Universidade Federal de São Carlos– UFSCar
Centro de Ciências Exatas e de Tecnologia– CCET
Departamento de Computação– DC
Programa de Pós-Graduação em Ciência da Computação– PPGCC

Banco de Dados Geográficos Relatório Final

Non-Conventional Databases

Docente: Drª. Marilde Terezinha Prado Santos

Discentes: Guilherme Vilar Balduino - 743546
Paulo Henrique Corrêa de Morais - 816091

1. Introdução

Este projeto é uma proposta apresentada no início da disciplina de Banco de Dados Não Relacionais, onde será trabalhado um Banco de Dados Geográfico, utilizando os dados de um levantamento georreferenciado com imagens georreferenciadas de uma aplicação real.

Os dados obtidos são de um experimento controlado de uma plantação de algodão gerenciada pelo Instituto Matogrossense de Algodão (IMA) em parceria com o Instituto Federal de Mato Grosso (IFMT). As imagens são obtidas por meio de sensores (câmeras termais, multiespectrais e RGB) embarcadas em um Veículo Aéreo Não Tripulado (VANT) e um trabalho manual de coleta de pontos de marcos de controle.

Este projeto foi estruturado para a regra de negócio conforme as atividades desenvolvidas na Pesquisa do Algodão. Assim será apresentado uma breve revisão sobre Banco de Dados Geográfico, depois disso as etapas de construção do projeto e implementação e por fim os testes realizados.

2. Revisão Bibliográfica

2.1. Sistema de Informação Geográfico

Um Sistema de Informação Geográfico (SIG), contém algumas complexibilidades em suas aplicações o que torna um desafio estrutura e armazenar os dados de forma a possibilitar a realização de operações de consulta e análise em um Sistema de Gerenciamento de Banco de Dados (SGBD) (LISBOA FILHO; IOCHPE, 2001).

Queiroz e Ferreira (2005, p. 4) apontam algumas características de SIGs que são:

- Inserir e integrar, numa única base de dados, informações espaciais provenientes de meio físico-biótico, de dados censitários, de cadastros urbano e rural, e outras fontes de dados como imagens de satélite, e GPS.
- Oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, bem como para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

Conforme o manual de Geociências do IBGE (2019), um dado espacial descreve um fenômeno associado a alguma dimensão no espaço. A dimensão está associada à sua localização na superfície terrestre, em determinado instante ou período de tempo.

Nos últimos anos o SIG, vem aprimorando a capacidade dos diversos programas em lidar com imagens e dados alfanuméricos simultaneamente, permitindo a visualização espacial das análises realizadas. Essa característica ampliou a capacidade de percepção dos resultados pelos usuários (SILVA; MELO; BRONDINO, 2021). A ilustração da **Figura 1** mostra a dinâmica de um SIG.

Figura 1 - Componentes de Sistema de Informação Geográfica

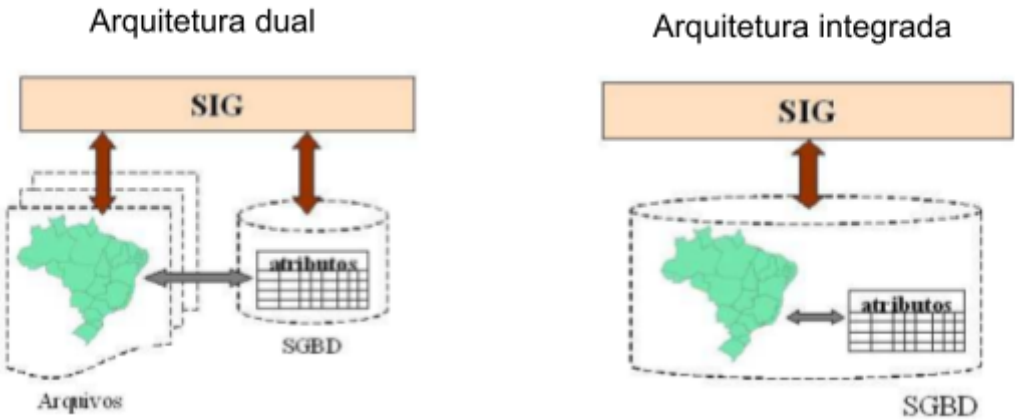
Fonte: Adaptado de AGROPOS. O que é geoprocessamento? Disponível em: https://agropos.com.br/o-que-e-geoprocessamento/. Acesso em: 03 abr. 2023.

Em uma reflexão sobre os avanços de SIG, LÜ, Guonian et al (2019) apontam alguns fatores que contribuíram para o impulso no desenvolvimento de SIG nos últimos 10 anos, dentre os fatores citam a capacidade de criação de multimídias, realidade virtual, gráficos e imagens gerados por computador, capacidade de armazenamento, comunicação em fibra ótica, o que impulsionou o desenvolvimento de GIS em ambientes distribuídos e assim popularizou o GIS para uso geral.

2.2. Banco de Dados Geográficos

Segundo Casanova et al (2005), os Bancos de Dados Geográficos (BDG) são, para os SIGs, o ponto central da arquitetura e o componente responsável pelo armazenamento dos dados. São apresentadas duas formas de integração entre os SIGs e os SGBDs, que são arquitetura dual e arquitetura integrada, conforme mostra a **Figura 2**.

Figura 2 - Formas de integração entre os SIGs e os SGBDs



Fonte: Adaptado de Casanova et al (2005)

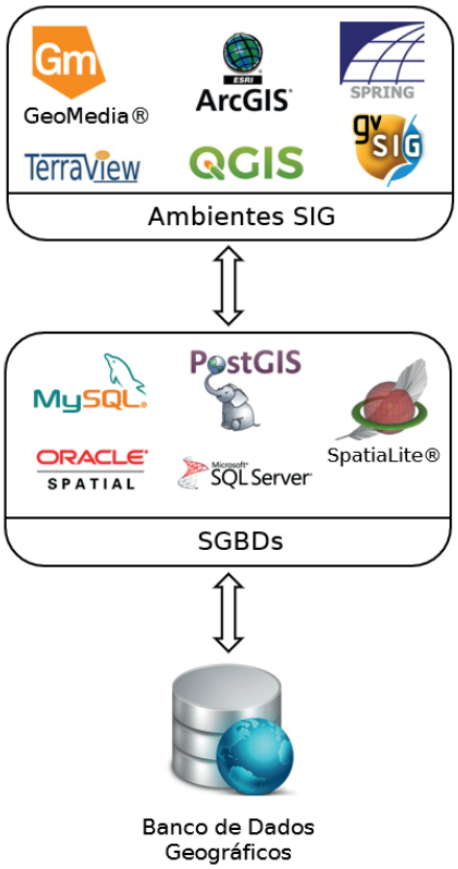
Na arquitetura dual os componentes espaciais são armazenados separadamente, a componente convencional, ou alfanumérica, é armazenada em um SGBD relacional e a componente espacial é armazenada em arquivos com formato proprietário. Alguns problemas no uso dessa arquitetura são: dificuldade na manipulação das componentes espaciais; dificuldade em manter a integridade das componentes espaciais, entre outros.

Na arquitetura integrada todos os dados são armazenados em um SGBD, sendo assim são utilizados os recursos de um SGBD para controle e manipulação de objetos espaciais, controle de integridade e linguagens próprias de consulta. Na arquitetura integrada pode ser baseada em: campos longos, extensões espaciais e combinadas.

De acordo com Casanova et al. (2005, p. 56), "ao codificar dados espaciais em BLOBs, esta arquitetura torna a sua semântica opaca para o SGBD. Ou seja, passa a ser responsabilidade do SIG implementar os operadores espaciais...".

O banco de dados geográfico pode ser acessado por multiusuário, a partir de diferentes ambientes SIGs ou diferentes aplicações. Os SGBDs controlam este acesso, conforme ilustra a **Figura 3**.

Figura 3 - Formas de acesso ao banco de dados geográficos.



Fonte: IBGE, Diretoria de Geociências, Coordenação de Cartográfica.

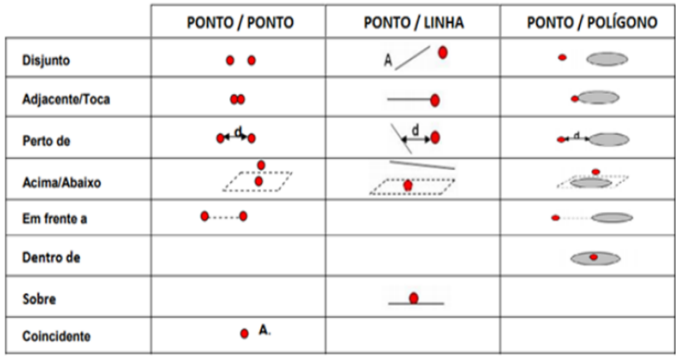
O manual de Geociências do IBGE (2019) apresenta os padrões de relacionamento espaciais conforme a *International Organization for Standardization* - ISO/ *Open Geospatial Consortium* – OGC, que são nove métodos:

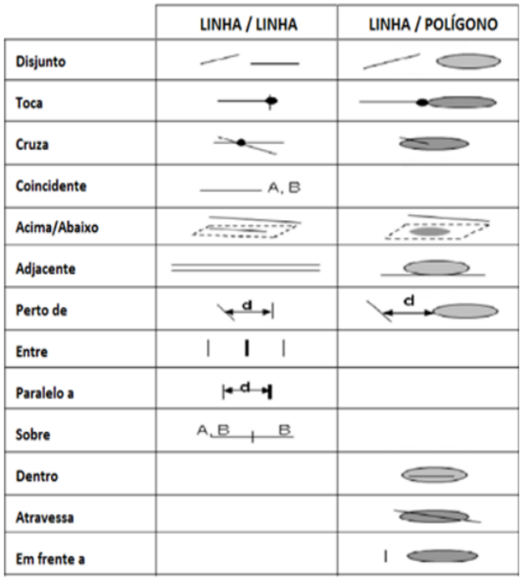
1. *equal* (iguais);
2. *disjoint* (disjuntos);

3. *intersects* (interceptam);
4. *touches* (tocam);
5. *crosses* (cruzam);
6. *within* (dentro de);
7. *contains* (contém);
8. *overlaps* (sobrepõem);
9. *relate* (relacionam-se).

A **Figura 4** apresenta uma exemplificação dos relacionamentos espaciais conforme o seu tipo (ponto, linha e área).

Figura 4 - Relacionamento espacial entre as geometrias





Fonte: Adaptado de IBGE. Coordenação de Cartografia. Acesso e uso de dados geoespaciais (2019).

3. Projeto do Banco de Dados

Para este projeto inicialmente foram analisadas as informações de caso de uso referente ao levantamento georreferenciado de imagens aéreas. Os dados são provenientes de um trabalho de pesquisa monitoramento do algodão *in loco* utilizando Veículos Aéreos Não Tripulados (VANTs) com câmeras termais, RGB e multiespectrais embarcadas.

O monitoramento consiste em: Definição do plano de voo, coleta de imagens em campo, demarcação de pontos de controle com GPS estação-base, pré-análise visual das imagens após o voo realizado, pré-processamento das imagens (ortomosaico), análise das parcelas por tipo de cultivo e tipo de condução de testes feitos no cultivo (esta análise é em conjunto com o experimentador do IMA). O monitoramento é realizado em vários estágios de crescimento do cultivo, onde se é possível acompanhar o histórico do desenvolvimento da planta.

3.1. Agentes Envolvidos

Nesta etapa será discriminado a delegação de tarefas de cada usuário envolvido no processo de tratamento dos dados.

Coordenadores: conjunto de usuários que têm total acesso às funcionalidades presentes no banco de dados de forma irrestrita, sendo o responsável em validar e analisar toda sua extensão juntamente com a função de alterar os integrantes dos demais grupos e seus níveis de acesso e manipulação.

Pesquisador: conjunto de usuários responsáveis por inserir e atualizar devidamente os dados no banco, sendo eles os principais que desempenham essa função. Tendo em vista o tamanho número de dados que devem ser extraídos, processados, e manipulados, eles delegam tarefas aos bolsistas sobre sua supervisão.

Bolsistas: estes usuários por sua vez somente podem extrair dados para processamento e inserir novos dados, mas os dados obtidos por eles somente serão disponibilizados ao público depois de serem avaliados por algum pesquisador ou coordenador.

Público: um usuário qualquer que não é participante do projeto, tendo somente permissão de visualização dos dados presentes e que foram validados.

3.2. Conjunto de Requisitos

Os requisitos para o projeto serão elencados a seguir:

- Deve-se ser possível extrair imagens de diferentes câmeras que tenham a mesma geolocalização armazenadas, de forma que possamos criar comparativos entre elas. Na mesma linha, a busca por imagens semelhantes deve ser bem otimizada visto que terá um número exorbitante de dados armazenados e devem ser encontrados de maneira rápida e diversos fins de processamento.
- Ao inserir um novo dado/imagem o deve ser inserido uma breve descrição do que está presente na imagem para que não seja necessário trazer a imagem para uma análise momentânea, sendo que essa descrição poderá ser feita automaticamente pelo sistema ou manualmente pelo usuário. Tal descrição também será utilizada para a busca de imagens, como por exemplo falhas em áreas com vegetação.
- As imagens devem ser armazenadas de forma que possam ser agrupadas conforme os padrões presentes nelas como falhas em áreas com vegetação, coloração da folhagem ou temperatura. Junto a esse requisito, elas devem ser armazenadas de modo que ao extrair elas o sistema possa criar um ortomosaico fidedigno de toda a área registrada de forma automatizada, assim como é feito no Agisoft hoje manualmente.
- Os dados somente podem ser disponibilizados para o público após eles serem avaliados por um pesquisador ou coordenador, portanto é necessário que seja possível ser feita essa distinção sem que seja perdido quem inseriu e analogamente se quem inseriu os dados já é um pesquisador logo o dado será considerado como avaliado.

3.3. Modelo Entidade Relacionamento

O diagrama do modelo Entidade-Relacionamento (ER) foi desenvolvido com o auxílio do software open-source TerraER conforme apresentado no Anexo 1. No diagrama são apresentados as entidades e seus relacionamentos para a criação do

banco de dados. A visualização gráfica se torna intuitiva e ágil para a arquitetar o SGBD e seus relacionamentos. A seguir serão comentados a entidades identificadas no projeto que aparecem em duas figuras centrais: os usuários e as imagens.

Entidade “usuário”: esta entidade tem o papel de identificar os agentes e seus papéis no SGBD. Os agentes são membros da organização interna ou membros externos à organização, ele também pode ser do tipo “Pesquisador” ou “Bolsista”. O usuário é responsável por inserir a informações

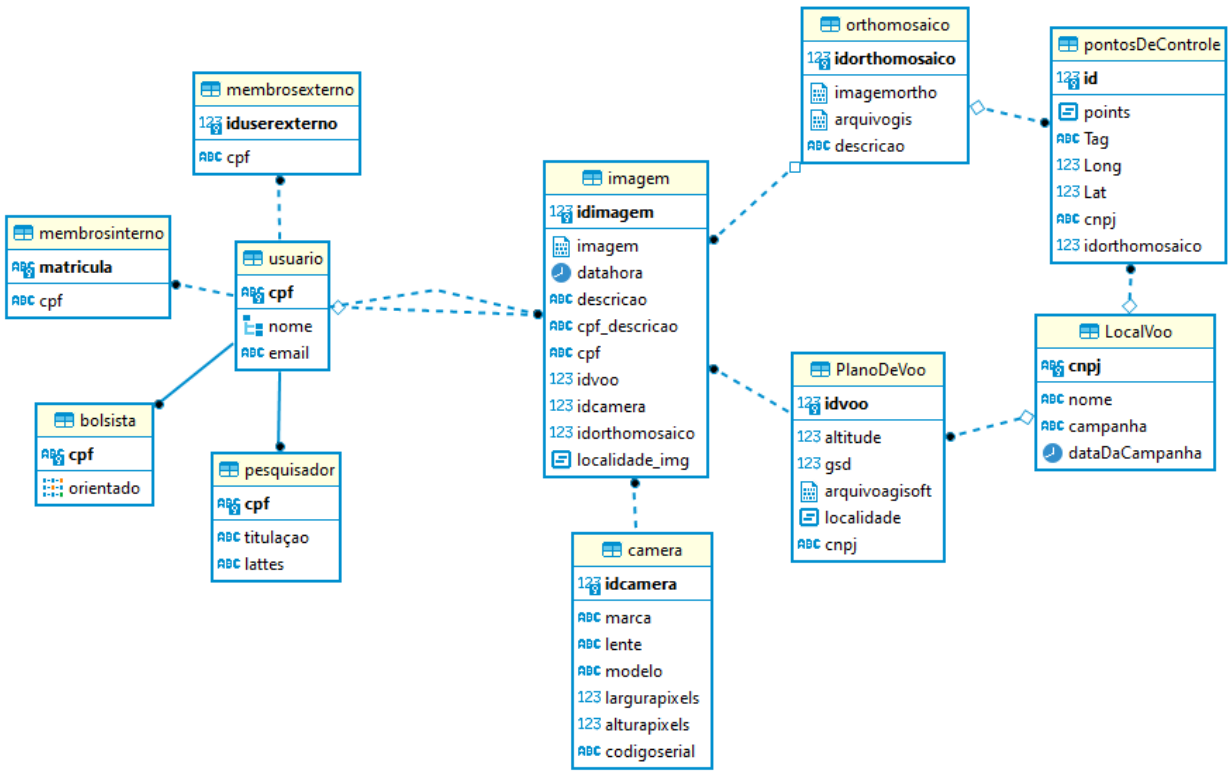
Entidade “imagem”: esta entidade é o *core* do projeto, pois todos os relacionamentos estão contidos nela. A entidade “imagem” é gerada por um sensor rotulado como a entidade “câmera”, e ela só pode ser inserida no SGBD por um usuário. Junto a entidade “imagem” são associadas as entidades “planoDeVoo” e “localDeVoo”.

A entidade “ortomosaico” é o resultado da junção das imagens junto com o georreferenciamento dos pontos de controle, no qual foi criado uma entidade para tal atividade.

3.4. Esquema Físico do SGBD

A Figura 5 mostra o esquema relacional gerado pelo DBeaver, onde é possível verificar as tabelas do SGBD e seus atributos e o tipo deles. Também mostra como as tabelas estão relacionadas e estruturadas no SGBD. Este esquema foi gerado pelo software DBeaver Community (2023).

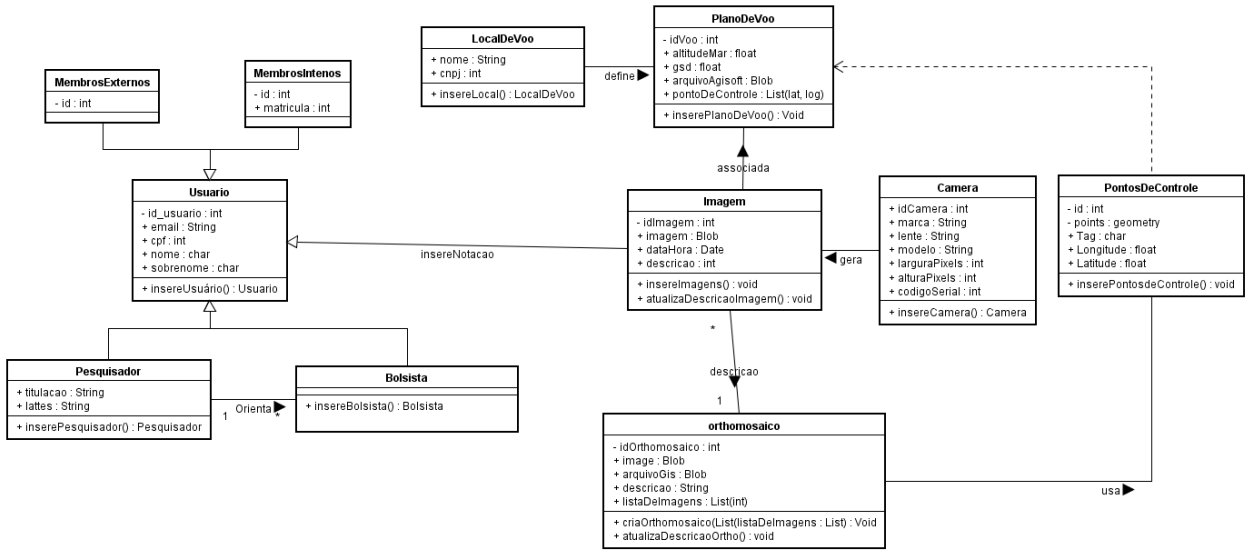
Figura 5 - Esquema Gerado pelo DBevear



Fonte: Autores

Foi elaborado um esquema físico do tipo diagrama de classes UML, mostrado na Figura 6. Diferente do esquema anterior da Figura 5, a Figura 6 apresenta os procedimentos disponíveis para comunicação com o SGBD, representados nos campos de métodos do diagrama UML. Cabe ressaltar que estão faltando as funções de buscas no diagrama contidas no script desenvolvido conforme Anexo 2.

Figura 6 - Esquema Físico Representado no diagrama UML



Fonte: Autores

4. Implementação do banco de dados

4.1. Descrição da implementação

Para a implementação do projeto foi escolhido o PostgreSQL para SGBD. O PostgreSQL vem com muitos recursos destinados a ajudar os desenvolvedores a criar aplicativos, os administradores a proteger a integridade dos dados e a criar ambientes tolerantes a falhas, além de ajudá-lo a gerenciar seus dados, independentemente do tamanho do conjunto de dados (POSTGRESQL, 2023).

A escolha do PostgreSQL para este projeto foi devido a sua extensão espacial PostGis adicionar suporte para objetos geográficos, permitindo que consultas de localização sejam executadas em SQL. O PostGIS adiciona tipos extras (geometria, geografia, raster e outros) ao PostgreSQL. Ele também adiciona funções, operadores e aprimoramentos de índice que se aplicam a esses tipos espaciais (POSTGIS, 2023).

O script da implementação pode ser consultado no Anexo 2.

4.2. Consultas

Confere se determinado ponto está contido no polígono analisado. Para realizar esta busca, foi utilizado a função *ST_Contains* da extensão PostGIS para o

PostgresSQL. Conforme a documentação da extensão PostGIS a função *ST_Contains* retorna TRUE se a geometria B estiver completamente dentro da geometria A. A contém B se e somente se nenhum ponto de B estiver no exterior de A e pelo menos um ponto do interior de B estiver no interior de A.

Com essa verificação, se verdadeiro, é possível o retorno de todos os objetos contidos no teste. Foi criado uma função denominada “confere_ponto_cidade” como argumentos devem ser passados o “id” do ponto desejado e o “id” do polígono desejado.

5. Testes do banco de dados

5.1. Interface python

Foi criada uma interface Python por linha de comando simples para manipulação do banco de dados e transformações de um tipo de dado em outra antes de passar para o usuário. Código em anexo.

5.2. Consulta utilizando Índices diferentes

Foram realizadas dez amostragens para diversos índices. Para a realização deste teste foi necessário a criação de um Banco de dados de teste, pois os dados originais do projeto inicial não eram suficientes para gerar tal diferença.

O DB de teste recebeu uma carga de dados com os municípios de Mato Grosso (MT), essa base de dados foi adquirida pelo site do IBGE dados de 2021. Também foi necessário a criação de pontos geográficos aleatórios na faixa territorial do estado de MT, no total foram criados 10 milhões de pontos geográficos.

A inserção e criação das tabelas foi por meio da interface *Geographic Information System* (GIS), conhecida popularmente como QGis. Como o objetivo era os testes, não foi gerado um script SQL. A Tabela 1 apresenta os resultados dos testes utilizando a função “*ST_Contains(geometry geomA, geometry geomB)*” da extensão PostGis os valores obtidos foram:

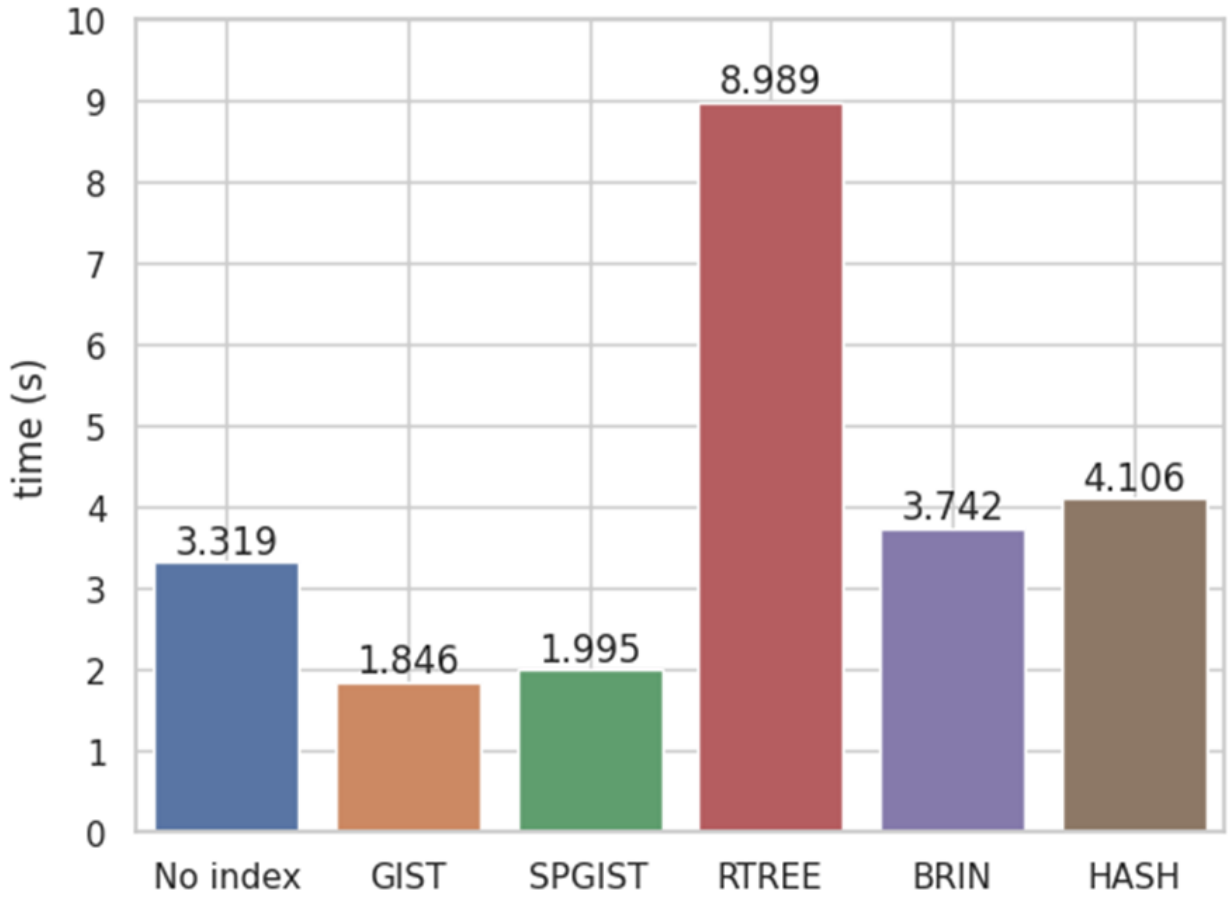
Tabela 1 - Tabela de tempo (s) de busca de acordo com os índices utilizados

Index	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Mean
No index	3.30	3.34	3.31	3.35	3.31	3.32	3.32	3.30	3.31	3.33	**3.32**
GIST	1.87	1.79	1.79	1.99	1.77	1.79	1.79	1.83	1.99	1.85	**1.85**
SPGIST	1.91	1.97	1.94	1.83	1.84	1.99	2.76	1.95	1.94	1.82	**1.99**
RTREE	9.45	9.69	9.44	9.31	8.61	8.75	8.09	8.79	8.87	8.89	**8.99**
BRIN	3.82	3.69	3.83	3.72	3.67	3.66	3.69	3.93	3.73	3.68	**3.74**
HASH	5.94	3.67	3.76	3.87	3.98	4.12	4.83	3.69	3.67	3.53	**4.11**

Fonte: Autores

O diagrama de barra da Figura 7 ajuda na visualização da diferença entre os índices. É importante destacar que quando foi utilizado o índice “RTree” o Postgresql retorna uma mensagem de aviso (NOTICE: substituting access method "gist" for obsolete method "rtree").

Figura 7 - Gráfico comparativo dos índices



Fonte: Autores

Em um segundo teste realizado foi utilizado a função "*ST_Area(geometry g1)*" que retorna a área de uma geometria poligonal. Os valores obtidos do tempo resposta estão na Tabela 2.

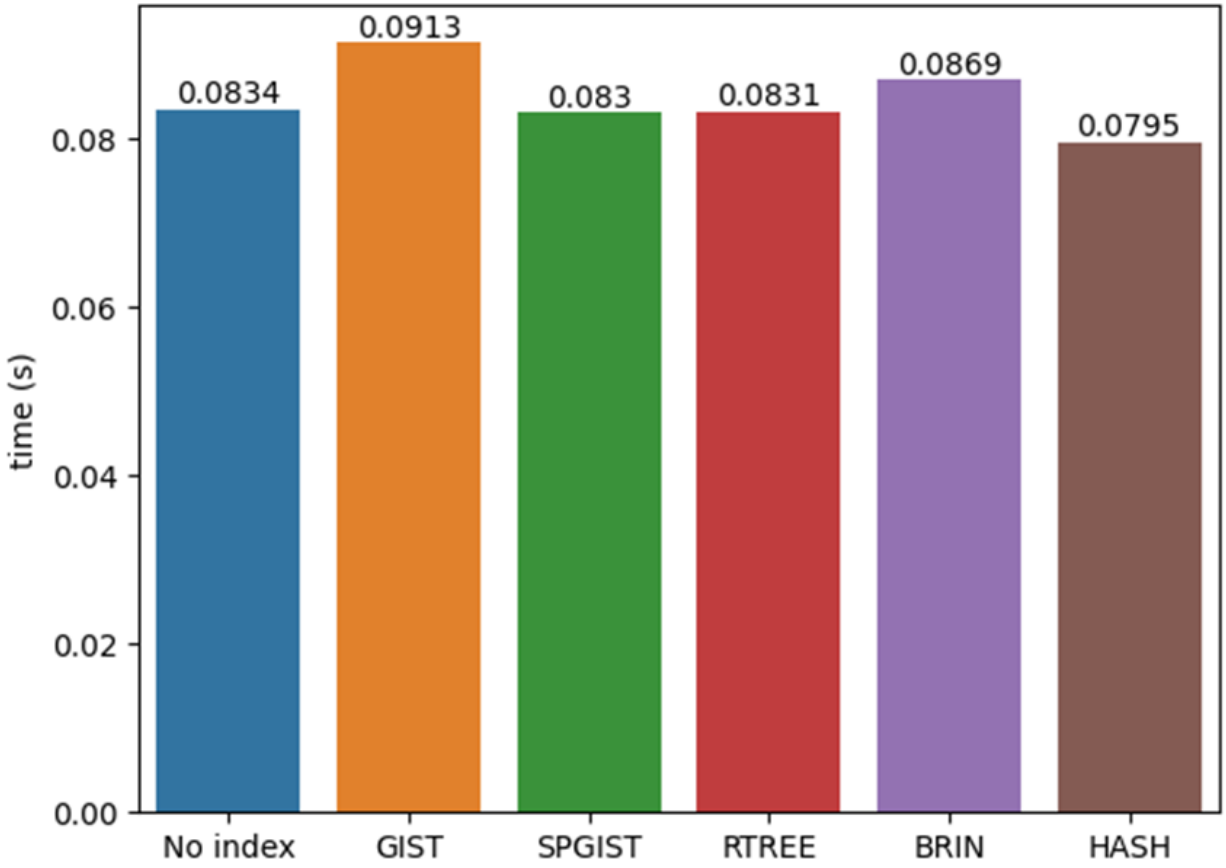
Tabela 2 - Tabela de tempo (s) de resposta em função do índice utilizado.

Index	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	Mean
No index	0.092	0.068	0.079	0.097	0.088	0.119	0.055	0.081	0.054	0.101	**0.0834**
GIST	0.049	0.095	0.093	0.099	0.095	0.108	0.109	0.080	0.105	0.080	**0.0913**
SPGIST	0.051	0.080	0.080	0.046	0.093	0.100	0.098	0.061	0.100	0.121	**0.0830**
RTREE	0.050	0.103	0.051	0.073	0.081	0.098	0.131	0.068	0.101	0.075	**0.0831**
BRIN	0.043	0.107	0.088	0.089	0.100	0.087	0.072	0.082	0.116	0.085	**0.0869**
HASH	0.050	0.097	0.112	0.058	0.111	0.056	0.079	0.092	0.085	0.055	**0.0795**

Fonte: Autores

Na figura é possível verificar a proximidade dos valores entre os índices. Uma hipótese dessa proximidade é a baixa quantidade de dados, no total 144 polígonos (municípios de Mato Grosso – IBGE 2021), sendo requisitado o valor da área de todos em uma única execução da função.

Figura 8 - Gráfico comparativo dos índices utilizando a função ST_Area



Fonte: Autores.

6. Considerações Finais

Foi visto na documentação do PostGis que o operador *within* utiliza um próprio índice ao contrário do operador *contains* que trabalha com os índices criados no banco, por conta disso as análises foram feitas com base no operador *contains* para encontrar o melhor índice.

O projeto encontra-se disponível no GitHub.

7. Propostas de Melhorias

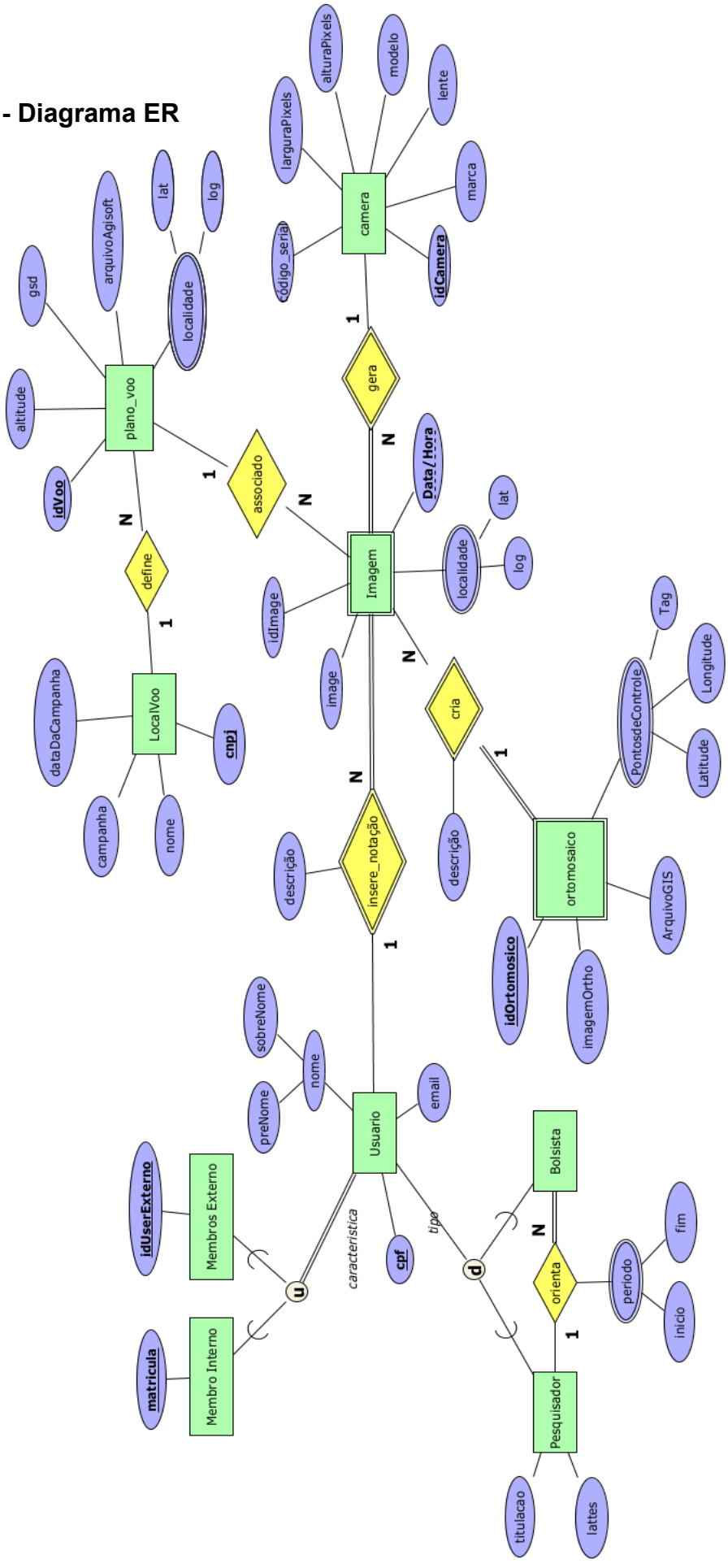
Aqui serão apresentados alguns pontos que precisam ser melhorados para trabalhos futuros e otimização da implementação, as observações foram:

- O usuário do tipo “Membro Externo” à Organização não deve ser autorizado a inserir imagens no banco, a não ser que seja um pesquisador.
- Revisar e atualizar todos os diagramas do projeto
- Aprimorar o script python para que ele trate o ponto de controle de modo que dada uma função que receba um ponto geográfico e retorne a latitude e longitude dessa forma retirar essa informação do banco pois está redundante.
- Desenvolver pesquisa de índices para investigar qual índice será o mais adequado a cada função/consulta distinta.

Referências

- CASANOVA, Marco Antonio et al. Bancos de Dados Geográficos. Maio, 2005. Curitiba Editora Mundogeo. 504p.
- DBeaver Community - About. Disponível em: https://dbeaver.io/about/. Acesso em: 03 abr. 2023.
- FILHO, Jugurta Lisboa; IOCHPE, Cirano. Modelagem de Bancos de Dados Geográficos. Porto Alegre, outubro de 2001. Universidade Federal de Viçosa, Departamento de Informática; Universidade Federal do Rio Grande do Sul, Instituto de Informática.
- IBGE. Coordenação de Cartografia. Acesso e uso de dados geoespaciais. Rio de Janeiro: IBGE, 2019. 143 p. (Manuais técnicos em geociências, ISSN 0103-9598; n.14). Inclui bibliografia. ISBN 978-85-240-4508-0.
- LÜ, Guonian et al. Reflections and speculations on the progress in Geographic Information Systems (GIS): a geographic perspective. International journal of geographical information science, v. 33, n. 2, p. 346-367, 2019.
- POSTGIS. PostGIS - Spatial and Geographic Objects for PostgreSQL. Disponível em: https://postgis.net/. Acesso em: 03 abr. 2023.
- POSTGIS. ST_Contains. [S. l.], 2021. Disponível em: https://postgis.net/docs/ST_Contains.html. Acesso em: 06 abr. 2023.
- POSTGRESQL. PostgreSQL: The world's most advanced open source database. Disponível em: https://www.postgresql.org/. Acesso em: 03 abr. 2023.
- QUEIROZ, Gilberto Ribeiro de; FERREIRA, Karine Reis. Banco de Dados Geográficos. São José dos, 2005.
- SILVA, Antônio Nélson Rodrigues da; MELO, Jônatas José de Oliveira; BRONDINO, Nair Cristina Margarido. Uma introdução ao planejamento de transportes com sistemas de informação geográfica. 2021.

Anexo 1 - Diagrama ER



Anexo 2 - Script de Implementação do Projeto em SQL

```
CREATE EXTENSION postgis;

drop function IF EXISTS encontra_cidade;
drop function IF EXISTS encontra_ponto_por_cidade;
drop function IF EXISTS confere_ponto_cidade;

drop procedure IF EXISTS atualiza_descricao_Imagem;
drop procedure IF EXISTS insere_bolsista;
drop procedure IF EXISTS insere_camera;
drop procedure IF EXISTS insere_planodevoo;
drop procedure IF EXISTS insere_local;
drop procedure IF EXISTS insere_pesquisador;
drop procedure IF EXISTS insere_usuario;
drop procedure IF EXISTS MembrosExterno;
drop procedure IF EXISTS MembrosInterno;
drop procedure IF EXISTS insere_Imagem;
drop procedure IF EXISTS insere_orthomosaico;

drop TABLE IF EXISTS Imagem;
drop TABLE IF EXISTS MembrosInterno;
drop TABLE IF EXISTS MembrosExterno;
drop TABLE IF EXISTS Pesquisador;
drop TABLE IF EXISTS Bolsista;
DROP TABLE IF EXISTS "pontosDeControle";
drop TABLE IF EXISTS orthomosaico;
drop TABLE IF EXISTS "PlanoDeVoo";
drop TABLE IF EXISTS "LocalVoo";
drop TABLE IF EXISTS Camera;
drop TABLE IF EXISTS Usuario;
drop type IF EXISTS nome;
drop type IF EXISTS Orientado;

-- Foram criados tipos compostos de dados
CREATE type Nome as (
        preNome varchar(20),
        sobreNome varchar(20)
    );


CREATE type Orientado as(

    inicio date,
    fim date,
    orientador char(11)

);

-- Base para todos os usuários
```

```
CREATE TABLE IF NOT EXISTS Usuario(
        CPF char(11) PRIMARY KEY,
        nome Nome NOT NULL,
        email varchar(50) NULL
);

-- Usuário presente no projeto
CREATE TABLE IF NOT EXISTS MembrosInterno(
    matricula varchar(50) PRIMARY KEY,
    CPF char(11) unique NOT NULL,
    FOREIGN KEY (CPF) REFERENCES Usuario (CPF)
);

-- Usuário que observa de fora do projeto
CREATE TABLE IF NOT EXISTS MembrosExterno(
    idUserExterno serial PRIMARY KEY,
    CPF char(11) unique NOT NULL,
    FOREIGN KEY (CPF) REFERENCES Usuario (CPF)
);

-- Usuários responsaveis pela pesquisa
CREATE TABLE IF NOT EXISTS Pesquisador (
        titulaçao varchar(30) NOT NULL,
        lattes varchar(256) NOT NULL,
        CPF char(11) PRIMARY KEY NOT NULL,
        FOREIGN KEY (CPF) REFERENCES Usuario (CPF)

);

-- Colaboradores subordinados aos pesquisadores
CREATE TABLE IF NOT EXISTS bolsista (
    orientado Orientado[],
    CPF char(11) PRIMARY KEY NOT NULL,
    FOREIGN KEY (CPF) REFERENCES Usuario (CPF)
);

-- Tabela para armazenar as cameras do projeto
CREATE TABLE IF NOT EXISTS camera(
    idCamera serial PRIMARY KEY,
    marca varchar(100) NOT NULL,
    lente varchar(100) NOT NULL,
    modelo varchar(100) NOT NULL,
    larguraPixels int NOT NULL,
    alturaPixels int NOT NULL,
    codigoSerial varchar(50) NULL
);


-- Tabela para armazenar informaçoes dos locais que foram feitos os voos
CREATE TABLE IF NOT EXISTS "LocalVoo"(
```

```
    cnpj char(14) PRIMARY KEY,
    nome varchar(100) NOT NULL,
    campanha varchar NOT NULL,
    "dataDaCampanha" timestamp NOT NULL
);


-- Tabela para armazenar informaçoes dos planos de voo
CREATE TABLE IF NOT EXISTS "PlanoDeVoo"(
    idVoo serial PRIMARY KEY,
    altitude float NOT NULL,
    gsd float NOT NULL,
    arquivoAgisoft bytea NOT NULL,
    localidade GEOGRAPHY(POINT,4326) NOT NULL,
    cnpj char(14) REFERENCES "LocalVoo"(CNPJ)
);

-- Tabela para armazenar orthomosaicos criados
CREATE TABLE IF NOT EXISTS orthomosaico(
    idOrthomosaico serial PRIMARY KEY,
    imagemOrtho bytea NOT NULL,
    arquivoGis bytea NULL,
    descricao varchar(256) NULL
);

-- Tabela para armazenar os pontos de controle
CREATE TABLE IF NOT EXISTS "pontosDeControle"(
    id serial PRIMARY KEY,
    points geometry(point, 4674) NULL,
    "Tag" varchar NULL,
    "Long" float NULL,
    "Lat" float NULL,
    cnpj char(14) REFERENCES "LocalVoo"(cnpj),
    idOrthomosaico int NULL,
    FOREIGN KEY (idOrthomosaico) REFERENCES orthomosaico (idOrthomosaico)
);

-- Tabela principal, onde vamos armazenar informaçoes das imagens
CREATE TABLE IF NOT EXISTS Imagem(
    idImagem serial PRIMARY KEY,
    imagem bytea NOT NULL,
    dataHora timestamp NOT NULL,
    descricao varchar(256) NULL,
    CPF_descricao char(11) NULL,
    CPF char(11) NOT NULL,
    idVoo int NOT NULL,
    idCamera int NOT NULL,
    idOrthomosaico int NULL,
    localidade_img GEOGRAPHY(POINT,4326) NULL,
    FOREIGN KEY (CPF) REFERENCES Usuario (CPF),
```

```
    FOREIGN KEY (CPF_descricao) REFERENCES Usuario (CPF),
    FOREIGN KEY (idVoo) REFERENCES "PlanoDeVoo" (idVoo),
    FOREIGN KEY (idCamera) REFERENCES Camera (idCamera),
    FOREIGN KEY (idOrthomosaico) REFERENCES orthomosaico (idOrthomosaico)
) ;



--------------------------------------------------------------------------------
---------------------- Procedures ----------------------------------------------
--------------------------------------------------------------------------------

-- Proc insere usuario
CREATE OR REPLACE PROCEDURE insere_usuario(CPF char(11), nome Nome, email varchar(50))
LANGUAGE SQL
AS $$
    INSERT INTO Usuario VALUES (CPF, nome, email);
$$;


-- Proc insere MembrosInterno
CREATE OR REPLACE PROCEDURE MembrosInterno(matricula varchar(50), CPF2 char(11), nome Nome, email
varchar(50))
AS $$
    BEGIN
    IF NOT EXISTS(SELECT * FROM usuario U WHERE U.cpf = CPF2) THEN
    CALL insere_usuario(CPF2, nome, email);
    END IF;
    INSERT INTO MembrosInterno VALUES (matricula, CPF2);
    END;
$$
LANGUAGE plpgsql;

-- Proc insere MembrosExterno
CREATE OR REPLACE PROCEDURE MembrosExterno(CPF2 char(11), nome Nome, email varchar(50))
AS $$
BEGIN
    IF NOT EXISTS(SELECT * FROM usuario U WHERE U.cpf = CPF2) THEN
    CALL insere_usuario(CPF2, nome, email);
    END IF;
    INSERT INTO MembrosExterno(CPF) VALUES (CPF2);
    end;
$$ LANGUAGE plpgsql;


-- Proc insere pesquisador
CREATE OR REPLACE PROCEDURE insere_pesquisador(titulaçao varchar(30), lattes varchar(256), CPF2
char(11))
AS $$
BEGIN
```

```
    if NOT exists(select * from bolsista p WHERE cpf2=p.cpf) THEN
        INSERT INTO Pesquisador VALUES (titulaçao, lattes, CPF2);
    else
        RAISE NOTICE 'Não é possivel adicionar um pesquisadoor que já é bolsista';
    end if;

END;
$$ LANGUAGE plpgsql;




-- Proc insere bolsista
CREATE
OR REPLACE procedure insere_bolsista(
    CPF2 varchar(11),
    nome Nome,
    email varchar(50),
    orientado_p Orientado
) as $$
BEGIN
    IF EXISTS(SELECT * FROM pesquisador p WHERE orientado_p.orientador = p.cpf) THEN  -- precisa ter um
pesquisador
        if NOT exists(select * from pesquisador p WHERE cpf2=p.cpf) THEN -- o bolsista nao pode ser um
pesquisador
            if EXISTS(select * from Bolsista b where b.cpf=cpf2 and
b.orientado[array_length(b.orientado,1)].fim < orientado_p.inicio) THEN
                UPDATE Bolsista set orientado[array_length(orientado,1)] = orientado_p where cpf = cpf2;
-- atualiza datas
            else
                if NOT EXISTS(select * from Bolsista b where b.cpf=cpf2) THEN
                    INSERT INTO Bolsista VALUES (array[orientado_p], CPF2); -- cria um novo
                ELSE
                RAISE NOTICE 'Não é possivel adicionar mais de um orientador por periodo de tempo';
                END IF;
                END IF;
        else
            RAISE NOTICE 'Não é possivel adicionar um bolsista que já é pesquisador';
        end if;
    ELSE
        RAISE NOTICE 'Orientador não existe.';
    END IF;
END;
$$ LANGUAGE plpgsql;



-- Proc insere camera
CREATE OR REPLACE PROCEDURE insere_camera(  marca varchar(100),
                                            lente varchar(100),
```

```
                                        modelo varchar(100),
                                        larguraPixels int,
                                        alturaPixels int,
                                        codigoSerial int)
    LANGUAGE SQL
    AS $$
        INSERT INTO Camera( marca,
                            lente,
                            modelo,
                            larguraPixels,
                            alturaPixels,
                            codigoSerial)
                            VALUES (marca,
                                    lente,
                                    modelo,
                                    larguraPixels,
                                    alturaPixels,
                                    codigoSerial);
$$;



-- Proc insere local
CREATE OR REPLACE PROCEDURE insere_local(cnpj char(14),
                                    nome varchar(100),
                                    campanha varchar,
                                    "dataDaCampanha" timestamp )
LANGUAGE SQL
AS $$
    INSERT INTO "LocalVoo" VALUES (cnpj, nome, campanha, "dataDaCampanha");
$$;


-- Proc insere PlanoDeVoo
CREATE OR REPLACE PROCEDURE insere_planodevoo ( cnpj char(14),
                                            altitude float,
                                            gsd float,
                                            arquivoAgisoft bytea,
                                            localidade GEOGRAPHY(POINT,4326))
--ST_GeographyFromText('POINT(-46.633309 -23.550520)')
LANGUAGE SQL
AS $$
    INSERT INTO "PlanoDeVoo"( altitude,
                            gsd,
                            arquivoAgisoft,
                            localidade,
                            cnpj)
                            VALUES ( altitude,
                            gsd,
                            arquivoAgisoft,
```

```
                            localidade,
                            cnpj)
$$;

-- Proc insere pontos de controle
CREATE OR REPLACE PROCEDURE insere_pontos_de_controle(points geometry(point, 4674),
                                                      "Tag" varchar,
                                                      "Long" float,
                                                      "Lat" float,
                                                      cnpj char(14),
                                                      idOrthomosaico int)
LANGUAGE SQL
AS $$
INSERT INTO "pontosDeControle"(points,
                               "Tag",
                               "Long",
                               "Lat",
                               cnpj,
                               idOrthomosaico
                               )
                           values (points,
                               "Tag",
                               "Long",
                               "Lat",
                               cnpj,
                               idOrthomosaico)
$$;

-- Proc insere orthomosaico
CREATE OR REPLACE PROCEDURE insere_orthomosaico(imagemOrtho bytea,
                                                arquivoGis bytea,
                                                descricao varchar(256))
LANGUAGE SQL
AS $$
INSERT INTO orthomosaico(   imagemOrtho,
                            arquivoGis,
                            descricao)
                            values (imagemOrtho,
                                    arquivoGis,
                                    descricao)
$$;

-- Proc insere imagem
CREATE OR REPLACE PROCEDURE insere_Imagem(
    imagem bytea,
    dataHora timestamp,
    CPF char(11),
    idVoo int,
    idCamera int,
    localidade_img GEOGRAPHY(POINT,4326))
```

```
LANGUAGE SQL
AS $$
INSERT INTO Imagem( imagem,
                    dataHora,
                    CPF,
                    idVoo,
                    idCamera,
                    localidade_img)
                    values( imagem,
                            dataHora,
                            CPF,
                            idVoo,
                            idCamera,
                            localidade_img)
$$;


-- Atualiza com a descrição e quem a colocou a descrição
CREATE OR REPLACE PROCEDURE atualiza_descricao_Imagem(
    idImagemP int,
    descricaoP varchar(256),
    CPF_descricaoP char(11))
LANGUAGE SQL
AS $$
    UPDATE Imagem SET descricao = descricaoP, CPF_descricao = CPF_descricaoP WHERE idImagem = idImagemP;
$$;

-- Atualiza com o id do orthomosaico que ela participa
CREATE OR REPLACE PROCEDURE atualiza_Orthomosaico_da_Imagem(
    idImagemP int,
    idOrthomosaicoP int)
LANGUAGE SQL
AS $$
    UPDATE Imagem SET idOrthomosaico = idOrthomosaicoP WHERE idImagem = idImagemP;
$$;
-------------------------------------------------------------------------------
--------------- Functions -----------------------------------------------
-------------------------------------------------------------------------------


-------------------------------------------------------------------------------
-------------- Verifica se o ponto está contido no poligono --------------
CREATE OR REPLACE FUNCTION confere_ponto_cidade (
    id_ponto INT,
    id_cidade INT
)RETURNS boolean

AS $$
DECLARE
pertence boolean;
BEGIN
```

```
    SELECT ST_Contains(m.geom, p.geom) INTO pertence
    FROM "Municipios_MT_2021" m
    JOIN pontos p ON ST_Contains(m.geom, p.geom)
    WHERE p.id = id_ponto and m.id = id_cidade;
    RETURN CASE
        WHEN pertence THEN true
        ELSE false
    END;
END;
$$ LANGUAGE plpgsql;



-------------------------------------------------------------------------------
--------------- Passa a cidade e encontra os pontos ---------------------------
CREATE OR REPLACE FUNCTION encontra_ponto_por_cidade (
    id_cidade INT
)RETURNS SETOF geometry

AS $$
DECLARE
pontos geometry;
BEGIN FOR pontos IN
    SELECT p.geom
    FROM "Municipios_MT_2021" m
    JOIN pontos p ON ST_Contains(m.geom, p.geom)
    WHERE m.id = id_cidade
    LOOP
    RETURN NEXT pontos;
    END LOOP;

    RETURN;
END;
$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------
--------------- Encontra a cidade através de um ponto ---------------
CREATE OR REPLACE FUNCTION encontra_cidade (
    id_ponto INT
)RETURNS text

AS $$
DECLARE
nome_cidade text;
BEGIN


    SELECT m.nm_mun INTO nome_cidade
    FROM "Municipios_MT_2021" m
    JOIN pontos p ON ST_Contains(m.geom, p.geom)
    WHERE p.id = id_ponto;
```

```
    RETURN nome_cidade;
END;
$$ LANGUAGE plpgsql;
```

Anexo 3 - Script Python

```
import psycopg2
import os

# Conecta ao banco de dados
conn = psycopg2.connect(
    "dbname=geoDatabase user=postgres password=postgis host=localhost")

print('\n\nBem Vindo ao Sistemas de Gerenciamento de dados Georeferênciados!')

def isValid(res):
    if res.isdigit():
        return
    else:
        print('Opção Inválida, por favor tente novamente.')

def cadastrarUsuario():
    while True:
        res = input(
            '\nCadastrar Usuário:\n1:Organização Interna\n2:Organização Externa\n')
        isValid(res)
        res = int(res)
        if (res == 1):
            with conn.cursor() as cur:
                nome = input('Digite o nome do usuário: ')
                sobrenome = input('Digite o sobrenome do usuário: ')
                cpf = input('Digite o CPF do NOVO usuário: ')
                mat = input('Digite a Mátricula do NOVO usuário: ')
                email = input('Digite o email do NOVO usuário: ')
                try:
                    cur.execute("CALL MembrosInterno(%s, %s, (%s, %s), %s)",
                                (mat, cpf, nome, sobrenome, email))
                    conn.commit()
                except psycopg2.Error as e:
                    # Se ocorrer um erro, exibe a mensagem de erro e faz o rollback da
transação
                    print("Ocorreu um erro:", e)
                    conn.rollback()

            cadastrar = input(
                '\nCadastrar Interno:\n1:Pesquisador\n2:Bolsista\nDigite qualquer outra
coisa se não deseja cadastrar o interno ou me passa um pix \n')
            if cadastrar == '1':
                cadastrarPesquisador(cpf)
            elif cadastrar == '2':
```

```
                cadastrarBolsista(cpf, nome, sobrenome, email)
            break


        elif (res == 2):
            with conn.cursor() as cur:
                nome = input('Digite o nome do NOVO usuário: ')
                sobrenome = input('Digite o sobrenome do NOVO usuário: ')
                cpf = input('Digite o CPF do NOVO usuário: ')
                email = input('Digite o email do NOVO usuário: ')
                try:
                    cur.execute("CALL MembrosExterno( %s, (%s, %s), %s)",
                                (cpf, nome, sobrenome, email))
                    conn.commit()
                except psycopg2.Error as e:
                    # Se ocorrer um erro, exibe a mensagem de erro e faz o rollback da
transação
                    print("Ocorreu um erro:", e)
                    conn.rollback()
            break
        else:
            print('Opção Inválida, por favor tente novamente.')


def cadastrarPesquisador(CPF):
    with conn.cursor() as cur:
        titulaçao = input('Qual a titulação do pesquisador? ')
        lattes = input('Qual o perfil lattes do pesquisador? ')
        try:
            cur.execute('CALL insere_pesquisador(%s, %s, %s)',
                        (titulaçao, lattes, CPF))
            conn.commit()
        except psycopg2.Error as e:
            # Se ocorrer um erro, exibe a mensagem de erro e faz o rollback da transação
            print("Ocorreu um erro: ", e)
            conn.rollback()


    return

def cadastrarBolsista(CPF, nome, sobrenome, email):
    with conn.cursor() as cur:
        dataInicio = input(
            'Digite a data de início da Orientaçâo (Ex.:YYYY-MM-DD): ')
        dataFim = input(
            'Insira o final do periodo de orientação (Ex. YYYY-MM-DD): ')
        CPF_orientador = input('Digite o CPF do Orientador: ')
        try:
            cur.execute('CALL insere_bolsista(%s,(%s, %s), %s, (%s, %s, %s ))',
                        (CPF, nome, sobrenome, email, dataInicio, dataFim, CPF_orientador))
            conn.commit()
        except psycopg2.Error as e:
```

```
            # Se ocorrer um erro, exibe a mensagem de erro e faz o rollback da transação
            print("Ocorreu um erro:", e)
            conn.rollback()

def planoDeVoo():
    print('Salvar Plano de Voo')
    cnpj = input('\nDigite o CNPJ do Local: ')
    altitude = float(input('Altitude do Voo: '))
    gsd = float(input('Qual o GSD estimado: '))
    diretorio = input('Entre com o diretório da arquivo:')
    with open(diretorio, "rb") as f:
        arquivo = f.read()
    lat = float(input('Entre com o valor da latitude do local: '))
    log = float(input('Entre com o valor da longitude do local: '))
    with conn.cursor() as cur:
        cur.execute("CALL insere_planodevoo(%s, %s, %s, %s, ST_GeographyFromText('POINT(%s
%s)'))",
                    (cnpj, altitude, gsd, arquivo, lat, log))
        conn.commit()


def defineLocal():
    print('Local de Voo')
    nome = input('Insira o nome do Local do Voo: ')
    cnpj = input('Informe o CNPJ: ')
    campanha = input('Insira o nome da campanha: ')
    dataDaCampanha = input('Insira a data da campanha: ')
    with conn.cursor() as cur:
        try:
            cur.execute('CALL insere_local(%s, %s, %s, %s)',
                        (cnpj, nome, campanha, dataDaCampanha))
            conn.commit()
        except psycopg2.Error as e:  # Se ocorrer um erro, exibe a mensagem de erro e faz o
rollback da transação
            print("Ocorreu um erro:", e)
            conn.rollback()

def imagens():
    from exif import Image
    import datetime
    print('''
    Para inserir as imagens crie uma pasta coloque as imagens desejadas depois
    copie o diretório da pasta correspondente e cole no campo quando solicitado''')
    CPF = input('Digite o seu CPF: ')
    with conn.cursor() as cur:
        cur.execute('SELECT * FROM "LocalVoo"')
        list_local_voo = cur.fetchall()
        for i in list_local_voo:
            print(list_local_voo.index(i), ':', 'Local:', i[1], 'CNPJ:', i[0])
        res = int(input('Escolha o Local do Voo: '))
        cur.execute(
            f"SELECT * FROM planodevoo where cnpj = '{list_local_voo[res][0]}'")
```

```
        list_plano_voo = cur.fetchall()
        for i in list_plano_voo:
            print(list_plano_voo.index(i), ':',
                  ' Altitude:', i[1], ' GSD:', i[2])
        res_voo = int(input('Escolha o plano de voo: '))
        cur.execute('SELECT * FROM camera')
        list_camera = cur.fetchall()
        for i in list_camera:
            print(list_camera.index(i), ':', 'Modelo:', i[3], 'Marca:', i[1])
        res_camera = int(input('Escolha o tipo da camera: '))
        diretorio = input('Insira o diretório da pasta imagens: ')
        # Problemas aqui ele não está recebendo somente o valor
        idVoo = (list_plano_voo[res_voo][0])
        idCamera = (list_camera[res_camera][0])

        for diretorio, subpastas, arquivos in os.walk(diretorio):
            for arquivo in arquivos:
                with open(diretorio + '\\' + arquivo, "rb") as f:
                    img = Image(f)
                    if (img.has_exif):
                        lat = img.get('gps_latitude')
                        lat = lat[2]
                        log = img.get('gps_longitude')
                        log = log[2]
                        coordenada = "ST_GeographyFromText('POINT(%s %s)')", (
                            lat, log)
                        dataHora = img.get('datetime_original')
                        dataHora = dataHora[:10].replace(
                            ':', '-') + dataHora[10:]
                    else:
                        DateSys = datetime.datetime.now()
                        dataHora = DateSys.strftime('%Y-%m-%d %H:%M:%S')
                with open(diretorio + '\\' + arquivo, "rb") as f:
                    imagem = f.read()
                    cur.execute("CALL insere_Imagem(%s, %s, %s, %s, %s,
ST_GeographyFromText('POINT(%s %s)'))",
                                (imagem, dataHora, CPF, idVoo, idCamera, lat*-1, log*-1))
                    conn.commit()


def test():
    import csv
    print('Inserir Orthomosaico')
    diretorioOrthomosaico = input(
        '\nInsira o diretório com o nome do arquivo do Orthomosaico: ')
    with open(diretorioOrthomosaico, "rb") as arq1:
        imagemOrthomosaico = arq1.read()
    ditetorioArquivoGis = input(
        'Insira o diretório com o nome do arquivo do Arquivo GIS: ')
    with open(ditetorioArquivoGis, "rb") as arq2:
        arquivoGis = arq2.read()
```

```
    descrição = input(
        'Escreva algum comentário sobre a imagem, caso contrário deixe vazio: ')
    with conn.cursor() as cur:
        cur.execute("CALL insere_orthomosaico(%s, %s, %s)",
                    (imagemOrthomosaico, arquivoGis, descrição))
        conn.commit()
    while True:
        res = input(
            '\nDeseja Inserir os pontos de controle:\n1:SIM\n2:NÃO\n')
        isValid(res)
        res = int(res)
        if (res == 1):
            with conn.cursor() as cur:
                cur.execute('SELECT idorthomosaico FROM orthomosaico')
                list_local_voo = cur.fetchall()
            diretorioCsv = input(
                '\nInsira o "diretório + nome_do_arquivo" csv dos pontos GPS: ')
            with open(diretorioCsv) as arquivo_csv:
                pontos = csv.reader(arquivo_csv, newline='')
                print(pontos)
        elif (res == 2):
            return
        else:
            return

def test2(idortho=None):
    import pandas as pd
    cnpj = '12345678912301'
    diretorioCsv =
'C:\\Users\\phcmo\\OneDrive\\Doutorado\\2022_2\\DBNC\\Project\\python\\points\\Sorriso.csv'
    with open(diretorioCsv) as arquivo_csv:
        pontos = pd.read_csv(arquivo_csv, header=None)
    for i in range(len(pontos)):
        tag = pontos.iloc[i, 0]
        lon = pontos.iloc[i, 1]
        lat = pontos.iloc[i, 2]
        with conn.cursor() as cur:
            cur.execute("CALL insere_pontos_de_controle(ST_MakePoint(%s, %s), %s, %s, %s,
%s, %s)",
                        (lon, lat, tag, lon, lat, cnpj, idortho))
            conn.commit()


def test3():
    import rasterio
    import psycopg2

    # Abre o arquivo raster com o Rasterio
    with rasterio.open("python/ortho/IMA_Sorriso.tif") as src:
        # Extrai os metadados do raster
        meta = src.meta
```

```
        # Cria uma nova tabela no PostGIS para armazenar o raster
        with conn.cursor() as cur:
            cur.execute(
                "CREATE TABLE t_raster (rid serial PRIMARY KEY, rast raster)")
            conn.commit()
        # Carrega o conteúdo do arquivo raster para o PostGIS
        with conn.cursor() as cur:
            with open("python/ortho/IMA_Sorriso.tif", "rb") as f:
                cur.execute(
                    "INSERT INTO your_table (rast) VALUES
(ST_SetBandNoDataValue(ST_AddBand(ST_MakeEmptyRaster(%(width)s, %(height)s, %(resolution)s,
%(bounds)s), 1), 1, %(nodata)s)) RETURNING rid;",
                    {
                        "width": meta["width"],
                        "height": meta["height"],
                        "resolution": meta["transform"][0],
                        "bounds": meta["bounds"],
                        "nodata": meta["nodata"]                    },
                    psycopg2.Binary(f.read()),
                )

    # Fecha a conexão com o banco de dados PostGIS
    conn.close()

################# MENU PRINCIPAL ##############
while True:
    res = input('''\nEscolha as opções a seguir:
    1:Cadastrar Usuário
    2:Cadastrar Local "Campo"
    3:Salvar Plano de Voo
    4:Inserir imagens
    5:Inserir Orthomosaico
    6:Inserir descrição
    7:Inserir pontos de controle
    8:Testes dev\n''')
    isValid(res)
    res = int(res)
    if (res >= 1 and res <= 8):
        if res == 1:
            cadastrarUsuario()
        if res == 2:
            defineLocal()
        if res == 3:
            planoDeVoo()
        if res == 4:
            imagens()
        if res == 8:
            test3()
        break
    else:
        print('Opção Inválida, por favor tente novamente.')
```