    SET value = %s
                    WHERE id_valor = %s AND id_metadado = %s
                """, update_valores)
                 update_valores.clear()

            conn.commit()
            print(f"Processadas {i} linhas...")

    if insert_valores:
        cursor.executemany("""
            INSERT INTO obs_valor_metadado (id_valor, id_metadado, value)
            VALUES (%s, %s, %s)
        """, insert_valores)

    if update_valores:
        cursor.executemany("""
            UPDATE obs_valor_metadado
            SET value = %s
            WHERE id_valor = %s AND id_metadado = %s
        """, update_valores)

    conn.commit()
    cursor.close()
    conn.close()

    print("Processamento concluído.")
    print(f"Total de itens atualizados: {len(itens_atualizados)}")
    elapsed_seconds = round(time.time() - start_time, 2)
    print(f"Tempo total de execução: {elapsed_seconds} segundos")

if __name__ == "__main__":
    #Etapa 1: Extração (realiza a extracao no inicio)

    #Etapa 2: Tratamento
    df_tratado_fiscalis = tratar_dados_acoes_fiscalis()
```

```
    df_tratado_fiscalis.to_csv('util/dados_acoes_tratados_fiscalis.csv', index=False)

    df_tratado_inrc = tratar_dados_acoes_incr()
    df_tratado_inrc.to_csv('util/dados_acoes_tratados_inrc.csv', index=False)

    #Junta os dois DataFrames mantendo todas as colunas
    df_unificado = pd.concat([df_tratado_fiscalis, df_tratado_inrc], ignore_index=True,
sort=False)
    df_unificado = df_unificado.fillna('')
    # Exporta o DataFrame com as Acoes do FISCALIS e INRC
    df_unificado.to_csv('util/dados_acoes_tratados.csv', index=False)

    #Etapa 3: Carga
    carregando_dados_armazem()
```

Fonte: elaborado pelos autores (2025).

e) View Implementada para o Indicador de Ações

A seguir, apresenta-se o código SQL usado para criação da view vw_indicador_acoes, implementada no Armazém de Dados.

Quadro 13 - View Vw_indicador_acoes

```
CREATE MATERIALIZED VIEW vw_indicador_acoes AS
SELECT
    i.id AS id_item,
    MAX(CASE WHEN m.nome = 'id_controle'              THEN v.value END) AS
id_controle,
    MAX(CASE WHEN m.nome = 'id_fiscalizacao'          THEN v.value END) AS
id_fiscalizacao,
    MAX(CASE WHEN m.nome = 'codigo_iphan'             THEN v.value END) AS
codigo_iphan,
    MAX(CASE WHEN m.nome = 'id_setor'                 THEN v.value END) AS id_setor,
    MAX(CASE WHEN m.nome = 'bem_entorno'              THEN v.value END) AS
bem_entorno,
    MAX(CASE WHEN m.nome = 'tipo_estado_conservacao'  THEN v.value END) AS
tipo_estado_conservacao,
    MAX(CASE WHEN m.nome = 'tipo_estado_preservacao'  THEN v.value END) AS
tipo_estado_preservacao,
    MAX(CASE WHEN m.nome = 'tem_intervencao'          THEN v.value END) AS
tem_intervencao,
    MAX(CASE WHEN m.nome = 'irregularidade_encontrada' THEN v.value END) AS
irregularidade_encontrada,
    MAX(CASE WHEN m.nome = 'descricao_irregularidade' THEN v.value END) AS
descricao_irregularidade,
    MAX(CASE WHEN m.nome = 'obra_em_andamento'        THEN v.value END) AS
obra_em_andamento,
```

```
  MAX(CASE WHEN m.nome = 'obra_autorizada'          THEN v.value END) AS
obra_autorizada,
  MAX(CASE WHEN m.nome = 'obra_em_conformidade'     THEN v.value END) AS
obra_em_conformidade,
  MAX(CASE WHEN m.nome = 'existe_dano'              THEN v.value END) AS
existe_dano,
  MAX(CASE WHEN m.nome = 'tipo_dano'                THEN v.value END) AS
tipo_dano,
  MAX(CASE WHEN m.nome = 'responsavel_identificado'  THEN v.value END) AS
responsavel_identificado,
  MAX(CASE WHEN m.nome = 'proprietario_identificado' THEN v.value END) AS
proprietario_identificado,
  MAX(CASE WHEN m.nome = 'descricao_estado_atual_bem' THEN v.value END) AS
descricao_estado_atual_bem,
  MAX(CASE WHEN m.nome = 'dano'                     THEN v.value END) AS dano,
  MAX(CASE WHEN m.nome = 'constatacao'              THEN v.value END) AS
constatacao,
  MAX(CASE WHEN m.nome = 'numero_documento'         THEN v.value END) AS
numero_documento,
  MAX(CASE WHEN m.nome = 'data_criacao'             THEN v.value END) AS
data_criacao,
  MAX(CASE WHEN m.nome = 'status_bem_sicg'          THEN v.value END) AS
status_bem_sicg,
  MAX(CASE WHEN m.nome = 'id_bem'                   THEN v.value END) AS id_bem,
  MAX(CASE WHEN m.nome = 'bem_tipo_protecao'        THEN v.value END) AS
bem_tipo_protecao,
  MAX(CASE WHEN m.nome = 'bem_por_tipo'             THEN v.value END) AS
bem_por_tipo,
  MAX(CASE WHEN m.nome = 'nome_bem'                 THEN v.value END) AS
nome_bem,
  MAX(CASE WHEN m.nome = 'ponto'                    THEN v.value END) AS ponto,
  MAX(CASE WHEN m.nome = 'longitude'                THEN v.value END) AS
longitude,
  MAX(CASE WHEN m.nome = 'latitude'                 THEN v.value END) AS latitude,
  MAX(CASE WHEN m.nome = 'url'                      THEN v.value END) AS url,
  MAX(CASE WHEN m.nome = 'regiao'                   THEN v.value END) AS regiao,
  MAX(CASE WHEN m.nome = 'uf_estado'                THEN v.value END) AS
uf_estado,
  MAX(CASE WHEN m.nome = 'uf'                       THEN v.value END) AS uf,
  MAX(CASE WHEN m.nome = 'uf_iso'                   THEN v.value END) AS uf_iso,
  MAX(CASE WHEN m.nome = 'nome_municipio'           THEN v.value END) AS
nome_municipio,
  MAX(CASE WHEN m.nome = 'nome_agente'              THEN v.value END) AS
nome_agente,
  MAX(CASE WHEN m.nome = 'tipo_agente'              THEN v.value END) AS
tipo_agente,
  MAX(CASE WHEN m.nome = 'total_acoes'              THEN v.value END) AS
total_acoes,
  MAX(CASE WHEN m.nome = 'tipo_acao'                THEN v.value END) AS
tipo_acao,
```

```
   MAX(CASE WHEN m.nome = 'nome_projeto'              THEN v.value END) AS
nome_projeto,
   MAX(CASE WHEN m.nome = 'instituicao_executora'     THEN v.value END) AS
instituicao_executora,
   MAX(CASE WHEN m.nome = 'ano_inicio'                THEN v.value END) AS
ano_inicio,
   MAX(CASE WHEN m.nome = 'ano_finalizacao'           THEN v.value END) AS
ano_finalizacao,
   MAX(CASE WHEN m.nome = 'valor_orcamentario'        THEN v.value END) AS
valor_orcamentario
FROM obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE i.id_colecao = 15
GROUP BY i.id;
```

Fonte: elaborado pelos autores (2025).

f) Consulta SQL implementada no Apache Superset para gerar o DATASET do Indicador de Ações

O select abaixo é responsável por recuperar os dados da view vw_indicador_acoes, utilizada como base para visualizações no Superset. Além dos metadados consolidados, o select também trata:

- Geração do link clicáveis para o campo url convertido em um link HTML único.

Esse select é usado diretamente na criação do dataset no Superset, para os dashboards de ações. Para garantir uma contagem precisa das informações, utilizamos o metadado id_controle, presente em ambas as ações, como controle em operações de COUNT(DISTINCT) quando necessário. A seguir, apresenta-se a consulta SQL da view vw_indicador_acoes no Superset.

Quadro 14 - Consulta Sql Da View Vw_indicador_acoes

```
SELECT
   *,
   REGEXP_REPLACE(total_acoes::TEXT, '\.0$', '') AS total_acoes_normalizado,
   REGEXP_REPLACE(ano_finalizacao::TEXT, '\.0$', '') AS
ano_finalizacao_normalizado,
   REGEXP_REPLACE(ano_inicio::TEXT, '\.0$', '') AS ano_inicio_normalizado,
   REGEXP_REPLACE(id_fiscalizacao::TEXT, '\.0$', '') AS id_fiscalizacao_normalizado,

   CASE
```

```
    WHEN url IS NOT NULL AND url != ''
    THEN CONCAT('<a href="', url, '" target="_blank">Acessar</a>')
    ELSE NULL
  END AS link_url
FROM vw_indicador_acoes;
```

Fonte: elaborado pelos autores (2025).

g) Implementação da POP-UP no Superset - Ações

Os códigos JavaScript utilizados na implementação da POP-UP no mapa com a distribuição geográfica das ações de fiscalização por bens culturais, materiais, arqueológicos e ferroviários, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator

Quadro 15 - Configuração do JavaScript onClick para o mapa com a distribuição geográfica das ações de fiscalização por bens culturais

```
d => `
 <div>
 <strong>${d.object.extraProps.nome_bem}</strong>
 </div>
 <div>
  Proteção do Bem: <strong>${d.object.extraProps.bem_tipo_protecao}</strong>
 </div>
 <div>
  Tipo de Ação: <strong>${d.object.extraProps.tipo_acao}</strong>
 </div>
 <div>
  Quantidade de Ações: <strong>${d.object.extraProps.total_acoes}</strong>
 </div>
 <div>
  Agente: <strong>${d.object.extraProps.nome_agente}</strong>
 </div>
 <div>
  Estado (UF): <strong>${d.object.extraProps.uf}</strong>
 </div>
 <div>
  Município: <strong>${d.object.extraProps.nome_municipio}</strong>
 </div>
 <div>
  Clique no Ponto e Saiba Mais
 </div>
`
```

Fonte: elaborado pelos autores (2025).

Com estes scripts se materializa a solução que o projeto de pesquisa elaborou para concatenar os diversos dados culturais oriundos das diversas fontes levantadas, com o objetivo de criar uma visão mais completa da área do Patrimônio Cultural, fortalecendo o SNPC. Agora será relatada outra inovação vinda do projeto, o processo de automação das atualizações das views.

3.1.3.1.4 Views Materializadas

A automação da atualização das views materializadas, as vw_indicador_bens, vw_indicador_agentes, vw_indicador_acoes não foram aplicadas, uma vez que a definição da política de atualização periódica por parte da equipe do Iphan não foi estabelecida. Essa definição estava prevista para ocorrer no âmbito do projeto; contudo, o encerramento antecipado impossibilitou sua execução conforme o cronograma original. Entretanto, abaixo apresenta-se um exemplo técnico de como essa automação pode ser implementada, considerando uma periodicidade de 6 em 6 meses:

- Crie um índice para acelerar consultas na view:

Quadro 16 - Índice Para Acelerar Consultas Na View

```
CREATE INDEX idx_vw_indicador_bens_id ON vw_indicador_bens(id_item);
```

Fonte: Elaborado pelos autores (2025).

- Crie uma função de atualização da view:

Quadro 17 - Função De Atualização Da View

```
CREATE OR REPLACE FUNCTION atualizar_vw_indicador_bens()
RETURNS void AS $$
BEGIN
   REFRESH MATERIALIZED VIEW CONCURRENTLY vw_indicador_bens;
END;
$$ LANGUAGE plpgsql;
```

Fonte: Elaborado pelos autores (2025).

- Agende a execução automática via pg_cron (extensão do PostgreSQL). Abaixo, apresenta-se um exemplo de agendamento automático para execução nos dias 2 de junho e 2 de dezembro, às 3h da manhã:

Quadro 18 - Agendamento Automático

```
SELECT cron.schedule(
  'atualizar_view_bens_6m',
  '0 3 2 6,12 *',  -- minuto hora dia mês dia_da_semana
  $$ SELECT atualizar_vw_indicador_bens(); $$
);
```

Fonte: elaborado pelos autores (2025).

Nota: Essa atualização deve ocorrer somente após a execução dos scripts de extração, tratamento e carregamento, garantindo que os dados estejam atualizados no banco de dados do armazém antes da atualização da view.

- Caso necessário, a atualização da view também pode ser feita manualmente com o seguinte comando:

Quadro 19 - Atualização Manualmente

```
REFRESH MATERIALIZED VIEW vw_indicador_bens;
```

Fonte: elaborado pelos autores (2025).

3.1.3.2 Dados das Fontes Externas

Além disso, foram coletados, tratados e inseridos no armazém de dados do OBS conjuntos de dados provenientes de fontes externas relevantes ao escopo da pesquisa realizada no âmbito do projeto. Esses dados complementares têm como objetivo subsidiar análises territoriais e institucionais relacionadas à proteção e valorização do patrimônio cultural brasileiro.

Entre as fontes externas integradas, têm-se: dados sobre legislações brasileiras criadas para subsidiar, proteger e fomentar o patrimônio cultural em diferentes esferas públicas; os limites das Unidades de Conservação Federais disponibilizados pelo Instituto Chico Mendes de Conservação da Biodiversidade (ICMBio); as terras reconhecidas como territórios quilombolas, conforme dados do Instituto Nacional de Colonização e Reforma Agrária (INCRA); as terras indígenas demarcadas oficialmente pela Fundação Nacional dos Povos Indígenas (FUNAI); e os bens culturais localizados em território brasileiro reconhecidos como Patrimônio Mundial pela Organização das Nações Unidas para a Educação, a Ciência e

a Cultura (UNESCO). Dessa forma, esses dados foram coletados, tratados e inseridos no armazém de dados do OBS.

Os dados de territórios quilombolas foram coletados separadamente por estado (UF), exigindo consolidação em um arquivo nacional único antes do tratamento. O Quadro abaixo apresenta-se o script desenvolvido para consolidar os dados estaduais em um único arquivo chamado “Áreas_de_Quilombolas_INCRA.csv”.

Quadro 20 - Pré-processamento dos dados coletados do INCRA

```
import os
import pandas as pd
from glob import glob

os.environ['GDAL_DATA'] = r'C:\Users\Tathiana\miniconda3\Library\share\gdal'

input_dir = "input_data"
filename = "Áreas de Quilombolas_TO.shp"
input_path = os.path.join(input_dir, filename)

output_dir = "output_data"

arquivos_csv = glob(os.path.join(input_dir, "Áreas de Quilombolas_*.csv"))

if not arquivos_csv:
  print("Nenhum arquivo 'Áreas de Quilombolas_*.csv' encontrado em", input_dir)
else:
  dfs = []
  for arquivo in arquivos_csv:
    try:
      df = pd.read_csv(arquivo, encoding='utf-8')
      print(f"\nArquivo: {arquivo}")
      print(f"Linhas: {len(df)}")
      dfs.append(df[['cd_uf', 'nm_comunid', 'perimetro_', "wkt_geometry"]])
      print(f"Sucesso: {os.path.basename(arquivo)}")
    except Exception as e:
      print(f"Erro ao ler {arquivo}: {str(e)}")
      continue


  if dfs:
    df_final = pd.concat(dfs, ignore_index=True)

    print("\nConsolidação:")
    print(f"- Total de registros: {len(df_final)}")

    output_path = os.path.join(input_dir, "Áreas_de_Quilombolas_INCRA.csv")
    df_final.to_csv(output_path, index=False, encoding='utf-8')
```

```
    print(f"\nArquivo consolidado salvo em:\n{output_path}")
  else:
    print("Nenhum DataFrame válido para consolidar.")
```

Fonte: elaborado pelos autores (2025).

No quadro abaixo apresenta-se o script utilizado no tratamento dos dados coletados das seguintes fontes: ICMBio, FUNAI e INCRA. Os dados geográficos estão disponibilizados no formato WKT Para viabilizar sua utilização no Superset, foi necessário converter esses dados do formato WKT para arrays de coordenadas no padrão [[[longitude, latitude], ...]], compatível com os recursos de visualização de mapas do Superset.

Quadro 21 - Script para converter os dados geográficos

```
import pandas as pd
import re
from pathlib import Path
import os

def wkt_to_coord_array(wkt_str):
  """
  Converte WKT para array de coordenadas no formato:
  [[[lon, lat], [lon, lat], ...]]  (mesmo formato para POLYGON e MULTIPOLYGON)
  """
  if not isinstance(wkt_str, str):
    return None

  try:
    wkt_clean = re.sub(r'\s+', ' ', wkt_str.strip())


    coord_groups = re.findall(r'\(([^()]+)\)', wkt_clean)


    if not coord_groups:
      return None

    result = []
    for group in coord_groups:
      coord_pairs = re.findall(r'([-+]?\d*\.\d+|\d+)\s+([-+]?\d*\.\d+|\d+)', group)


      coords = [[float(lon), float(lat)] for lon, lat in coord_pairs]

      if coords:
        if coords[0] != coords[-1]:
```

```
            coords.append(coords[0])
          result.extend(coords)


    return [result] if result else None

  except Exception as e:
    print(f"Erro ao processar WKT: {wkt_str[:50]}... - {str(e)}")
    return None

def process_geometries_to_csv(filename, input_dir, output_dir, output_cols,
geom_col='geometry', output_prefix='superset_'):
  """
  Processa arquivos CSV convertendo WKT para arrays de coordenadas
  """
  Path(output_dir).mkdir(parents=True, exist_ok=True)

  if filename.endswith('.csv'):
    input_path = os.path.join(input_dir, filename)
    output_path = os.path.join(output_dir, f"{output_prefix}{filename}")

    print(f"Processando {filename}...")

    try:
      df = pd.read_csv(input_path)

      if geom_col not in df.columns:
        print(f"Aviso: Coluna '{geom_col}' não encontrada")

      df['coordinates'] = df[geom_col].apply(wkt_to_coord_array)

      df_valid = df.dropna(subset=['coordinates'])
      print(f"Linhas com geometria válida: {len(df_valid)}/{len(df)}")


      df_valid[output_cols].to_csv(output_path, index=False, encoding='utf-8')


    except Exception as e:
      print(f"Erro ao processar {filename}: {str(e)}")

if __name__ == "__main__":
  # ICMBio
  FILENAME = "ICMBio_limiteucsfederais_a.csv"
  INPUT_DIR = "input_data"
  OUTPUT_DIR = "output_data"
  output_cols = ['nomeuc', 'ufabrang', 'perimm', 'coordinates']
  process_geometries_to_csv(
    filename=FILENAME,
    input_dir=INPUT_DIR,
```

```
    output_dir=OUTPUT_DIR,
    output_cols=output_cols,
    geom_col="the_geom",
    output_prefix="tratado_"
)

# FUNAI
FILENAME = "FUNAI_tis_poligonais.csv"
INPUT_DIR = "input_data"
OUTPUT_DIR = "output_data"
output_cols = ['terrai_nome', 'uf_sigla', 'superficie_perimetro_ha', 'coordinates']
process_geometries_to_csv(
    filename=FILENAME,
    input_dir=INPUT_DIR,
    output_dir=OUTPUT_DIR,
    output_cols=output_cols,
    geom_col="the_geom",
    output_prefix="tratado_"
)

# INCRA
FILENAME = "Áreas_de_Quilombolas_INCRA.csv"
INPUT_DIR = "input_data"
OUTPUT_DIR = "output_data"
output_cols = ['cd_uf', 'nm_comunid', 'perimetro_', 'coordinates']
process_geometries_to_csv(
    filename=FILENAME,
    input_dir=INPUT_DIR,
    output_dir=OUTPUT_DIR,
    output_cols=output_cols,
    geom_col="wkt_geometry",
    output_prefix="tratado_"
)
```

Fonte: elaborado pelos autores (2025).

Além disso, a partir dos dados disponibilizados pela UNESCO[1], foram selecionados os Patrimônios Mundiais localizados no Brasil. No quadro abaixo, apresenta-se o trecho do script utilizado para realizar essa filtragem por meio do parâmetro iso_code = “br”. Após a extração, os dados são traduzidos para o português do Brasil.

Quadro 22 - Script para extração dos Patrimônios Mundiais localizados no Brasil

```
import pandas as pd

# Carrega o arquivo xlsx
arquivo_xlsx = 'whc-sites-2024.xlsx'
```

[1] Site da UNESCO. Disponível em: https://whc.unesco.org/. Acesso em: 12 ago. 2025.

```
df = pd.read_excel(arquivo_xlsx)

# Filtra as linhas onde iso_code é 'br'
df_filtrado = df[df['iso_code'] == 'br']

# Define as colunas desejadas
colunas_desejadas = [
   'unique_number', 'id_no', 'name_es', 'date_inscribed', 'longitude', 'latitude',
'area_hectares',
   'category', 'iso_code', 'short_description_es', 'justification_en'
]

# Seleciona as colunas
df_final = df_filtrado[colunas_desejadas]

# Salva em um arquivo CSV
arquivo_csv = 'unesco.csv'
df_final.to_csv(arquivo_csv, index=False, encoding='utf-8')

print(f"Arquivo {arquivo_csv} salvo com sucesso!")
print(df_final.head())
```

Fonte: elaborado pelos autores (2025).

Portanto, a seguir, apresenta-se como essas informações estão sendo recuperadas por meio do Superset, na seguinte ordem:

1) Indicador Legislação
2) UNESCO
3) INCRA
4) FUNAI
5) ICMbio

3.1.3.2.1 Indicador - Legislação

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do Indicador de Recorte das Legislações:

Quadro 23 - Select - Indicador Legislação

```
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'tipologia_normativa' THEN v.value END) AS
tipologia_normativa,
   MAX(CASE WHEN m.nome = 'total' THEN v.value END) AS total,
```

```
   MAX(CASE WHEN m.nome = 'porcentagem' THEN v.value END) AS porcentagem,
   MAX(CASE WHEN m.nome = 'legislacao_federal' THEN v.value END) AS
legislacao_federal,
   MAX(CASE WHEN m.nome = 'legislacao_distrital' THEN v.value END) AS
legislacao_distrital,
   MAX(CASE WHEN m.nome = 'legislacao_municipal' THEN v.value END) AS
legislacao_municipal,
   MAX(CASE WHEN m.nome = 'legislacao_estadual' THEN v.value END) AS
legislacao_estadual,
   MAX(CASE WHEN m.nome = 'bem_por_tipo' THEN v.value END) AS bem_por_tipo
FROM obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE i.id_colecao = 4
GROUP BY i.id;
```

Fonte: elaborado pelos autores (2025).

Além disso, para recuperar o Indicador de Recorte das Legislações por Estado (UF) utilizando o padrão ISO, foi implementada a seguinte consulta SQL no armazém de dados:

Quadro 24 - Consulta SQL - Legislação

```
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'Legislação' THEN v.value END) AS legislacao,
   MAX(CASE WHEN m.nome = 'bem_por_tipo' THEN v.value END) AS bem_por_tipo,
   MAX(CASE WHEN m.nome = 'UF_ISO' THEN v.value END) AS uf_iso
FROM obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE i.id_colecao = 5
GROUP BY i.id;
```

Fonte: elaborado pelos autores (2025).

3.1.3.2.2 Mapa - UNESCO

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do mapa com os pontos dos Patrimônios Mundiais no Brasil reconhecidos pela UNESCO:

Quadro 25 - Select - Indicador Património Mundial

```
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'unique_number'      THEN v.value END) AS
unique_number,
   MAX(CASE WHEN m.nome = 'id_no'          THEN v.value END) AS id_no,
```

```
  MAX(CASE WHEN m.nome = 'date_inscribed'     THEN v.value END) AS
date_inscribed,
  MAX(CASE WHEN m.nome = 'longitude'          THEN v.value END) AS longitude,
  MAX(CASE WHEN m.nome = 'latitude'           THEN v.value END) AS latitude,
  MAX(CASE WHEN m.nome = 'area_hectares'      THEN v.value END) AS
area_hectares,
  MAX(CASE WHEN m.nome = 'category'           THEN v.value END) AS category,
  MAX(CASE WHEN m.nome = 'iso_code'           THEN v.value END) AS iso_code,
  MAX(CASE WHEN m.nome = 'name'               THEN v.value END) AS name,
  MAX(CASE WHEN m.nome = 'short_description'  THEN v.value END) AS
short_description,
  MAX(CASE WHEN m.nome = 'justification'      THEN v.value END) AS justification,
  MAX(CASE WHEN m.nome = 'estado_uf'          THEN v.value END) AS estado_uf
FROM
  obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE
  i.id_colecao = 8
GROUP BY
  i.id
```

Fonte: elaborado pelos autores (2025).

Os códigos JavaScript utilizados na implementação da POP-UP no mapa de Patrimônios Mundiais no Brasil, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator

Quadro 26 - Configuração do JavaScript tooltip - 2

```
d => `
 <div>
 <strong>${d.object.extraProps.name}</strong>
 </div>
 <div>
  Estado (UF): <strong>${d.object.extraProps.estado_uf}</strong>
 </div>
 <div>
  Área em Hectares: <strong>${d.object.extraProps.area_hectares}</strong>
 </div>
 <div>
  Categoria: <strong>${d.object.extraProps.category}</strong>
 </div>
 <div>
  Data de Inscrição: <strong>${d.object.extraProps.date_inscribed}</strong>
 </div>
 <div>
```

```
   Descrição: <strong>${d.object.extraProps.short_description} (Fonte:
UNESCO)</strong>
  </div>
  </div>
  Clique no ponto e Saiba mais
  </div>
`
```

Fonte: elaborado pelos autores (2025).

- Configuração da POP-UP em: JavaScript onClick href

Quadro 27 - Configuração do JavaScript onClick 2

```
d => 'https://whc.unesco.org/'
```

Fonte: elaborado pelos autores (2025).

3.1.3.2.3 Mapa - INCRA

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do mapa com os polígonos geográficos das terras reconhecidas como territórios quilombolas, conforme dados do INCRA. Para otimizar o desempenho da consulta no Apache Superset, foi implementada a view materializada "vw_poligonos_territorios_quilombolas_incra" para esses dados. Conforme apresentado a seguir:

- View implementada:

Quadro 28 - View - Territórios Quilombolas

```
CREATE MATERIALIZED VIEW vw_poligonos_territorios_quilombolas_incra AS
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'nm_comunid' THEN v.value END) AS nm_comunid,
   MAX(CASE WHEN m.nome = 'uf'        THEN v.value END) AS uf,
   MAX(CASE WHEN m.nome = 'perimetro_' THEN v.value END) AS perimetro,
   MAX(CASE WHEN m.nome = 'coordinates' THEN v.value END) AS coordinates
FROM
   obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE
   i.id_colecao = 6
GROUP BY
   i.id;
```

Fonte: elaborado pelos autores (2025).

- Consulta SQL implementada no Apache Superset para gerar o DATASET:

Quadro 29 - Select dataset - Território Quilombola

```
SELECT * FROM vw_poligonos_territorios_quilombolas_incra;
```

Fonte: Elaborado pelos autores (2025).

O código JavaScript utilizado na implementação da POP-UP no mapa de Territórios Indígenas, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator.

Quadro 30 - Configuração do JavaScript tooltip 4

```
d => `
  <div>
  <strong>${d.object.extraProps.nm_comunid}</strong>
  </div>
  <div>
   UF: <strong>${d.object.extraProps.uf}</strong>
  </div>
`
```

Fonte: elaborado pelos autores (2025).

3.1.3.2.4 Mapa - FUNAI

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do mapa com os polígonos geográficos dos territórios indígenas, conforme dados da FUNAI. Para otimizar o desempenho da consulta no Apache Superset, foi implementada a view materializada "vw_poligonos_terras_indigenas_funai" para esses dados. Conforme apresentado a seguir:

- View implementada:

Quadro 31 - View - Território Indigena

```
CREATE MATERIALIZED VIEW vw_poligonos_terras_indigenas_funai AS
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'terrai_nome' THEN v.value END) AS terrai_nome,
   MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
   MAX(CASE WHEN m.nome = 'superficie_perimetro_ha' THEN v.value END) AS
superficie_perimetro_ha,
   MAX(CASE WHEN m.nome = 'coordinates' THEN v.value END) AS coordinates
```

```
FROM obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE i.id_colecao = 7
GROUP BY i.id;
```

Fonte: Elaborado pelos autores (2025).

- Consulta SQL implementada no Apache Superset para gerar o DATASET:

Quadro 32 - Select dataset - Território Indigena

```
SELECT * FROM vw_poligonos_terras_indigenas_funai;
```

Fonte: Elaborado pelos autores (2025).

O código JavaScript utilizado na implementação da POP-UP no mapa de Territórios Indígenas, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator

Quadro 33 - Configuração do JavaScript tooltip 3

```
d => `
  <div>
  <strong>${d.object.extraProps.terrai_nome}</strong>
  </div>
  <div>
   UF: <strong>${d.object.extraProps.uf}</strong>
  </div>
`
```

Fonte: elaborado pelos autores (2025).

3.1.3.2.5 Mapa - ICMBio

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do mapa com os polígonos geográficos dos limites de unidades de conservação federais, conforme dados do ICMBio. Para otimizar o desempenho da consulta no Apache Superset, foi implementada a view materializada "vw_poligonos_limites_unidades_conservacao_icmbio" para esses dados. Conforme apresentado a seguir:

- View implementada:

Quadro 34 - View - ICMbio

```
CREATE MATERIALIZED VIEW
vw_poligonos_limites_unidades_conservacao_icmbio AS
SELECT
   i.id AS id_item,
   MAX(CASE WHEN m.nome = 'nome' THEN v.value END) AS nome,
   MAX(CASE WHEN m.nome = 'uf' THEN v.value END) AS uf,
   MAX(CASE WHEN m.nome = 'criacaoano' THEN v.value END) AS criacaoano,
   MAX(CASE WHEN m.nome = 'coordinates' THEN v.value END) AS coordinates
FROM obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE i.id_colecao = 9
GROUP BY i.id;
```

Fonte: Elaborado pelos autores (2025).

- Consulta SQL implementada no Apache Superset para gerar o DATASET:

Quadro 35 - Select dataset - ICMbio

```
SELECT * FROM vw_poligonos_limites_unidades_conservacao_icmbio;
```

Fonte: Elaborado pelos autores (2025).

O código JavaScript utilizado na implementação da POP-UP no mapa de Limites de Unidades de Conservação Federais, são apresentados a seguir:

- Configuração da POP-UP em: JavaScript tooltip generator

Quadro 36 - Configuração do JavaScript onClick 5

```
d => `
 <div>
 <strong>${d.object.extraProps.nome}</strong>
 </div>
 <div>
  UF: <strong>${d.object.extraProps.uf}</strong>
 </div>
 <div>
  Ano de criação: <strong>${d.object.extraProps.criacaoano}</strong>
 </div>
`
```

Fonte: elaborado pelos autores (2025).

3.1.3.2.6 Dados de Categorização Municipal

Além disso, foram coletados dados sobre a categorização dos municípios que se trata de um indicador do portal do turismo. Conforme evidenciado no relatório de metas 3, a categorização municipal é um indicador desenvolvido pelo Ministério do Turismo que avalia o desempenho da economia turística nos municípios incluídos no mapa do turismo brasileiro. Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET.

Quadro 37 - Select - Indicador Categorização Municipal

```
SELECT
  i.id AS id_item,
  MAX(CASE WHEN m.nome = 'MACRO'                    THEN v.value END) AS
macro,
  MAX(CASE WHEN m.nome = 'UF'                    THEN v.value END) AS uf,
  MAX(CASE WHEN m.nome = 'REGIAO_TURISTICA'                THEN v.value
END) AS regiao_turistica,
  MAX(CASE WHEN m.nome = 'MUNICIPIO'                THEN v.value END)
AS municipio,
  MAX(CASE WHEN m.nome = 'COD_IBGE'                THEN v.value END) AS
cod_ibge,
  MAX(CASE WHEN m.nome = 'QUANTIDADE_EMPREGOS'                THEN
v.value END) AS quantidade_empregos,
  MAX(CASE WHEN m.nome = 'QUANTIDADE_ ESTABELECIMENTOS'
THEN v.value END) AS quantidade_estabelecimentos,
  MAX(CASE WHEN m.nome = 'QUANTIDADE_VISITAS_ESTIMADAS_
INTERNACIONAL'  THEN v.value END) AS visitas_estimadas_internacional,
  MAX(CASE WHEN m.nome = 'QUANTIDADE_VISITAS_ESTIMADAS_ NACIONAL'
THEN v.value END) AS visitas_estimadas_nacional,
  MAX(CASE WHEN m.nome = 'ARRECADACAO'                THEN v.value
END) AS arrecadacao,
  MAX(CASE WHEN m.nome = 'CLUSTER'                THEN v.value END) AS
cluster
FROM
  obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE
  i.id_colecao = 13
GROUP BY
  i.id;
```

Fonte: Elaborado pelos autores (2025).

3.1.3.2.7 Polígonos dos Estados (UF) Brasileiros

Abaixo apresenta-se a consulta SQL implementada no Apache Superset, com acesso ao armazém de dados, utilizada para gerar o DATASET do mapa com os polígonos geográficos dos estados brasileiros. Para otimizar o desempenho no Apache Superset, foi implementada a view materializada "vw_poligonos_unidades_federativas_ brasileiras" para esses dados. Conforme apresentado a seguir:

- View implementada:

Quadro 38 - View - Polígonos dos Estados (UF) Brasileiros

```
CREATE MATERIALIZED VIEW vw_poligonos_unidades_federativas_brasileiras AS
SELECT
  i.id AS id_item,
  MAX(CASE WHEN m.nome = 'nome'           THEN v.value END) AS nome,
  MAX(CASE WHEN m.nome = 'uf_sigla'       THEN v.value END) AS uf_sigla,
  MAX(CASE WHEN m.nome = 'uf_iso'         THEN v.value END) AS uf_iso,
  MAX(CASE WHEN m.nome = 'regiao'         THEN v.value END) AS regiao,
  MAX(CASE WHEN m.nome = 'coordinates_json' THEN v.value END) AS
coordinates_json,
  MAX(CASE WHEN m.nome = 'num_pontos'     THEN v.value END) AS num_pontos
FROM
  obs_itens i
LEFT JOIN obs_valor_metadado v ON i.id = v.id_valor
LEFT JOIN obs_metadados m ON v.id_metadado = m.id
WHERE
  i.id_colecao = 12
GROUP BY
  i.id;
```

Fonte: Elaborado pelos autores (2025).

- Consulta SQL implementada no Apache Superset para gerar o DATASET:

Quadro 39 - Select dataset - Polígonos dos Estados (UF) Brasileiros

```
SELECT * FROM vw_poligonos_unidades_federativas_brasileiras;
```

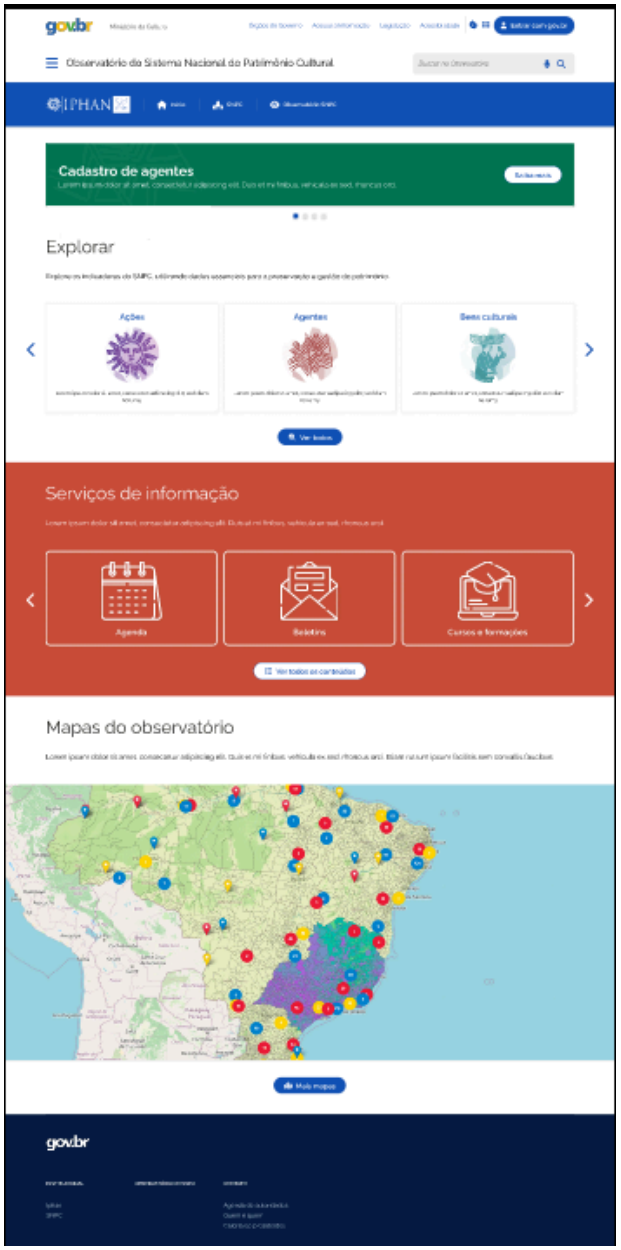
Fonte: Elaborado pelos autores (2025).

3.1.4 Testes e aprovação dos módulos

A implementação da primeira versão do ambiente de homologação foi realizada no dia 14 de Março, desde então vem sendo implementando atualizações e melhorias para ele.

Devido algumas dificuldades de alinhamento a respeito do design do portal, optou-se em manter o *Design System* do Gov para se manter nas diretrizes do governo digital. Para fins de documentação das modificações será apresentado o primeiro protótipo do Observatório e posteriormente as modificações. O primeiro protótipo foi elaborado desta forma:

Figura 1 - FIGMA do primeiro Protótipo do OBS

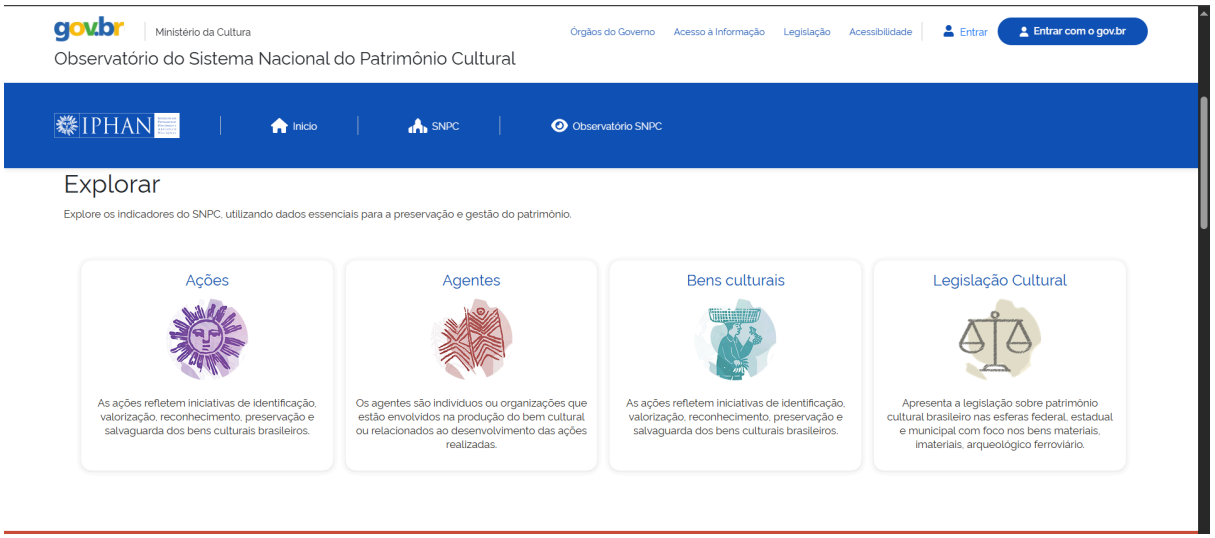


Fonte: Observatório do SNPC (2025).

Este foi o design finalista, uma vez que está alinhado com o design system do governo. Esta foi a primeira versão do observatório. Em suma, o design foi mantido, foram adicionados novos elementos. O primeiro está na seção dos indicadores,antes apenas

contendo os 3 indicadores principais (Ações, Agentes, Bens Culturais), agora foi adicionado um novo indicador de “Legislação Cultural”. Conforme pode ser observado na figura abaixo:

Figura 2 - Seção indicadores

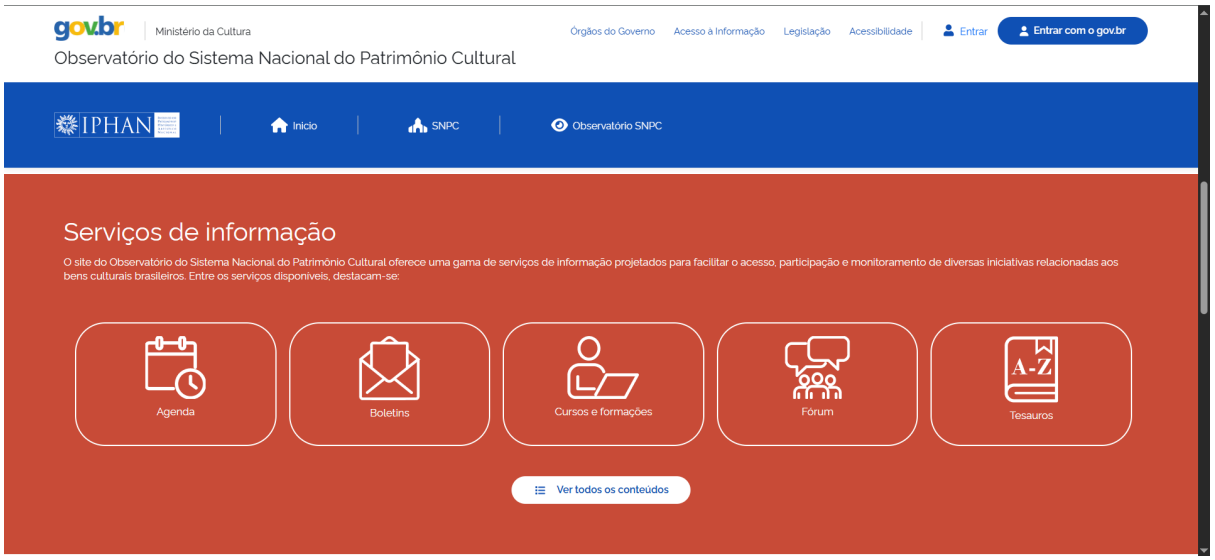


Fonte: Observatório do SNPC (2025).

Com este novo indicador foi elaborado uma nova arte e um texto explicativo. Por ter uma relevância no monitoramento do setor, teve essa ideia de deixá-lo em um lugar de destaque, a elaboração deste indicador foi apresentado no relatório de metas 3.

Continuando com a série de aprimoramentos que a página sofreu. Na seção de Serviços de informação, também ocorreu uma modificação, apenas continha 3 serviços, Agenda, Boletins e Cursos e Formações. Agora foram implementados dois novos serviços, o de tesauro (devidamente relatado no relatório de metas 3), e o Fórum. Conforme pode ser observado na figura abaixo:

Figura 3 - Serviços de informação



Fonte: Observatório do SNPC (2025).

Por fim, um dos últimos aprimoramentos foi na seção de mapas. Durante a prospecção de softwares, o Visão foi apresentado como uma solução para visualização geográfica. No entanto, após alinhamento com a equipe do IPHAN, optou-se por utilizar exclusivamente o Visão para a elaboração e apresentação dos dados geográficos. Com isso, os mapas inicialmente desenvolvidos no Superset foram migrados para o Visão. O resultado final pode ser visto na figura a seguir.

Figura 4 - Mapas

Fonte: Observatório do SNPC (2025).

Devido ao encerramento prematuro do projeto, a parte de validação de todos os módulos pela equipe responsável do IPHAN ficou prejudicada. Visto que, a meta 3 e 4 foram

realizadas em um período inferior ao esperado. Dito isso, não houve de maneira oficial a validação dos módulos.

3.2 Implementação do Ambiente de produção no IPHAN

A etapa 3.2 consiste na instalação dos sistemas e serviços no ambiente de produção do IPHAN. Essa fase é transição do projeto da fase de testes para o uso institucional, permitindo que as soluções desenvolvidas sejam incorporadas de forma estável e funcional à rotina de trabalho do órgão.

Nessa etapa, são definidos os procedimentos para transferência tecnológica e adaptação dos sistemas ao ambiente do IPHAN, considerando suas especificidades técnicas e operacionais. Após essa transferência, os sistemas são implantados em produção e passam a operar com dados reais e usuários finais. Também são previstas ações de suporte e manutenção para garantir a continuidade do funcionamento, a resolução de eventuais problemas e o aprimoramento contínuo da plataforma, assegurando a sustentabilidade da solução no longo prazo.

3.2.1 Criação do modelo de transferência de sistemas informatizados para produção

A criação de um modelo de transferência consiste no empacotamento de todos os serviços já implementado no site de homologação[2].Para isso foram realizados backups das instalações do ambiente de homologação.

Para a transferência de todo o pacote tecnológico elaborado pela equipe de pesquisa, guias de instalação foram encaminhados para a equipe de TI para auxiliar na transferência. Serão relatados as etapas realizadas para elaborar o backup do wordpress e Superset

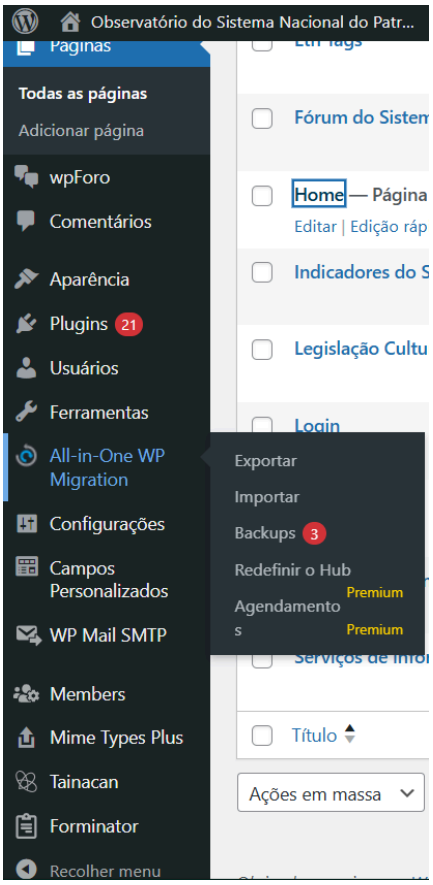
3.2.1.1 Wordpress

Com auxílio do plugin All-In-One WP Migration foi montado o backup dos dois wordpress (Observatório e Mural de Agentes). ass etapa para construção destes backups foram as seguintes:

[2] Disponível em: https://observatorio.iphan.ibict.br/. Acesso em: 30 jul. 2025.

1. No menu lateral, clique em All-in-One WP Migration. Conforme a figura abaixo:

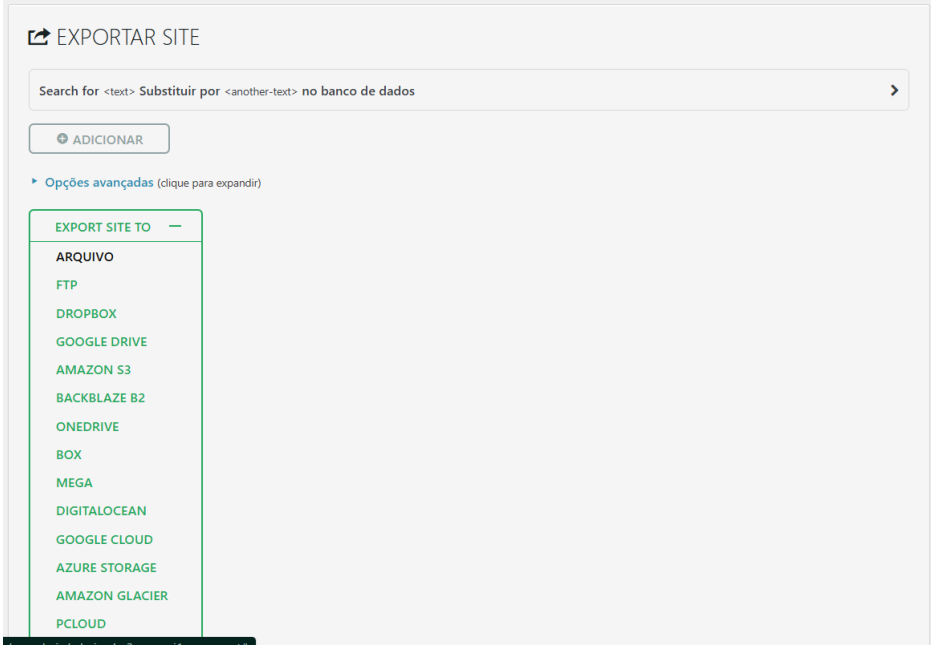
Figura 5 - Acessar All-in-One WP Migration



Fonte: Wordpress Observatório SNPC (2025).

a. Selecione a opção Exportar.
b. Em "Exportar para", clique em Arquivo. Conforme a figura abaixo:

Figura 6 - Exportação do Backup



Fonte: Wordpress Observatório SNPC (2025).

2. O plugin começará a preparar os dados do site (incluindo banco de dados, arquivos, plugins, temas e uploads).

    a. Aguarde o processo de preparação do arquivo de exportação.

    b. Após a finalização, clique em Download quando o botão aparecer.

3. O arquivo terá a extensão .wpress

Uma vez com este arquivo, está contido todas as informações da instalação wordpress que está no ambiente de homologação do IBICT. Com este documento é possível na instalação iphan rodar esse backup e ter a instalação completa pronto para o uso.

3.2.1.2 Apache Superset

O Superset, por ser uma ferramenta de BI (*Business Intelligence)*, tem como foco principal a criação, organização e visualização de dashboards interativos a partir de bases de dados conectadas. Diferente de um CMS como o WordPress, ele não oferece ampla personalização visual de layout, temas ou páginas estáticas. Portanto, o processo de backup e

migração concentra-se essencialmente na preservação das configurações dos dashboards, gráficos, filtros, fontes de dados e permissões vinculadas.

Assim, a transposição do Superset para outro ambiente envolve a cópia do arquivo de configuração superset_config.py e a exportação do banco de dados que armazena todas as configurações internas do Superset, como dashboards, permissões, conexões, usuários e Datasets. Além disso, é necessário exportar o banco de dados do armazém, que contém os dados efetivamente utilizados por esses Datasets. Abaixo estão descritos os comandos realizados para gerar os backups e possibilitar a transferência para uma nova instalação.

1. **Exportar banco de dados do superset por meio da ferramenta pg_dump:**

Quadro 40 - Comando para backup do banco de dados do Superset

```
pg_dump -U user -h <host> -d banco_de_dados > superset_backup.sql
```

Fonte: elaborado pelos autores (2025).

2. **Exportar banco de dados do armazém de dados por meio da ferramenta pg_dump:**

Quadro 41 - Comando para backup do banco de dados do Armazém

```
pg_dump -U user -h <host> -d banco_de_dados > armazem_iphan_backup.sql
```

Fonte: elaborado pelos autores (2025).

Os arquivos, guia tecnológico e backups relacionados ao processo de migração foram disponibilizados no repositório Git do IBICT. Na atividade seguinte será apresentado como é importado os dados para um novo ambiente. Com isso conclui-se a criação de pacotes para a transferência de tecnologias.

3.2.2 implementação dos sistemas informatizados para o ambiente de produção do IPHAN

Para a implementação do observatório era necessário a instalação dos softwares selecionados. Desta forma foi indicado como foi informado na atividade 4.1.1 o passo a passo para instalação dos softwares. Primeiramente do wordpress.

3.2.2.1 WORDPRESS

a) Instalação

Conforme salientado na atividade 4.1.1 com os requisitos de sistemas para instalação do Wordpress. Agora será apresentado a instalação do wordpress e a recuperação do backup do ambiente de homologação. Vale ressaltar que o Observatório do SNPC utiliza duas instalações do *WordPress* para a implantação do site do Observatório e do Mural dos Agentes.

A instalação do WordPress pode ser feita com Docker usando a imagem oficial. O arquivo docker-compose.yml abaixo automatiza a criação dos contêineres do WordPress e do banco de dados MariaDB, configurando todo o ambiente necessário.

Quadro 42 - Arquivo docker-compose.yml

```
version: '3.9'

services:
  wordpress:
    image: wordpress:latest
    restart: unless-stopped
    ports:
      - "8080:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_NAME: xxxxxxx
      WORDPRESS_DB_USER: xxxxxxxx
      WORDPRESS_DB_PASSWORD: xxxxxxxx
    volumes:
      - wordpress_data:/var/www/html

  db:
    image: mariadb:10.6
    restart: unless-stopped
    environment:
      MARIADB_DATABASE: xxxxxxx
      MARIADB_USER: xxxxxxxx
      MARIADB_PASSWORD: xxxxxxx
      MARIADB_ROOT_PASSWORD: xxxxxxx
    volumes:
      - db_data:/var/lib/mysql

volumes:
  wordpress_data:
  db_data:
```

Fonte: Elaborado pelos autores (2025).

Para executar o arquivo *docker-compose.yml*, pode-se utilizar o comando: “docker-compose up -d”, dessa forma, o comando iniciará todos os serviços definidos no arquivo em segundo plano.

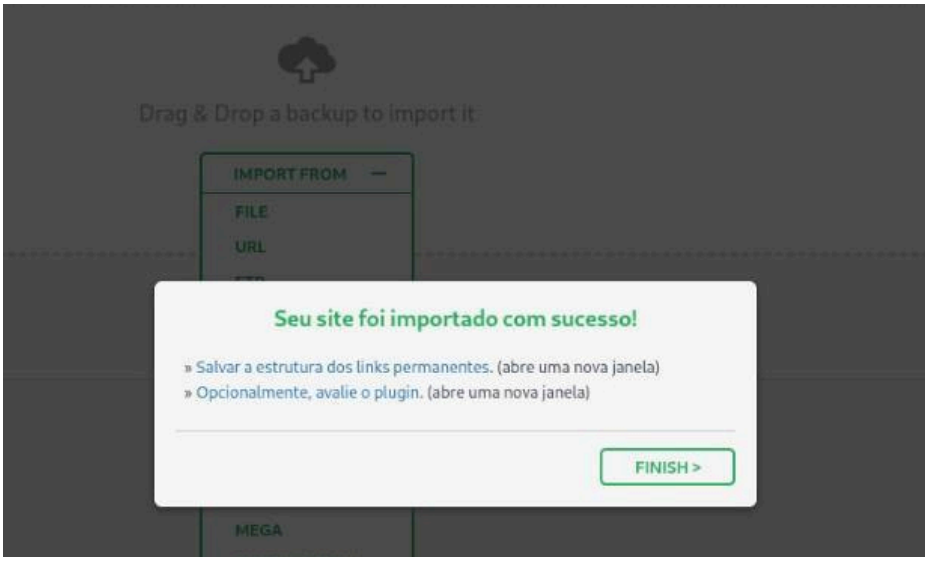
b) Migração

Para a migração iremos restaurar o conteúdo do Observatório do SNPC e o Mural dos Agentes, incluindo bancos de dados, páginas, temas, *plugins* e configurações via *plugin*. É usado o mesmo plugin de backup apresentando na atividade 4.1.1 o *plugin All-in-One WP Migration*[3] *com a funcionalidade de restauração*. A seguir apresentam-se os passos detalhados para realizar a restauração:

1. Acesse o Painel de Administração: Faça o login e vá para o painel de administração.
2. Instale e Ative o *Plugin All-in-One WP Migration* (se ainda não estiver instalado):
    a. No menu do painel, clique em Plugins > Adicionar Plugin;
    b. Pesquise por “*All-in-One WP Migration*” e clique em Instalar Agora;
    c. Após a instalação, clique em Ativar.
3. Acesse a Ferramenta de Importação: No menu do painel, clique em *All-in-One WP Migration* > Importar.
4. Importar o Arquivo .wpress de *Backup*:
    a. Clique no botão Importar de e selecione a opção Arquivo.
    b. Escolha o arquivo .wpress do *backup* e aguarde o *upload*. Esse processo pode levar algum tempo.
5. Confirmar a Importação: Após o *upload*, o *plugin* exibirá uma mensagem avisando que o processo de importação substituirá todos os dados atuais do *site*. Clique em Continuar para confirmar e prosseguir com a restauração do backup.
6. Aguardar a Conclusão da Importação: O *All-in-One WP Migration* restaurará os dados, substituindo o conteúdo e o banco de dados atual pelo backup do arquivo .wpress. Ao final do processo, o plugin exibirá uma mensagem de confirmação de que a importação foi concluída com sucesso conforme a Figura 1.

[3] Disponível em: https://wordpress.org/plugins/all-in-one-wp-migration/. Acesso em 24 jul. 2025.

Figura 7 - Mensagem após a importação bem-sucedida



Fonte: Wordpress Observatório SNPC (2025).

7. Finalize e Verifique o Site: Clique em “Finish” e acesse o site

   Ao final desses passos, a migração será concluída com sucesso.

3.2.2.2 APACHE SUPERSET

a) Instalação

A instalação do *Apache Superset* pode ser realizada com o uso de contêineres *Docker*, por meio de uma imagem customizada voltada para produção, baseada na imagem oficial disponível no Docker Hub, conforme descrito na documentação oficial[4]. A instalação tem como foco a internalização do sistema para o Observatório. A seguir, apresenta-se o passo a passo da instalação, com base na documentação oficial do Superset para Docker[5]:

- A estrutura do projeto foi organizada para facilitar o versionamento, a replicação do ambiente e a automação do processo de implantação. A árvore de diretórios é a seguinte:

Quadro 43 - Árvore de diretórios

```
superset-deploy/
  docker/
    Dockerfile          # Imagem do Superset
    superset_config.py  # Arquivo de configuração do Superset
  docker-compose.yml    # Orquestração dos serviços
```

[4] Disponível em: https://superset.apache.org/docs/installation/installation-methods. Acesso em: 10 jun. 2025.
[5] Disponível em: https://superset.apache.org/docs/installation/docker-compose/. Acesso em: 10 jun. 2025.

```
backup/
  superset_backup.sql  # Dump do banco de dados do Superset
  armazem_obs_backup.sql  # Dump do banco de dados do armazém
```

Fonte: Elaborado pelos autores (2025).

- O arquivo Dockerfile foi baseado na imagem oficial apache/superset: versão 4.1.2

Quadro 44 - Exemplo de arquivo Dockerfile

```
FROM apache/superset:4.1.2

USER root

# Instala drivers e dependências
RUN pip install --no-cache-dir \
  psycopg2-binary \
  apache-superset[celery,redis] \
  && apt-get update \
  && apt-get install -y chromium-driver

# Copia o arquivo de configuração customizado
COPY docker/superset_config.py /app/pythonpath/

USER superset
```

Fonte: Elaborado pelos autores (2025).

- Abaixo apresenta-se um exemplo de arquivo docker-compose.yml utilizado para instalar e configurar o Superset em um ambiente Docker. Esse arquivo automatiza a instalação do Superset, seus serviços auxiliares e a restauração do arquivo de configuração superset_config.py como volume dentro do contêiner.

Quadro 45 - Exemplo de arquivo docker-compose.yml

```
version: "3.3"

services:

 superset:
  build:
   context: .
   dockerfile: docker/Dockerfile
  container_name: superset_app
  restart: always
  environment:
   SUPERSET_CONFIG_PATH: /app/pythonpath/superset_config.py
  ports:
   - "8088:8088"
```

```
    volumes:
      - ./docker/superset_config.py:/app/pythonpath/superset_config.py
    depends_on:
      - redis

  redis:
    image: redis:7
    container_name: superset_redis
    restart: always

  celery_worker:
    build:
      context: .
      dockerfile: docker/Dockerfile
    container_name: superset_celery_worker
    command: celery --app=superset.tasks.celery_app:app worker --pool=prefork
--loglevel=INFO
    restart: always
    depends_on:
      - superset
      - redis


  celery_beat:
    build:
      context: .
      dockerfile: docker/Dockerfile
    container_name: superset_celery_beat
    command: celery --app=superset.tasks.celery_app:app beat --loglevel=INFO
    restart: always
    depends_on:
      - superset
      - redis
```

Fonte: Elaborado pelos autores (2025).

- Considerando o ambiente do Iphan, onde as políticas de segurança preservam que os bancos de dados sejam mantidos em um servidor separado, fora da infraestrutura gerenciada pelo Docker, a abordagem para recuperar e conectar o banco de dados ao Apache Superset pode ser implementada da seguinte forma:
    - O backup foi realizado utilizando o utilitário pg_dump, o arquivo está localizado em backup/superset_backup.sql. Para realizar a restauração do banco, basta utilizar o comando psql, apontando para o banco de destino previamente criado:

Quadro 46 - Comando para restaurar o banco de dados do Superset

```
psql -U <usuario> -h <host> -d <nome_do_banco> < superset_backup.sql.sql
```

Fonte: Elaborado pelos autores (2025).

- Para que o Superset se conecte ao banco externo, é necessário informar explicitamente a URL de conexão ao banco por meio da variável SQLALCHEMY_DATABASE_URI no arquivo superset_config.py, localizado no diretório docker/. Por exemplo:

Quadro 47 - Exemplo de conexão do Superset com o banco de dados externo

```
SQLALCHEMY_DATABASE_URI =
'postgresql+psycopg2://superset_user:superset_pass@X.X.X.X:5432/superset_db'
```

Fonte: Elaborado pelos autores (2025).

- Importante que o arquivo pg_hba.conf do servidor do PostgreSQL aceito conexão do superset, exemplo local:

Quadro 48 - Configuração do arquivo pg_hba.conf para conexão

```
# Permitir acesso do container Superset sem SSL
host    superset        superset        172.16.0.0/12        md5
```

Fonte: Elaborado pelos autores (2025).

Nota: É essencial que o servidor onde o Apache Superset está hospedado tenha acesso de rede aos servidores de banco de dados, garantindo que as conexões tanto com o banco de metadados do Superset quanto com o armazém de dados sejam possíveis durante a execução dos dashboards e consultas.

- Por fim, a instalação é concluída com a inicialização dos serviços por meio do arquivo docker-compose.yml, utilizando o comando: “docker-compose up -d --build”. O acesso a interface web será em: http://localhost:8088.

b) Migração

O armazém de dados utilizado no Apache Superset para a criação dos datasets foi desenvolvido em PostgreSQL. Esse banco não é o banco de dados do Superset, mas sim o

armazém de dados que alimenta os dashboards. Para criar o banco de dados com a extensão PostGIS, utilize os comandos abaixo:

Quadro 49 - Comandos para instalar o PostGIS, criar o banco e criar a extensão PostGIS

```
#Instala o PostGIS (no sistema operacional)
sudo apt install postgis

#Cria o banco de dados
CREATE DATABASE armazem_iphan;

#Conecta-se ao banco
\c armazem_iphan

#Cria a extensão PostGIS
CREATE EXTENSION postgis;
```

Fonte: Elaborado pelos autores (2025).

Ele deve ser restaurado e estar acessível para que o Superset consiga consultar os dados definidos nos datasets existentes. Portanto, o backup foi realizado utilizando o utilitário pg_dump, o arquivo está localizado em backup/armazem_obs_backup.sql. Para realizar a restauração desse banco, basta utilizar o comando psql, apontando para o banco de destino previamente criado:

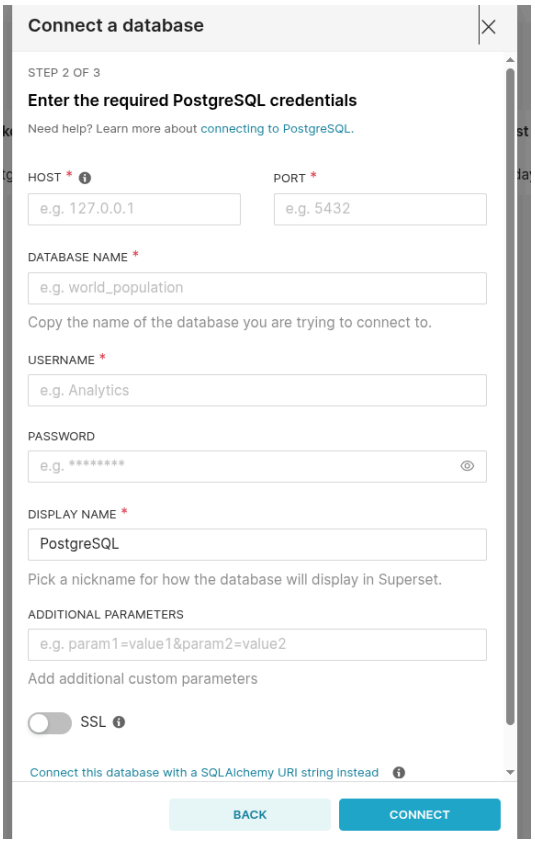
Quadro 50 - Comando para restaurar o banco de dados do armazém

```
psql -U <usuario> -h <host> -d <armazem_iphan> < armazem_obs_backup.sql
```

Fonte: Elaborado pelos autores (2025).

Para manter o ambiente configurado funcional após a migração, será necessário reconectar o armazém de dados ao Superset. A Figura 9 ilustra a conexão com o banco de dados no Superset.

Figura 8 - Interface Superset para conexão com bancos de dados



Fonte: Superset Observatório do SNPC (2025).

Com o backup do banco de dados do Superset restaurado. Siga os passos abaixo na interface do Superset: Acesse o menu Data > Databases

- Localize a conexão do armazém;
- Clique em “Edit”;
- Atualize o host, nome do usuário, senha, no do banco de dados.
- E por fim clique em “Finish”.

3.2.3 Apoio a manutenção do observatório

Para viabilizar a transferência completa do pacote tecnológico desenvolvido pela equipe de pesquisa, foi elaborado o GUIA TECNOLÓGICO DO OBSERVATÓRIO DO SNPC: O passo a passo para a instalação, configuração e manutenção, com as informações detalhadas para a transferência. Além disso, disponibilizados especialistas para acompanhar diretamente o processo, prestando suporte técnico e esclarecendo eventuais dúvidas.

Reuniões periódicas foram realizadas com a equipe de tecnologia do IPHAN, com o objetivo de garantir uma transferência eficiente, segura e bem orientada. Todo o processo de instalação dos softwares e migração dos dados foi conduzido de forma articulada entre a equipe técnica do projeto e os profissionais do IPHAN, por meio de interações contínuas via e-mail e encontros remotos, assegurando o alinhamento necessário para a implementação das soluções tecnológicas propostas.

4 CONSIDERAÇÕES FINAIS

A Meta 4, conforme o plano de trabalho, representa a entrega do pacote tecnológico desenvolvido nas metas anteriores viabilizando a sua transferência para o ambiente de produção no âmbito do IPHAN. O projeto foi estruturado em cinco metas, a meta 1, voltada ao levantamento das fontes informacionais e à formação da equipe técnica especializada; Meta 2, dedicada à concepção e modelagem do Observatório enquanto sistema de informação; Meta 3, responsável pela implementação do modelo em ambiente laboratorial e sua validação para a equipe do IPHAN; Meta 4, que trata da transferência e implantação do sistema no ambiente institucional do IPHAN; e, por fim, Meta 5, que se destina à produção e disseminação dos materiais de apoio e estratégias de divulgação do Observatório.

No entanto, é importante destacar que, devido à interrupção antecipada do projeto, as Metas 3 e 4 precisaram ser executadas de forma simultânea e em um prazo significativamente inferior ao originalmente previsto. Esta situação comprometeu o tempo necessário para a finalização de certas etapas e atividades, principalmente no que tange ao refinamento e validação técnica das integrações e funcionalidades. Como consequência, algumas atividades esperadas não puderam ser finalizadas, o que impacta diretamente na completude e robustez do resultado entregue.

Apesar dessas limitações, foram alcançados avanços expressivos: ambientes de homologação e produção foram estruturados com base em soluções livres e interoperáveis (WordPress, Apache Superset, PostgreSQL), rotinas automatizadas de integração de dados foram desenvolvidas com qualidade técnica e rigor, e o processo de transferência contou com o acompanhamento direto da equipe de pesquisa junto ao Iphan, por meio de reuniões periódicas e materiais de orientação.

A estrutura tecnológica entregue permite que o Observatório do SNPC avance para a etapa de institucionalização, sendo capaz de apoiar políticas públicas com transparência, monitoramento contínuo e análise qualificada de dados sobre o patrimônio cultural brasileiro. Ainda assim, recomenda-se que ações complementares futuras busquem retomar as atividades que foram afetadas pela descontinuidade do projeto, garantindo a consolidação integral da proposta original.

REFERÊNCIAS

GUSMÃO, M. R. Observatório apoia a adoção de tecnologias de gestão. **Informe**, v. 26, n. 175, 2006.

WORDPRESS. **Documentation**. Disponível em: https://wordpress.org/documentation/. Acesso em: 25 jun. 2025.

SUPERSET. **Introduction**. Disponível em: https://superset.apache.org/docs/intro. Acesso em: 20 jun. 2025.