ParaibaHidroGIS

Release 0.1

COBRAPE Curitiba

15 jul. 2024

Sumário

Paraiba HidroGIS é um aplicativo desenvolvido para análise de bacias hidrográficas desenvolvido no contexto do Plano de Recursos Hídricos da Bacia Hidrográfia do Rio Paraíba (PRH-RPB).

Nota: Este projeto está em desenvolvimento ativo.

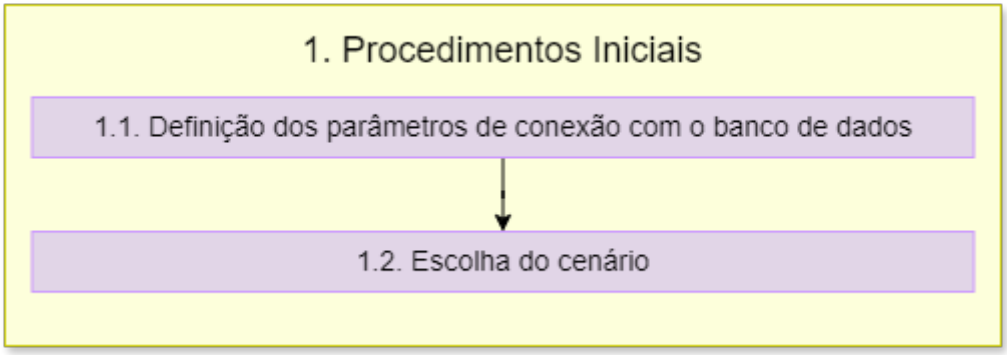
CAPÍTULO 1

Conteúdo

1.1 1. Conexão com o Banco de Dados

> **Aviso:** A última versão do banco é a **versão 8** e deve ser utilizada para execução com a última atualização da aplicação. Isto é importante para que haja compatibilidade e evitar erros de execução difíceis de serem compreendidos pelo usuário.

O fluxograma de processos desta etapa é apresentado a seguir:



1.1.1 1.1 Definição dos parâmetros de conexão com o BDG

A função **parametros_padrao_bd** define os parâmetros de conexão *padrão* com o banco de dados.

host	localhost
nome	bdg_prh_rpb
usuario	postgres
senha	cobrape
porta	5432

A variável **parametros_conexao** cria um dicionário que contém parâmetros de conexão padrão (host, nome do banco, usuário, senha, porta e schema) com o banco de dados.

1.1.1 Verificação da conexão PostGIS

A função **verifica_parametros_bd** apresenta os parâmetros de conexão com o banco de dados e possibilita ao usuário decidir se mantém os parâmetros de conexão padrão ou se deseja inserir parâmetros personalizados.

A variável **verifica_postgis** utiliza a classe **QMessageBox** para verificar se o usuário deseja continuar com os parâmetros de conexão padrão ou se deseja alterar os parâmetros de entrada.

Nota: A classe **QMessageBox** faz parte do framework Qt e é usada para criar e gerenciar caixas de diálogo que exibem mensagens para o usuário, podendo ser utilizadas para fornecer informações, pedir confirmação ou solicitar entrada do usuário.

Se a resposta do usuário for *sim*, a leitura do código será continuada e serão utilizados os parâmetros padrão.

Se a resposta for *não*, o código segue para a função **patrametros_personalizados_bd**.

1.1.2. Definição dos parâmetros personalizados

A função **parametros_personalizados_bd** utiliza a classe **QInputDialog** para obter novos valores para os parâmetros de conexão. A classe é utilizada para cada parâmetro de conexão (host, nome do banco, usuário, senha, porta e schema) e, portanto, o processo é repetido seis vezes.

O código faz a verificação se algum campo ficou vazio. Em casos que o usuário deixe os campos vazios, será utilizado os parâmetros de conexão padrão.

Depois de inserir os valores, é chamada a função **verifica_parametros_bd** onde são atualizados os parâmetros de conexão.

Nota: A classe **QInputDialog** faz parte do framework Qt e é utilizada para criar caixas de diálogo que solicitam entrada do usuário. Essas caixas de diálogo podem ser usadas para coletar informações como texto, números ou opções de uma lista.

1.1.2 1.2. Escolha do cenário

Os cenários foram criados nos Schemas do banco de dados. Os Schemas estão divididos por:

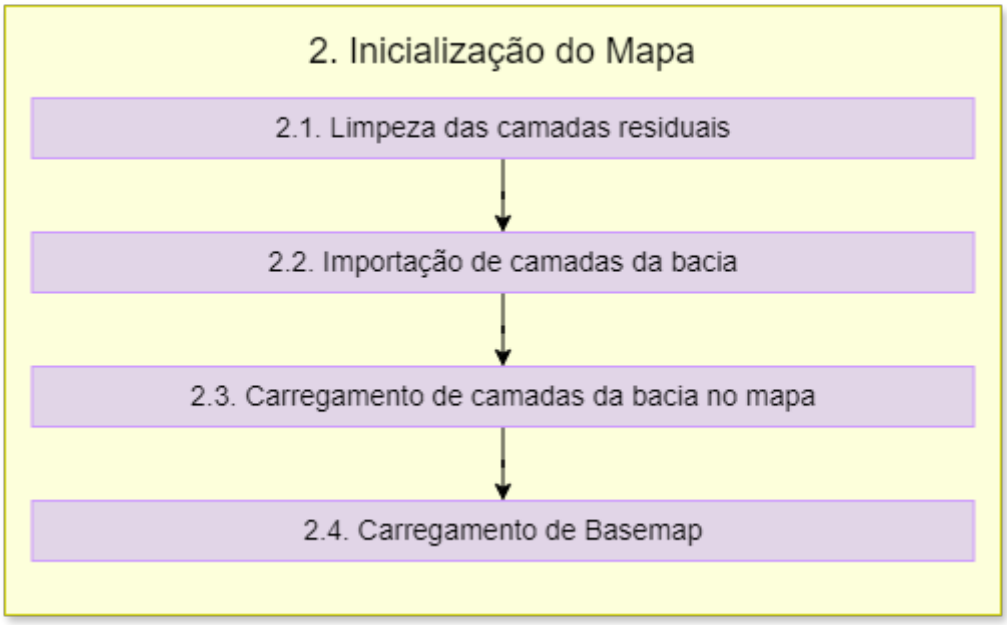
- **Basemap**: onde contém as ottobacias e ottotrechos.
- **Cenário 0**: vazão incremental 1 e captação 0 para todas as ottobacias.
- **Cenário 1**: 10 ottobacias com captação 1 distribuídas pela bacia.
- **Cenário 2**: algumas ottobacias estão com valores diferentes para realizar testes das classificações do ISR.

Os resultados são salvos no próprio Schema de cada cenário, em forma de Visualizações (VIEWS). Estes cenários criados são apenas para fins de testes.

A função **definir_cenario** exibe inicialmente uma caixa de mensagem perguntando ao usuário se ele deseja alterar o cenário atual. Caso a resposta for *Sim*, será solicitado que ele insira o nome de um novo cenário. Assim o *novo_cenario** substituirá o Schema atual no dicionário de parâmetros de conexão.

1.2 2. Inicialização do Mapa

O fluxograma de processos desta etapa é apresentado a seguir:



O **import requests** faz a importação da biblioteca Requests, a qual é utilizada para fazer requisições HTTP em python de forma simples e eficiente.

1.2.1 2.1. Limpeza das camadas residuais

A função **limpeza_residuos** realiza a limpeza de camadas residuais do projeto no QGIS.

A variável **camada_residual** utiliza o **QgsProject.instance** para obter um dicionário de todas as camadas do projeto usando o método mapLayers() da instância do projeto e, em seguida, obtém os valores desse dicionário, resultando em uma lista de todas as camadas no projeto.

Nota: O **QgsProject** é uma classe central no QGIS que representa o projeto em si. Ele armazena informações sobre camadas, configurações do projeto, sistemas de coordenadas e outros elementos relacionados ao ambiente de trabalho no QGIS.

Após a criação da lista com as camadas do projeto, é feita a verificação da existência de camadas residuais, caso haja qualquer camada residual é feita a remoção da mesma. Caso contrário, será apenas obtido e atualizado o canvas do mapa do projeto utilizando o **qgis.utils.iface.mapCanvas()** e o **canvas.refresh()**.

Nota: O **iface** é uma instância da classe QgisInterface que fornece acesso às interfaces do QGIS para plugins. O *mapCanvas* é o método utilizado para obter a referência à tela de visualização do mapa atual no QGIS.

1.2.2 2.2. Importação de camadas da bacia

Nesse processo será feita a importação das camadas do banco de dados.

A função **importar_camada_bdg** recebe informações sobre o banco de dados (nome, schema, nome da camada) para importar a camada vetorial correspondente.

A variável *uri* utiliza o **QgsDataSourceUri** para armazenar informações sobre a fonte de dados da camada vetorial, e, posteriormente, configura as informações de conexão com o banco de dados na URI.

Nota: O **QgsDataSourceUri** é uma classe na biblioteca QGIS que é usada para representar e manipular informações de conexão com fontes de dados, como bancos de dados espaciais, arquivos shapefile, serviços da web, entre outros. Essa classe permite que você construa e manipule de forma pragmática URIs (Uniform Resource Identifiers) que especificam a fonte de dados que será utilizada em um projeto QGIS.

A variável *camada_importada* cria um objeto **QgsVectorLayer** usando a URI configurada e define o nome da camada.

Nota: O **QgsVectorLayer** é uma classe na biblioteca QGIS que representa uma camada vetorial dentro do ambiente QGIS. Essa classe é parte da API do QGIS e é usada para manipular dados vetoriais, como pontos, linhas e polígonos.

1.2.3 2.3. Carregamento de camadas da bacia no mapa

A função **carregar_camada** é responsável por configurar a simbologia de uma camada. Para isso, a variável **tipo_geometria** obtém o tipo de geometria da camada fornecida como parâmetro e depois é verificado se é um ponto. A classe **QgsWkbTypes** é usada para verificação do tipo de geometria, enquanto as classes **QgsSimpleMarkerSymbolLayer**, **QgsLineSymbol** e **QgsFillSymbol** são utilizadas para representar a simbologia das camadas.

- Se for um ponto, é criado um símbolo de marcador simples com base nos parâmetros de cor e tamanho fornecidos na simbologia. Se não for ponto, verifica se é uma linha. Os parâmetros de simbologia que podem ser modificados são a cor e o tamanho.
- Se for uma linha, cria um símbolo de linha simples com base nos parâmetros de cor e espessura fornecidos na simbologia. Se não for uma linha, verifica se é uma geometria de polígono. Os parâmetros de simbologia que podem ser modificados são a cor e a espessura.
- Se for um polígono, cria um símbolo de preenchimento simples com base nos parâmetros de cor de preenchimento, cor de contorno e espessura do contorno fornecidos na simbologia. Os parâmetros de simbologia que podem ser modificados são a cor do preenchimento, a cor da borda e a espessura da borda.

Então é criado um renderizador de símbolo único (**QgsSingleSymbolRenderer**) usando o símbolo definido anteriormente. E por fim, a camada é adicionada ao projeto QGIS.

Nota:

- A classe **QgsWkbTypes** é usada para lidar com tipos Well-Known Binary, que é um formato binário usado para representar objetos geométricos. A classe fornece enumerações e funções para trabalhar com esses tipos WKB.
- A classe **QgsSimpleMarkerSymbolLayer** é utilizada para representar uma camada de símbolo de marcador simples.
- A classe **QgsLineSymbol** é utilizada para representar símbolos de linha em camadas vetoriais.
- A classe **QgsFillSymbol** é utilizada para representar símbolos de preenchimento em camadas vetoriais.

2.3.1. Guia RGBA

A simbologia RGBA (Red, Green, Blue, Alpha) é um modelo de cor usado para definir cores em gráficos, imagens e mapas. Cada componente representa a intensidade de uma cor primária (Red, Green, Blue), enquanto o componente Alpha representa a transparência da cor.

A intensidade varia de 0 a 255 para todos, sendo que quanto maior o valor, mais próximo da saturação e luminosidade da cor estará aquele componente. Para a transparência, quanto maior o valor, menor é a transparência. Os valores também podem ser normalizados de 0 a 1. O preto é representado pelo 0 em todos os componentes.

Exemplo:
azul = (0, 0, 1, 1) vermelho semi-transparente = (1, 0, 0, 0.5)

1.2.4 2.4. Carregamento de basemap

A função **importar_camada_fundo** tem como objetivo carregar uma camada de plano de fundo usando a biblioteca QGIS.

A variável **service_url** contém uma URL para um serviço de mapas do Google. Os placeholders {x}, {y} e {z} são utilizados para representar os valores de latitude, longitude e zoom.

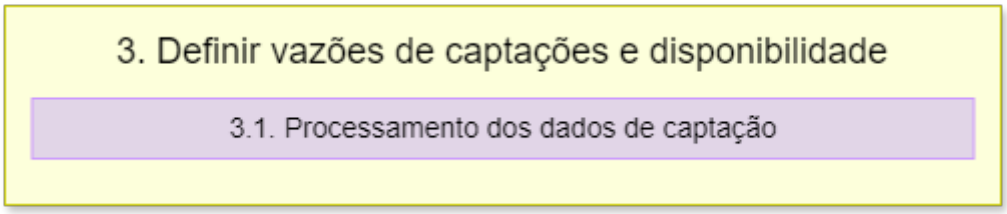
A variável **serivce_uri** contém a URI do serviço de mapas, formatada com os parâmetros necessários. A função **requests.utilis.quote** é usada para garantir que a URL seja codificada corretamente.

A função **iface.addRasterLayer** da interface do QGIS é utilizada para adicionar uma camada raster com os argumentos:

- **service_uri**: a URI do serviço de mapas
- **Google_Road**: nome da camada a ser adicionada
- **wms**: tipo de serviço, indicando que é um Web Map Service

1.3 3. Processamento de captações

O fluxograma de processos desta etapa é apresentado a seguir:



O módulo **psycopg2** é uma biblioteca do PostgreSQL para Python. Essa biblioteca permite a conexão e interação com um banco de dados PostgreSQL a partir do Python.

1.3.1 3.1. Processamento dos dados de captação

A função **processamento_captacoes** é definida para que possa se estabelecer uma conexão com o banco de dados PostgreSQL usando os parâmetros de cone~xão fornecidos anteriormente, além de criar um objeto cursor para executar comandos SQL no banco de dados.

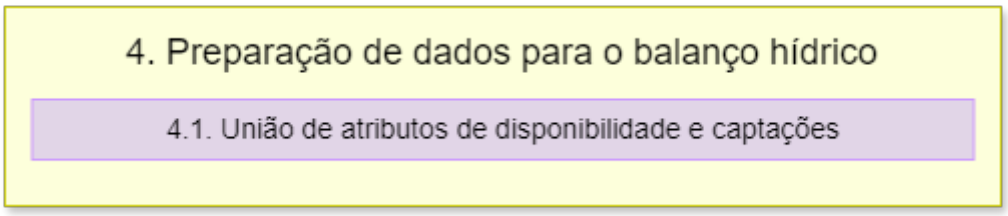
Assim, é executado uma série de comandos SQL que serão descritos a seguir:

- **DROP VIEW IF EXISTS**: é utilizado para remover a view *captacoes_ottobacias* no schema atual se ela existir.
- **CASCADE**: é uma opção que remove todas as dependentes daquela VIEW. Isso garante que não haja nenhum objeto associado à VIEW removida.
- **CREAT VIEW**: é criada uma nova view chamada *captacoes_ottobacias* no schema atual.
- **SELECT, FROM, JOIN, GROUP BY, ORDER BY**: a view realiza uma consulta que seleciona a coluna *cobacia* da tabela *ottobacias_pb_5k* no schema basemap, e então é calculado a soma das captações solicitadas agrupadas por cobacia, usando informações das tabelas outorgas e ottobacias_pb_5k no schema especificado pelos parâmetros de conexão.

Depois disso é feito um commit das mudanças realizadas no banco de dados (conexao.commit), o cursor é fechado (cursor.close) e a conexão com o banco de dados é fechada (conexao.close).

1.4 4. Preparação de dados para balanço hídrico

O fluxograma de processos desta etapa é apresentado a seguir:



Primeiramente é importado o módulo **psycopg2**.

1.4.1 4.1 União entre disponibilidade hídrica e captações

A função **uniao_disp_cap** é definida para que possa se estabelecer uma conexão com o banco de dados PostgreSQL usando os parâmetros de conexão fornecidos anteriormente, além de criar um objeto cursor para executar comandos SQL no banco de dados.

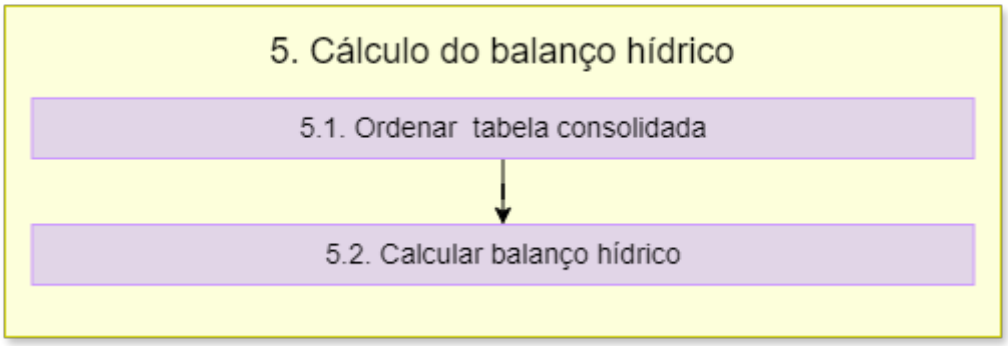
Assim, é executado uma série de comandos SQL que serão descritos a seguir:

- **DROP VIEW IF EXISTS**: é utilizado para remover a view *dados_balanco* no schema atual se ela existir.
- **CASCADE**: é uma opção que remove todas as dependentes daquela VIEW. Isso garante que não haja nenhum objeto associado à VIEW removida.
- **CREAT VIEW**: é criada uma nova view chamada *dados_balanco* no schema atual.
- **SELECT, FROM, LEFT JOIN, ORDER BY**: a view realiza uma consulta que seleciona colunas específicas da tabela *ottotrechos_pb_5k* no schema *basemap* e colunas da tabela *disponibilidade_hidrica* no schema especificado pelos parâmetros de conexão, e posteriormente, as tabelas são unidas.
- **COALESCE**: é utilizado para lidar com valores nulos. Caso a coluna *captacao_solicitada* da tabela *captacoes_ottobacias* for nula, ela será substituída por 0.

Depois disso é feito um commit das mudanças realizadas no banco de dados (conexao.commit), o cursor é fechado (cursor.close) e a conexão com o banco de dados é fechada (conexao.close).

1.5 5. Cálculo do balanço hídrico

O fluxograma de processos desta etapa é apresentado a seguir:



Primeiramente é importado o módulo **psycopg2**.

1.5.1 5.1 Ordenar tabela consolidada

A função **criar_matriz** cria uma matriz de balanço hídrico a partir dos dados do banco de dados.

Para isso é aberta a conexão com o banco de dados usando os parâmetros fornecidos. Depois, é criado um cursor a partir da conexão com o banco de dados para executar comandos SQL.

A primeira consulta SQL realizada seleciona (SELECT) todas as colunas da tabela **dados_balanco** no esquema especificado **schema_cenario** (FROM), ordenando (ORDER BY) os resultados pela coluna **cobacia** em ordem descendente (DESC).

A variável **rows** recupera todas as linhas resultantes da consulta SQL, enquanto a variável **matriz** será utilizada para armazenar os dados recuperados do banco de dados. Depois, é feito uma iteração sobre cada linha recuperada do banco de dados, onde para cada linha, a iteração converte os valores em uma lista de strings e adiciona essa lista à **matriz**.

A iteração da linha na matriz é realizada para adicionar zeros para criar novos campos que futuramente vão ser preenchidos no cálculo do balanço.

1.5.2 5.2 Calculo do Balanço Hídrico

A função **calcular_balanco** calcula o balanço de disponibilidade e captação de recursos hídricos em cada linha da matriz.

Para isso são realizadas iterações sobre cada linha da matriz, onde primeiramente é realizada a verificação se a linha representa um trecho de cabeceira ou não. Se o trecho for de cabeceira, a função calcula a vazão jusante subtraindo a captação solicitada da vazão incremental na linha atual da matriz.

Então, é verificado se a vazão jusante calculada é menor que zero, caso a resposta seja sim, é calculado o déficit e depois, definida a vazão jusante como zero para calcular a captação atendida. Caso a resoista seja não, então a captação atendida é igual à captação solicitada e o déficit é zero.

Se o trecho não for de cabeceira, a função calcula a vazão jusante considerando os trechos montantes correspondentes. É feita uma iteração reversa para encontrar trechos montantes, ou seja, o loop faz a iteração sobre as linhas anteriores à linha atual, de trás para frente, comparando o código do trecho atual com o código do trecho jusante do trecho anterior.

Se houver um trecho montante correspondente, a vazão jusante do trecho atual é calculada somando as vazões jusantes dos trechos montantes. A vazão montante é armazenada na matriz. A vazão jusante é calculada somando a vazão montante calculada ao valor da disponibilidade local e subtraindo a captação. E então é calculado o déficit, semelhante ao explicado acima, com base na vazão jusante calculada.

O **campo_captacao_acumulada** atualiza a captação acumulada somando a mesma com as linhas anteriores, tendo um máximo de soma dela mesma com duas linhas anteriores.

O cálculo do **Índice de Condição de Recursos Hídricos (ISR)** é feito com base na relação entre a soma da captação acumulada com o déficit e a vazão natural. Essa relação é dividida em intervalos e o ISR é atribuído de acordo com esses intervalos.

Por fim, a função retorna a matriz com os cálculos de vazão jusante, déficit e ISR para cada linha.

5.2.1 Salvar resultados

A função **salvar_resultado** tem a finalidade de salvar os resultados do balanço hídrico em um banco de dados PostgreSQL. Primeiramente é realizada a conexão com o banco de dados usando os parâmetros de conexão. Depois, é criado um objeto de cursor para exercutar comandos SQL no banco de dados. Então são definidos os nomes dos campos que serão utilizados para criar as VIEWS no banco de dados.

Com o **cursor.execute** são executados os comandos SQL para criar uma VIEW chamada **resultado_balanco** no banco de dados. Essa view contém os campos especificados na lista *campos* e os dados da matriz **matriz_balanco**

Um processo semelhante é realizado para a VIEW **ottobacias_isr**. Essa VIEW contém informações sobre a cobacia, geometria, ISR e classe de ISR, obtidas a partir da tabela *ottobacias_pb_5k* e da view *resultado_balanco*. Por fim, é feito um commit para que as alterações sejam salvas no banco de dados, o cursor é fechado e a conexão com o banco de dados é encerrada.

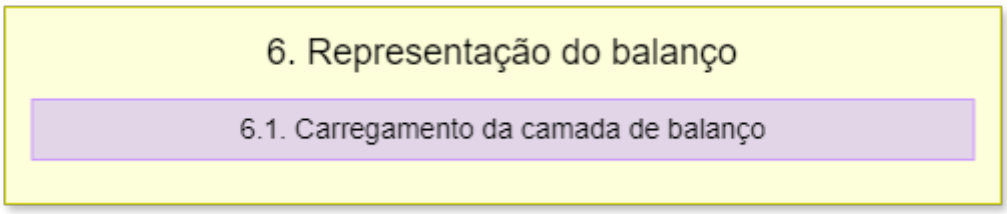
Os comandos utilizados SQL são:

- DROP VIEW IF EXISTS
- CREAT VIEW
- SELECT, FROM
- ORDER BY
- LEFT JOIN

Os comandos já foram descritos anteriormente nos itens 3 e 4.

1.6 6. Representação do balanço

O fluxograma de processos desta etapa é apresentado a seguir:



As funções **carregar_camada_balanco** e **importar_camada_balanco** são o mesmo processo das funções definidas na documentação da etapa 2, nos itens 2.2 e 2.3. Isso porque as etapas estão em arquivos diferentes, então para chamar as funções é necessário repeti-las.

A única diferença é que na função **carregar_camada_balanco** são definidas as configurações da simbologia da camada **ottobacias_isr** com base nos valores únicos do campo **classe_isr**. As cores dos símbolos são atribuídas com base em um dicionário **cores_classes** e os rótulos das categorias são definidos em um dicionário **rotulos_classes**. Por fim, é criado um renderizador de símbolos categorizado através do **QgsCategorizedSymbolRenderer**, o qual é atribuído à camada **ottobacias_isr**.

O método **triggerRepaint** é chamado na camada para garantir que as alterações de simbologia sejam aplicadas e então, a função retorna a camada ottobacias_isr.

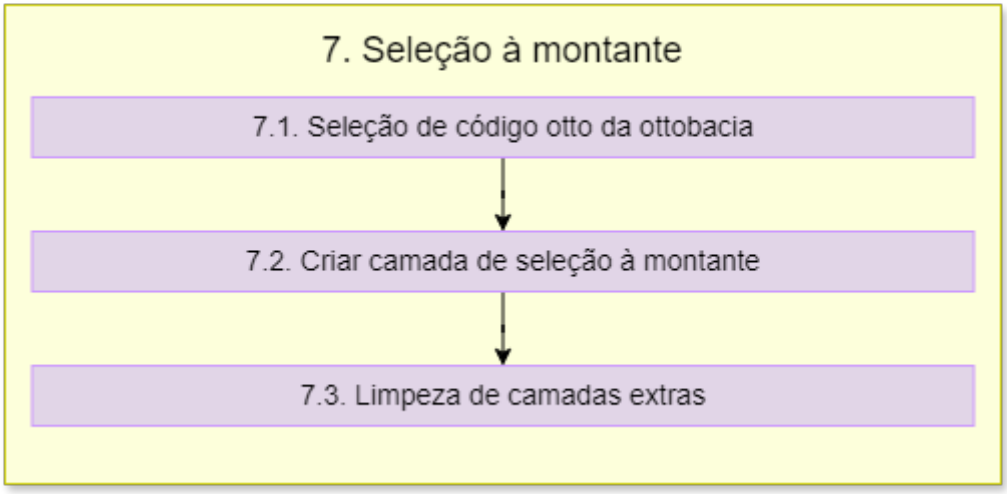
Nota:

- A classe **QgsCategorizedSymbolRenderer** é utilizada para criar um renderizador de símbolos categorizados.
- Através da classe **QgsRendererCategory** são definidas as categorias e os símbolos.

A função **limpeza_camadas_extras** realiza a remoção das camadas extras do projeto no QGIS.

1.7 7. Seleção da ottobacia

O fluxograma de processos desta etapa é apresentado a seguir:



Primeiramente é importado o módulo **psycopg2**.

1.7.1 7.1. Seleção de código otto da ottobacia

É definida uma classe chamada **PointTool** que é uma extensão especializada da funcionalidade de identificação de feições fornecida pelo QGIS. Essa ferramenta permite a captura de pontos no mapa clicando na tela do canvas. Ela herda de **QgsMapToolEmitPoint** que é uma classe base do QGIS.

Nota: A classe **QgsMapToolEmitPoint** é usada para criar ferramentas que respondem a eventos de cliques no mapa, emitindo a posição geográfica dos pontos clicados.

É definida uma função com o método **__init__** que é o construtor da classe **QgsMapToolEmitPoint** e é chamado quando uma instância da classe é criada.

A função com o método **canvasReleaseEvent** é chamada sempre que ocorre um "clique" com o mouse no canvas do mapa. A variável **disconnect_signal** verifica se a ferramenta está ativa para capturar um clique, se for *false*, a ferramenta processará o clique. Quando a variável **disconnect_signal** for *true* não será feita a captura de cliques.

A variável **point** converte a posição do clique para coordenadas de mapa. **lat_ponto** e **long_ponto** extraem a latitude e a longitude do ponto clicado, respectivamente.

Depois, é feita uma consulta espacial para encontrar a bacia hidrográfica do ponto selecionado. A variável **ottobacia_selecionada** vai criar uma camada vetorial virtual executando uma consulta SQL para selecionar a bacia (camada_ottobacias_isr) que contém o ponto clicado. A camada resultante é adicionada ao projeto QGIS, mas não é exibida na legenda. A variável **cod_otto_bacia** é definida como a variável global para armazenar o código da ottobacia.

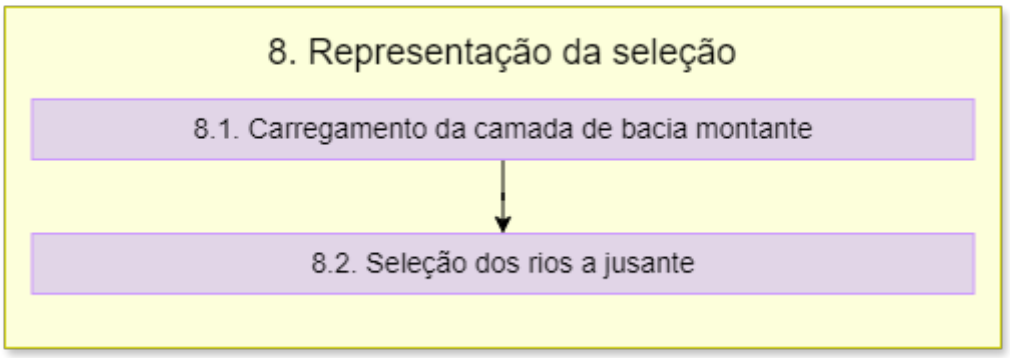
A variável **ponto_interesse** é estabelecida para fazer a adição do ponto de interesse ao mapa. Para isso, é criada uma camada vetorial virtual com o ponto de interesse e verificado se a camada já existe no projeto.

- Se a camada existe, obtém a referência para a camada existente e remove a mesma do projeto. A nova camada de ponto é adicionada ao projeto.
- Se a camada não existe é configurada uma conexão com o banco de dados PostgreSQL. Então é criada uma camada vetorial a partir de uma tabela do banco de dados (ottotrechos_pb_5k). E por fim, cria e ativa uma ferramenta (point_tool) de captura de pontos no canvas do mapa.

1.8 8. Representação da seleção

> **Aviso:** É necessário verificar a itemização dessa etapa.

O fluxograma de processos desta etapa é apresentado a seguir:



Primeiramente é importado o módulo **psycopg2**.

A função **carregar_camada_balanco** possui o mesmo processo da função definida na documentação da etapa 2, no item 2.3.

É feita a conexão com o banco de dados PostgreSQL usando o **psycopg2**.

1.8.1 8.1. Carregamento da camada de bacia montante

É feita uma iteração do **cod_otto_bacia** para obter o código de montante. Se o código otto for par, ele é usado diretamente. Se for ímpar, itera através dos dígitos do código, procurando o último dígito par.

A variável **ottobacias_isr_montante** cria uma camada vetorial virtual que seleciona bacias montantes com base no código otto obtido anteriormente. Caso a camada já exista, é removida e a nova camada é adicionada ao projeto.

A variável **bacia_montante** cria uma camada virtual a partir de comandos SQL para unir as geometrias das bacias montantes selecionadas.

1.8.2 8.2. Seleção dos rios a jusante

Primeiramente é feita a inicialização das variáveis:

- rio: uma lista vazia para armazenar os códigos otto dos rios a jusante;
- rios: um contador inicizalizado em 0;
- compri: uma variável inicializada em 4.

Para a seleção dos rios a jusante é realizado um loop sobre os índices do código otto da bacia. Para cada índice, o código é analisado. Se o último dígito do código for par, esse código é adicionado à lista rio. Se for ímpar , itera através dos dígitos do código, procurando o último dígito par.

Uma variável vazia chamada selecao é inicializada. Para cada código Otto na lista rio, é construída uma parte da cláusula WHERE da consulta SQL, que busca trechos de rio cujo código corresponde ao código otto do rio a jusante.

Utiliza-se a variável sele2 para filtrar os trechos de rio a jusante. É criada uma camada vetorial virtual (ottotrechos_jusante) que contém os trechos de rio a jusante selecionados pela consulta SQL.

Para consertar a parte do Rio Paraíba, então é necessário verificar se o ponto selecionado está na bacia 7588 ou na região abaixo ou acima dele. A condição é verificada e a seleção SQL é estendida de acordo com a situação.

Uma segunda camada vetorial virtual (ottotrechos_jusante_2) é criada, desta vez com base na seleção SQL estendida.

As camadas são carregadas no projeto QGIS com simbologias específicas, usando a função **carregar_camada**.

1.9 9. Códigos Auxiliares

Este diretório contém arquivos antigos, que foram descontinuados no processo de desenvolvimento do aplicativo principal, ou arquivos que foram criados para atividades específicas relacionadas ao projeto mas que não fazem parte do desenvolvimento do aplicativo principal.

1.9.1 9.1. Códigos Descontinuados

a) geral.py: Este arquivo é uma versão antiga que já foi o código principal do aplicativo em algum momento do desenvolvimento. Ele foi descontinuado, mas possui alguns códigos de processamentos que podem ser úteis.

Nota: Possui processamentos que executam o cálculo de população dentro de ottobacias utilizando código python (PyQGIS).

---

b) 7_SelecaoOttobacia.py: Este arquivo é uma versão antiga da etapa 7 do aplicativo, com outras ferramentas para seleção da ottobacia, utilizando o QgsMapToolIdentify para identificar feições e obter informações detalhadas da camada, enquanto a nova versão faz a seleção pela coordenada com outra funcionalidade.

c) 8_RepresentacaoSelecao.py: Este arquivo é uma versão antiga da etapa 8 do aplicativo. Nessa versão são criadas views no banco de dados, enquanto no código novo são utilizadas camadas virtuais. É uma versão mais robusta para a representação da seleção.

d) 9_JanelaInformacoes.py: Esse arquivo foi retirado do código principal pois não funcionava direito. Essa forma não é a mais adequada para fazer a janela de informações, portanto, optou-se pela retirada do código para posterior construção da janela em um plugin.

1.9.2 9.2. Comandos Úteis

comando_uteis.py: Neste arquivo são apresentados alguns comandos que possam ser utilizados em algum momento do desenvolvimento.

1.9.3 9.3. Códigos Extras

9.3.1. Seleção Montante Multipontos

Os arquivos deste diretório (SelecaoMontanteMultipontos) foram criados no contexto de uma demanda específica do William Cantos Corrêa, para análise de porcentagem de área de diferentes tipos de usos do solo para a região à montante de cada ponto de monitoramento de qualidade de água. Foram também selecionados pontos de ETAS e Outorgas à montante de cada ponto de monitoramento. Para cada tipo de seleção foi criado um arquivo python individual.

Nota: Para a execução destes códigos, faz-se necessário ter as camadas de entrada adicionadas no painel de camadas do QGIS.

Característica dos dados utilizados:

Os dados de uso do solo utilizados provém da coleção 8 do MapBiomas Brasil do ano de 2022. Eles foram vetorizados e reclassificados, de modo que algumas classe foram agrupadas. A reclassificação não interfere no processamentos de nenhum código python abaixo.

Breve descritivo dos códigos:

> **a) selecao_montante_uso_solo_geometria.py**: Este código faz a seleção das feições de uso do solo que estão contidas nas ottobacias à montante de cada ponto de monitoramento.

RESULTADO: Este código exporta as geometrias selecionadas no formato shapefile (.shp, .dbf, .shx).

> **b) selecao_montante_uso_solo_area.py**: Este código calcula a área das feições de uso do solo que estão contidas nas ottobacias à montante de cada ponto de monitoramento. O resultado é agrupado na soma do valor de área por tipo de uso.

RESULTADO: Este código exporta uma tabela no formato excel (.xlsx).

> **c) selecao_montante_etas.py**: Este código faz a seleção das ETAS, representadas por pontos, que estão contidas nas ottobacias à montante de cada ponto de monitoramento.

RESULTADO: Este código exporta uma tabela no formato excel (.xlsx) com o número total de ETAS contidas à montante de cada ponto de monitoramento.

> **d) selecao_montante_outorgas_usuarios.py**: Este código faz a seleção de pontos de outorgas que estão contidas nas ottobacias à montante de cada ponto de monitoramento.

RESULTADO: Este código exporta uma tabela no formato excel (.xlsx) com o número total de outorgas contidas à montante de cada ponto de monitoramento.

9.3.2. Seleção Montante e Dissolve

Nota: Esse código não funciona de forma independente.

O código **SelecaoMontanteDissolve.py** é uma alternativa para fazer o dissolve da seleção a montante sem precisar utilizar o banco de dados.

Basicamente o código seleciona as sub-bacias a montante com base em critérios específicos e consultas SQL. Realiza também um dissolve para combinar as sub-bacias selecionadas em uma única camada. Carrega e estiliza a nova camada resultante da dissolução, adicionando-a ao projeto QGIS.

1.10 10. Próximos Passos

Este diretório contém sugestões e atividades para os próximos do projeto.

1.10.1 10.1. Melhorias no Balanço Hídrico

a) Sofisticar: envolve refinar o código existente.

b) Otimizar cálculo: envolve aprimorar o desempenho do código, utilizando banco de dados em nuvem, além de otimizar os algoritmos existentes para melhorar a eficiência computacional.

c) Janela de informações: adicionar funcionalidade para apresentar uma janela com as informações de interesse.

d) Cores: ainda em questão de refinamento, abrir discussão sobre a escolha de cores para a visualização dos atributos.

e) Resultados: exportar os resultados, gerar relatórios dos dados de cenários, tabelas em CSV ou Excel, PDF.

f) ISR: ordenar o ISR do menos para o mais crítico.

1.10.2 10.2. Plugin

Desenvolvimento de um plugin a partir do código do balanço hídrico. Desenvolver uma interface, incluir funcionalidades adicionais que possam ser úteis.