Objetivos:

- Extrair subconjuntos de feições com base em critérios específicos;
- Operar feições com filtro.

I. Introdução à Classe QgsFeatureRequest no PyQGIS

Quando trabalhamos com dados vetoriais no QGIS, frequentemente precisamos acessar apenas um subconjunto específico de feições de uma camada — seja para filtrar registros por atributos, delimitar uma área de interesse ou otimizar a performance em operações com grandes volumes de dados.

É nesse contexto que se destaca a classe QgsFeatureRequest, uma ferramenta essencial no PyQGIS, que permite:

- Filtrar feições utilizando expressões similares ao SQL (ex.: `uf = 'SP'`);
- Selecionar apenas atributos ou geometrias relevantes, economizando memória e processamento;
- Definir regiões espaciais para consultas focadas (útil, por exemplo, para operações de zoom em mapas);
- Ordenar os resultados (ex.: nomes de municípios, em ordem crescente).

Comparação: Com vs Sem QgsFeatureRequest

Operação	Abordagem tradicional	Com QgsFeatureRequest
Filtro por atributo	Iterar todas feições e verificar manualmente	Filtro no banco de dados (mais rápido)
Recuperar geometria	Sempre carrega geometria	Pode desativar (setNoGeometry())
Campos necessários	Carrega todos atributos	Seleciona apenas campos específicos

II. Filtro por atributo

Na manipulação de dados geoespaciais, é comum a necessidade de selecionar apenas um subconjunto de feições com base em seus atributos. O PyQGIS oferece a classe `QgsFeatureRequest` para realizar essa filtragem de forma eficiente, evitando a necessidade de percorrer manualmente todas as feições de uma camada.

Nesta seção, aprenderemos a:

- Filtrar feições usando expressões SQL-like no PyQGIS;
- Utilizar corretamente as aspas em expressões de filtro;
- Recuperar atributos específicos das feições resultantes.

Exemplo 1: Listar os nomes dos municípios do estado de SP.

```
# Carrega a camada vetorial a partir de um GeoPackage
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')
```

```
# Cria um QgsFeatureRequest para filtrar apenas os municípios de SP
request = QgsFeatureRequest().setFilterExpression("uf = 'SP'")

# Itera sobre as feições filtradas e imprime o nome de cada município
for feicao in camada.getFeatures(request):
    print(feicao["municipio"])
```

Configuração do filtro

- `QgsFeatureRequest()` cria uma requisição de feições;
- `.setFilterExpression("uf = 'SP'")` define a expressão de filtro:
  - `uf` refere-se ao **nome do campo** na tabela de atributos;
  - `'SP'` (entre aspas simples) é o **valor** que estamos buscando. Não pode ser usado aspas duplas para envolver o valor que estamos comparando, ou seja, a instrução a seguir não funcionará:

```
request = QgsFeatureRequest().setFilterExpression('uf = "SP"')
```

Iteração sobre os resultados

- `camada.getFeatures(request)` retorna apenas as feições que atendem ao filtro (uf = 'SP');
- O loop `for` percorre cada feição e imprime o valor do campo `"municipio"`.

Regras para expressões de Filtro

- A expressão do filtro deve estar envolvida por aspas duplas;
- Valores textuais devem estar entre aspas simples (`'SP'`);

Exemplo 2: Listar os municípios do estado de SP cujo nome começa com JA.

Para filtrar municípios que pertencem ao estado de São Paulo (`uf = 'SP'`) **E** cujos nomes começam com `'JA'`, podemos combinar expressões SQL no `QgsFeatureRequest` usando o operador `AND` e o operador `ILIKE` para comparação parcial de texto.

```
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')

# Cria um QgsFeatureRequest para filtrar apenas os municípios de SP
# Filtro composto: UF = SP E nome começa com JA
request = QgsFeatureRequest().setFilterExpression(
    "uf = 'SP' AND municipio ILIKE 'JA%'"
)

# Itera sobre as feições filtradas e imprime o nome de cada município
for feicao in camada.getFeatures(request):
    print(feicao["municipio"])
```

Explicação:

- `uf = 'SP'` filtra apenas registros de São Paulo;
- `municipio ILIKE 'JA%'` filtra nomes que começam com `'JA'` (o símbolo `%` é um curinga em SQL);

- AND operador lógico que combina as duas condições.

III. Filtro espacial

Os filtros espaciais permitem selecionar feições com base em sua localização geográfica e relação com outras geometrias. A classe `QgsFeatureRequest` oferece métodos eficientes para esse tipo de filtragem, sendo especialmente útil para:

- Selecionar feições dentro de uma área de interesse;
- Identificar elementos que intersectam outras geometrias;
- Otimizar o desempenho em operações com grandes conjuntos de dados.

Exemplo 3: Listar municípios que intersectam com o retângulo formado pelas coordenadas (`-46.193, -23.518`) e (`-45.766, -22.936`).

Para filtrar as geometrias que fazem interseção com o retângulo usamos o método `setFilterRect`.

```
# Carrega a camada de municípios
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')

# Define os pontos do retângulo de interesse
ponto_inferior_esquerdo = QgsPointXY(-46.193, -23.518)
ponto_superior_direito = QgsPointXY(-45.766, -22.936)
retangulo = QgsRectangle(ponto_inferior_esquerdo, ponto_superior_direito)

# Cria o filtro espacial para filtrar por interseção
request = QgsFeatureRequest().setFilterRect(retangulo)

# Itera sobre as feições filtradas e imprime o nome de cada município
for feicao in camada.getFeatures(request):
    print(feicao["municipio"])
```

IV. Combinar filtros de atributo e espacial

Nesta seção, apresentamos como realizar consultas combinadas no PyQGIS, aplicando simultaneamente filtros espaciais e filtros por atributos com a classe `QgsFeatureRequest`. Esse recurso é fundamental para refinar seleções em bases de dados espaciais, assegurando análises mais precisas e eficientes.

Exemplo 4: Municípios que intersectam um retângulo e cujo nome começa com 'J'.

```
# Carrega a camada de municípios
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')

# Define os pontos do retângulo de interesse
ponto_inferior_esquerdo = QgsPointXY(-46.193, -23.518)  # Longitude, Latitude
ponto_superior_direito = QgsPointXY(-45.766, -22.936)
retangulo = QgsRectangle(ponto_inferior_esquerdo, ponto_superior_direito)
```

```
# Cria o filtro espacial e de atributo
request = QgsFeatureRequest().setFilterRect(retangulo).setFilterExpression(
    "municipio ILIKE 'J%'"
)

# Processa as feições resultantes
for feicao in camada.getFeatures(request):
    print(feicao["municipio"])
```

V. Ordenação

A ordenação é fundamental quando desejamos organizar os resultados de consultas espaciais e não espaciais, permitindo que os dados sejam apresentados de forma sistemática, facilitando a interpretação e a análise.

Exemplo 5: Ilustra como listar os nomes dos municípios do estado de São Paulo (SP), ordenando-os inicialmente pela mesorregião (campo mesoreg) e, dentro de cada mesorregião, pelo nome do município (campo municipio), ambos em ordem crescente (ascendente).

```
# Carrega a camada vetorial a partir de um GeoPackage
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')

# Cria um QgsFeatureRequest para filtrar apenas os municípios de SP
# e ordenar por mesorregião (ascendente) e nome do município (ascendente)
request = QgsFeatureRequest()\
    .setFilterExpression("uf = 'SP'")\
    .addOrderBy("mesoreg", ascending=True)\
    .addOrderBy("municipio", ascending=True)

# Itera sobre as feições filtradas e imprime o nome de cada município
for feicao in camada.getFeatures(request):
    print(f"{feicao['mesoreg']} - {feicao['municipio']}")
```

VI. Limitar a quantidade de feições no resultado

Em diversas aplicações no processamento de dados espaciais, é importante restringir a quantidade de resultados retornados por uma consulta, especialmente para otimizar o desempenho, evitar sobrecarga de processamento ou para visualizar apenas uma amostra dos dados.

O método `setLimit()` da classe `QgsFeatureRequest`, permite especificar o número máximo de feições (registros) que devem ser retornadas pelo provedor de dados. Esta funcionalidade é análoga à cláusula LIMIT utilizada em consultas SQL, sendo útil para análises exploratórias ou para apresentar apenas os primeiros registros de um conjunto de dados ordenado.

Exemplo 6: Ilustra como listar apenas os 10 primeiros registros de uma consulta.

```
# Carrega a camada vetorial a partir de um GeoPackage
camada = QgsVectorLayer('D:/pasta/municipio.gpkg', 'mun', 'ogr')
```

```
# Cria um QgsFeatureRequest para filtrar apenas os municípios de SP
# e ordenar por mesorregião (ascendente) e nome do município (ascendente)
# e apresetnar apenas os 10 primeiros registros
request = QgsFeatureRequest()\
    .setFilterExpression("uf = 'SP'") \
    .addOrderBy("mesoreg", ascending=True) \
    .addOrderBy("municipio", ascending=True) \
    .setLimit(10)

# Itera sobre as feições filtradas e imprime o nome de cada município
for feicao in camada.getFeatures(request):
    print(f"{feicao['mesoreg']} - {feicao['municipio']}")
```

Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercícios 1 e 2: https://youtu.be/d05e7OuY2dc

Exercícios 3 a 5: https://youtu.be/U_UfTbPkKeM

Exercício 1: Alterar o Exemplo 3 para listar os nomes dos municípios que estão totalmente contidos no retângulo formado pelas coordenadas `(-46.193, -23.518)` e `(-45.766, -22.936)`.

Dicas:

- Converte o retângulo para geometria. O objeto `QgsGeometry` possui o método `contains`;

```
retangulo_geom = QgsGeometry.fromRect(retangulo)
```

- Para cada feição filtrada, verifique se a feição está contida no retângulo:

```
if retangulo_geom.contains(feicao.geometry()):
    print(feicao["municipio"])
```

Por que essa abordagem?

- `setFilterRect()` reduz o número de feições a serem verificadas (usa índice espacial);
- `contains()` garante a precisão da relação espacial.

Resultado esperado: apenas Jacareí.

Exercício 2: Fazer um programa para listar os nomes dos municípios que interceptam o Retângulo Envolvente Mínimo (REM) do polígono correspondente ao município de Jacareí.

Passos principais:

1. Carregar a camada de municípios normalmente utilizando `QgsVectorLayer`;

2. Criar um `QgsFeatureRequest` para filtrar pela feição cujo atributo `municipio` seja igual a `'Jacareí'`, utilizando o método:

```
.setFilterExpression("municipio = 'Jacareí'");
```

3. Iterar sobre as feições filtradas, obter a geometria correspondente a Jacareí e armazená-la em uma variável;
4. Obter o REM da geometria de Jacareí utilizando o método:

```
.boundingBox()
```

5. Criar um novo `QgsFeatureRequest` para realizar o filtro espacial, utilizando o retângulo obtido:

```
.setFilterRect(retangulo)
```

6. Iterar sobre as feições que satisfazem o filtro espacial e imprimir no console o nome (`municipio`) de cada feição que intersecta o REM de Jacareí.

Resultado esperado:

Moji das Cruzes
Guararema
Santa Branca
Igaratá
Santa Isabel
Jacareí
São José dos Campos
Jambeiro

Exercício 3: Altere o código do exercício anterior para filtrar apenas as geometrias que **tocam** Jacareí, ou seja, que compartilham uma borda ou um ponto com a geometria, mas não a intersectam internamente.

Principais alterações:

- Incluir a seguinte alteração no o Passo 6 do exercício anterior: Adicionar uma estrutura de decisão `if` para verificar se as geometrias se tocam, usando o método `touches`.

```
if geom.touches(geom_jacarei):
```

Resultado esperado:

Guararema
Santa Branca
Igaratá
Santa Isabel
São José dos Campos
Jambeiro

Exercício 4: Altere o código do exercício anterior para exibir o resultado ordenado pelo nome do munícipio.

Alteração necessária:

- Incluir no `QgsFeatureRequest` o `addOrderBy()` após o `.setFilterRect()`.

Resultado esperado:

Guararema
Igaratá
Jambeiro
Santa Branca
Santa Isabel
São José dos Campos

Exercício 5: Altere o código do exercício anterior para listar somente os municípios que possuem área menor que Jacareí.

Alterações necessárias:

- Use o método `.area()` para calcular a área do polígono de Jacareí e guardar em uma variável;
- Antes de imprimir no console o nome do município, inclua uma estrutura de decisão `if` para checar a área do município:

```
# Calcula a área da geometria
area = geom.area()

# Aplica o filtro de área
if area < area_jacarei:
    print(f"{feicao['municipio']} - Área: {area} unidades²")
```

Resultado esperado:

Guararema - Área: 0.023894307302891278 unidades²

Igaratá - Área: 0.02587731857201525 unidades²

Jambeiro - Área: 0.016217495562706866 unidades²

Santa Branca - Área: 0.024308514060792746 unidades²

Santa Isabel - Área: 0.031910136397348465 unidades²