

# 1 Introdução

Neste material, estudaremos sobre a **cursors**. Veja a sua documentação oficial no Link 1.1.

Link 1.1

<https://www.postgresql.org/docs/current/plpgsql-cursors.html>

Um cursor é **uma estrutura que encapsula uma consulta e viabiliza a leitura do resultado linha a linha**.

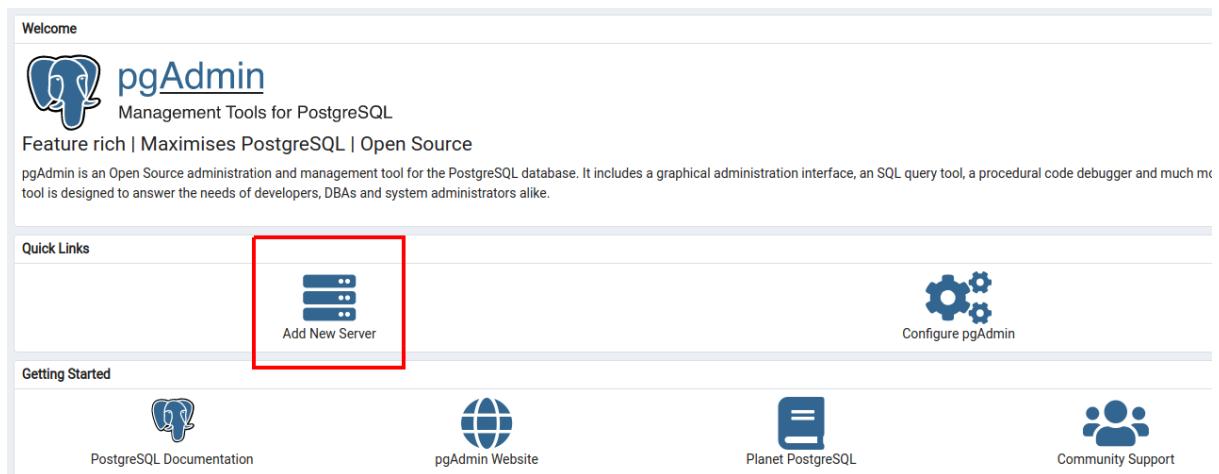
Casos de uso típicos são

- Processamento de consultas com potencial de trazer coleções muito grandes de dados, as quais podem demandar mais memória do que há disponível.
- Criação de funções que devolvem uma referência a um cursor, o que pode ser útil para devolver ao código cliente grandes coleções de tuplas.

## 2 Passo a passo

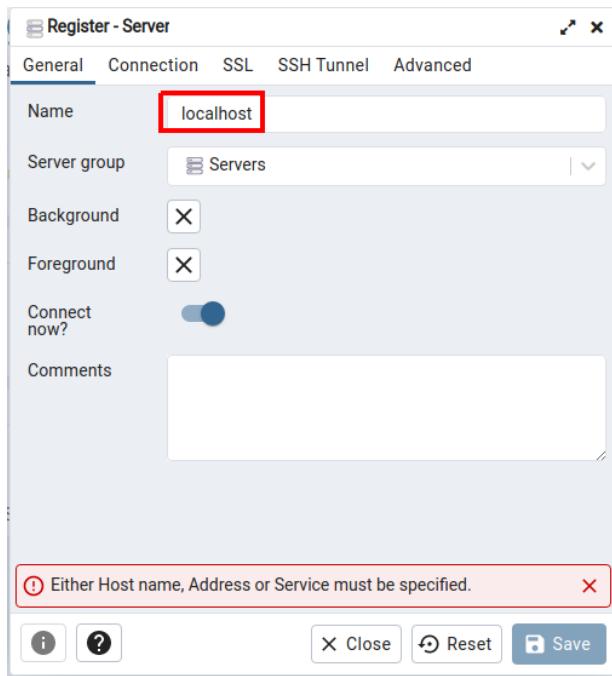
**2.1 (Criando um servidor)** Caso ainda não possua um servidor, abra o pgAdmin4 e clique em **Add New Server**, como mostra a Figura 2.1.1.

Figura 2.1.1



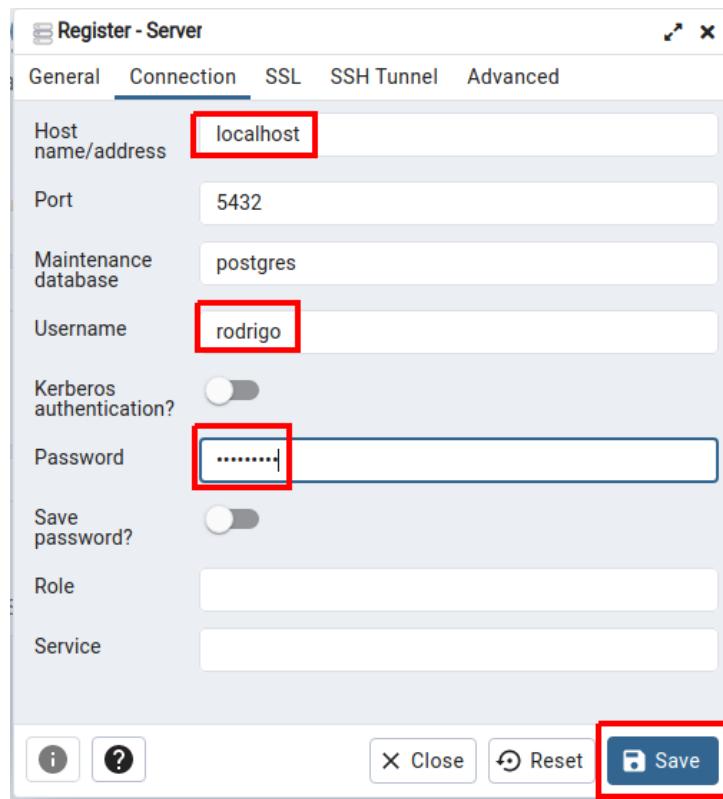
O nome do servidor pode ser algo que lhe ajude a lembrar a razão de ser dele. Como é um servidor que está executando localmente, podemos chamá-lo de algo como **localhost**, como na Figura 2.1.2. Depois de preencher o nome, clique na aba **Connection**.

Figura 2.1.2



Agora clique na aba **Connection**, como na Figura 2.1.3. Preencha os campos como destacado e clique em **Save**.

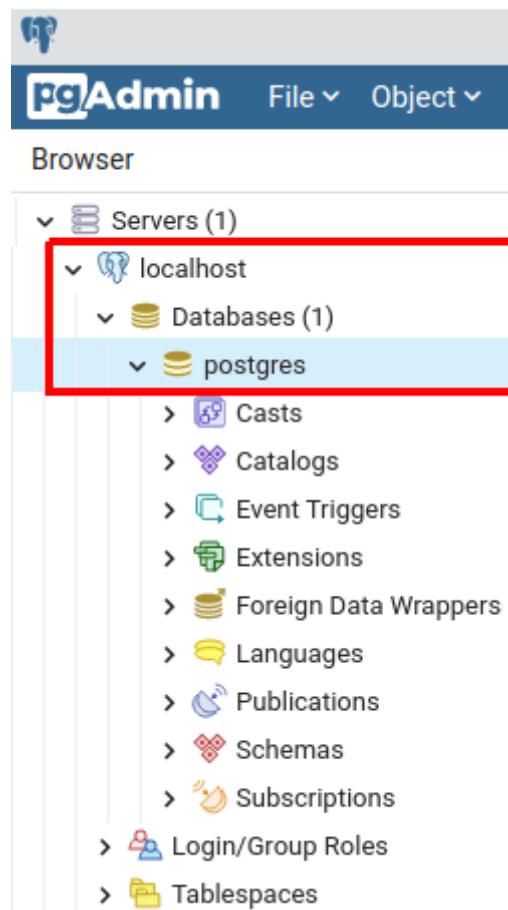
Figura 2.1.3



**Nota.** Usuário e senha dependerão de suas configurações. No Windows, é comum a existência de um usuário chamado postgres com a senha também igual a postgres.

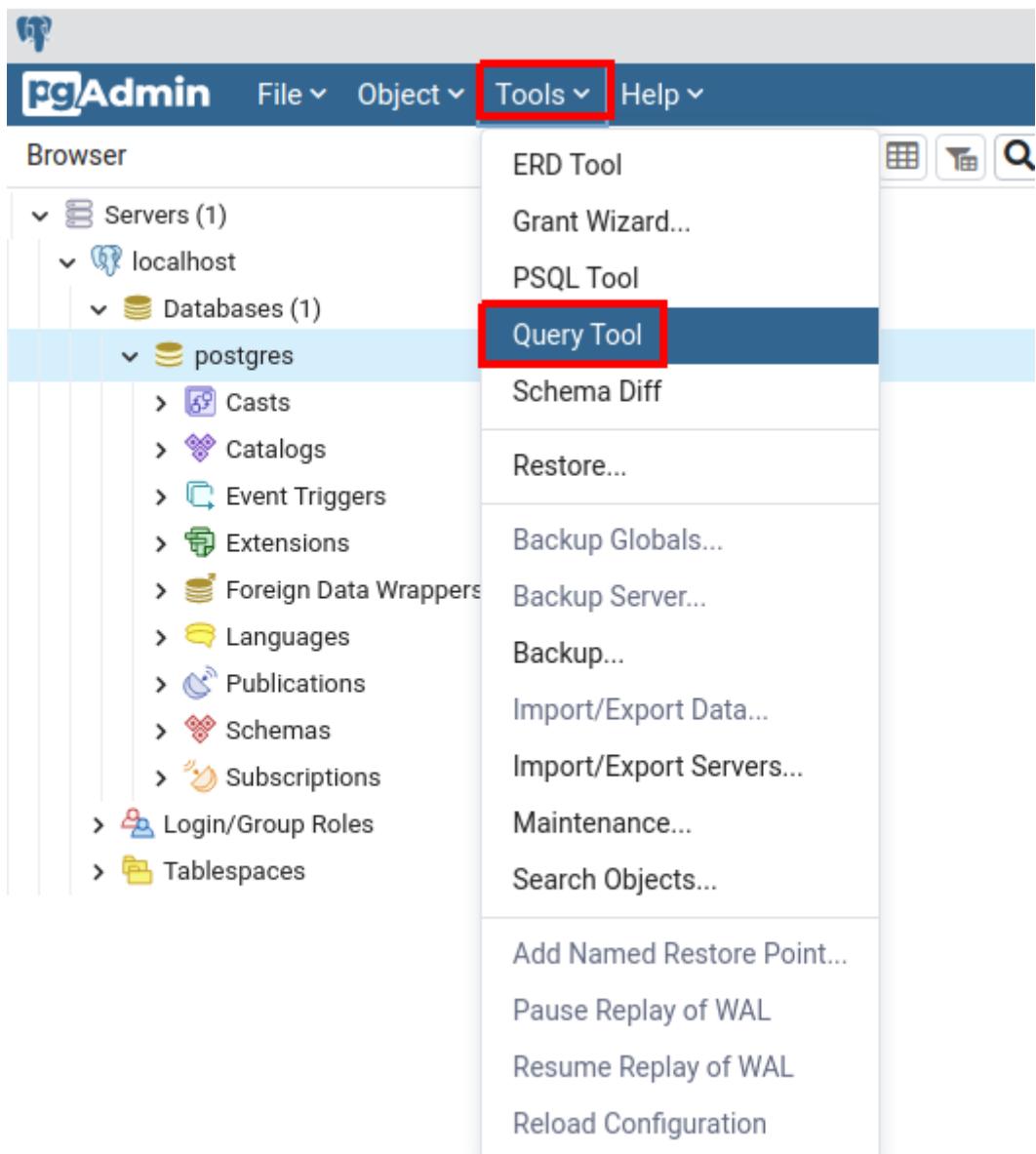
No canto superior esquerdo, encontre o seu servidor e clique sobre ele. Expanda **Databases** e encontre o database chamado **postgres**, cuja existência é muito comum. Veja a Figura 2.1.4.

Figura 2.1.4



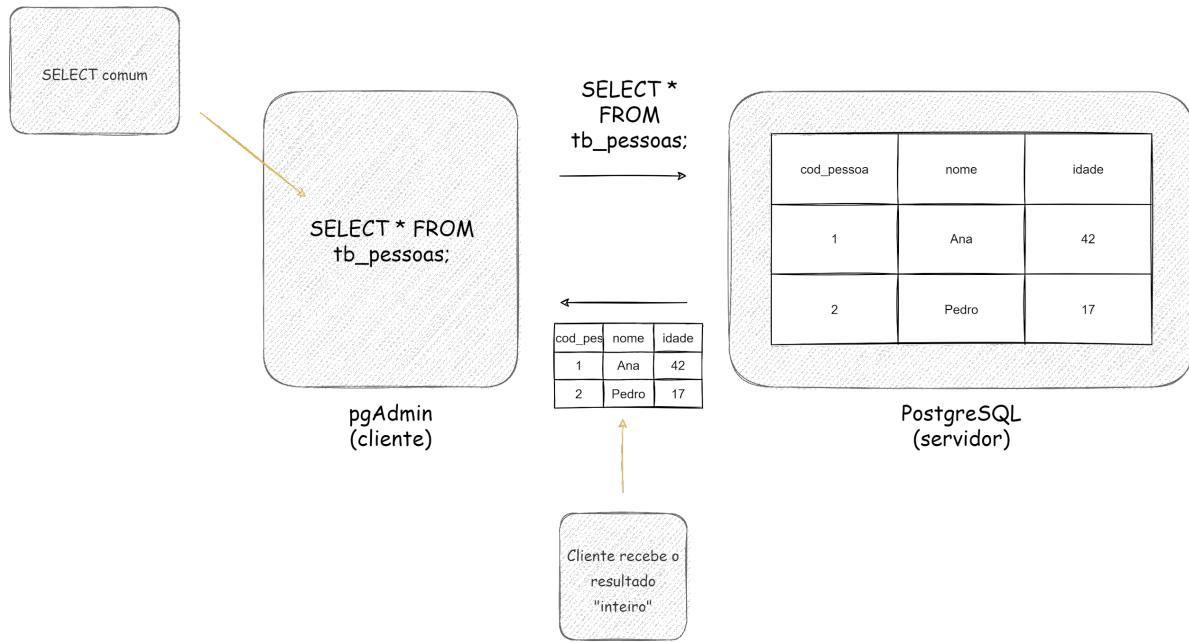
Para abrir um editor em que possa digitar seus comandos SQL, clique em **Tools > Query Tool**, como mostra a Figura 2.1.5.

Figura 2.1.5



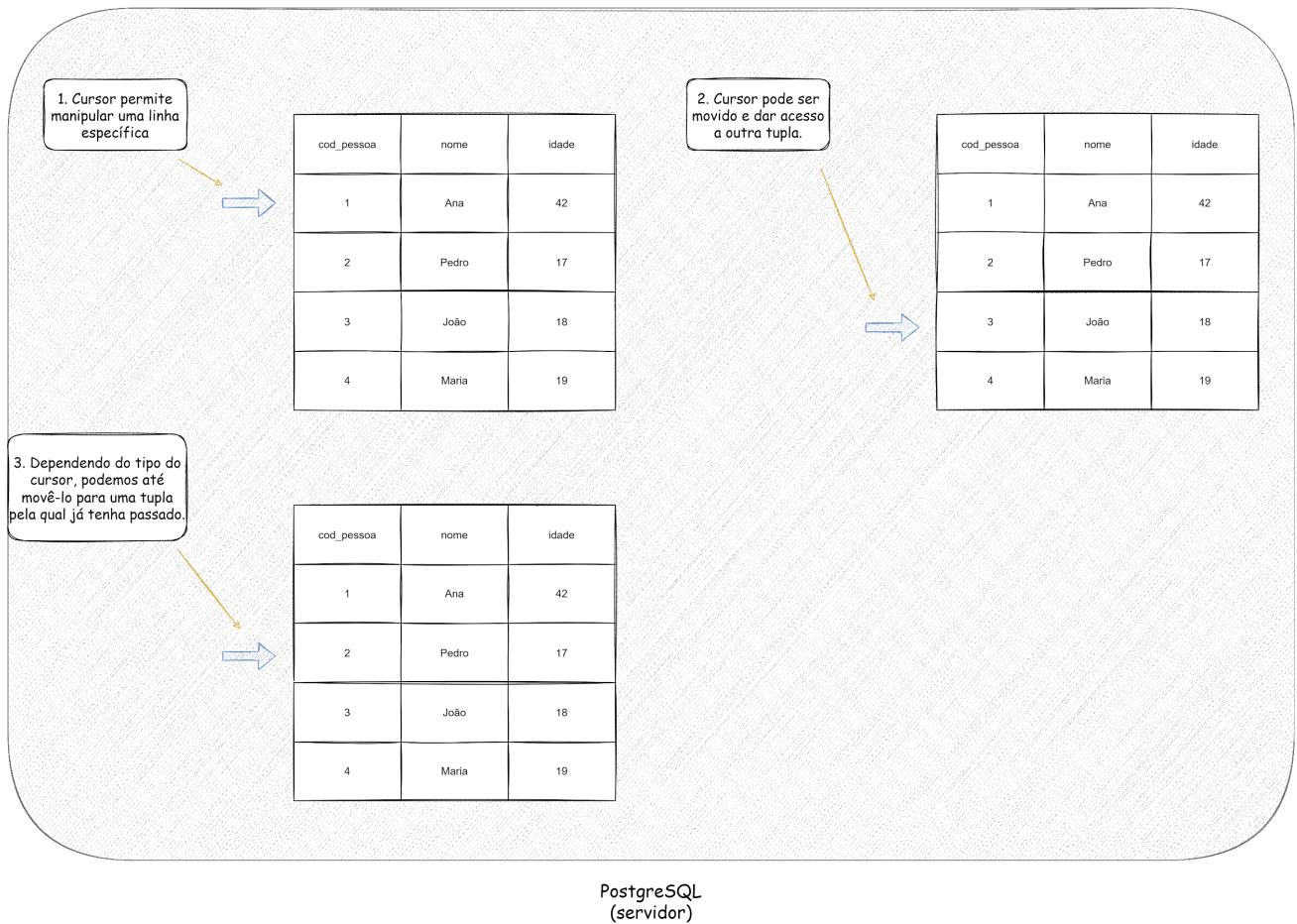
**2.2 (SELECTs regulares e cursores)** Quando executamos um comando SELECT regular, o resultado completo da consulta nos é entregue, como mostra a Figura 2.2.1. Utilizando SQL padrão, não temos como percorrer o resultado linha a linha para aplicar um processamento arbitrário.

Figura 2.2.1



Quando executamos um comando SELECT encapsulado por um cursor, temos a estrutura necessária para que percorrer o resultado linha a linha seja possível. Veja a Figura 2.2.2.

Figura 2.2.2



**2.3 (O que é preciso fazer para usar um cursor)** O uso de um cursor se dá por meio dos seguintes passos

- **Declaração:** O primeiro passo é declarar o cursor. Ele é sempre do tipo **refcursor**. A especificação do comando **SELECT** **pode** ser feita agora.
- **Abertura:** O segundo passo é abrir o cursor. Aqui também é possível especificar o **SELECT** a que o cursor estará vinculado.
- **Recuperação de dados:** O terceiro passo é a manipulação dos dados. Aqui podemos usar comandos como **FETCH** para obter a linha atual e **MOVE** para mover o cursor para uma linha desejada.
- **Fechamento:** Todo cursor deve ser fechado a fim de que recursos alocados sejam liberados.

**2.4 (Base de dados para nossos testes)** Para realizar testes utilizando cursores, vamos utilizar a base de dados disponível no Link 2.4.1.

Link 2.4.1

<https://www.kaggle.com/datasets/surajjha101/top-youtube-channels-data>

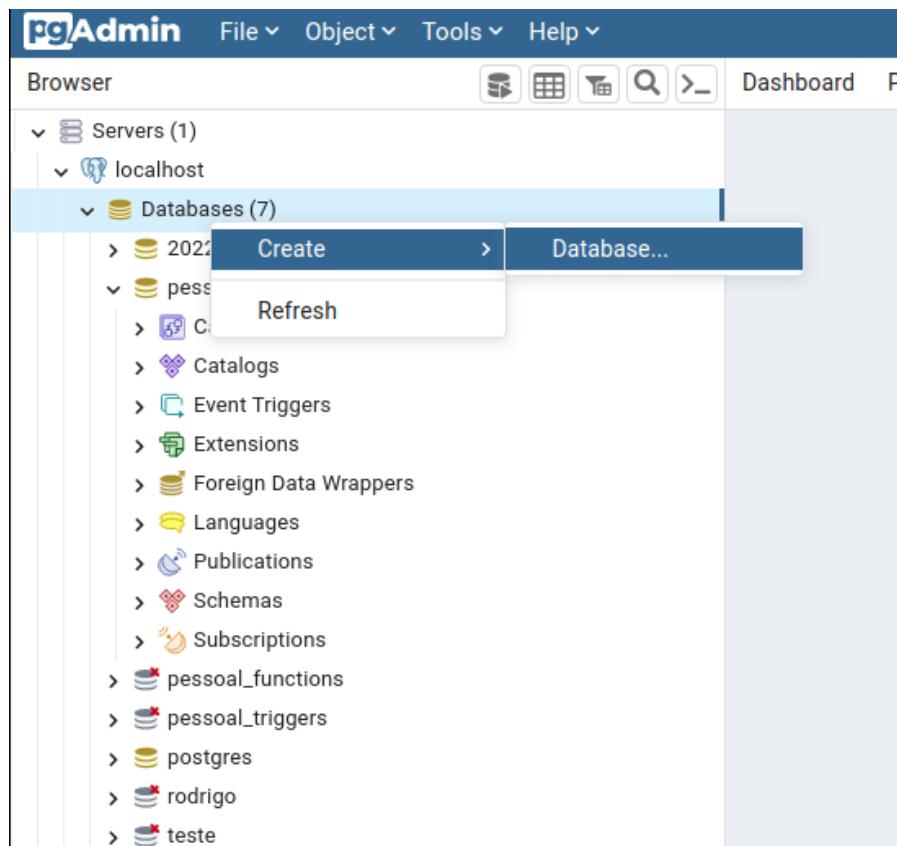
Trata-se de uma base de dados que contém informações sobre os canais dos principais "youtubers". As variáveis são as seguintes:

- rank: O ranking do canal em função do número de inscritos.
- youtuber: nome do youtuber dono do canal.
- subscribers: número de inscritos
- video views: número de visualizações de todos os vídeos do canal
- video count: número de vídeos do canal
- category: categoria do canal
- started: ano em que o canal começou as atividades

Faça o download da base de dados e descompacte para ter acesso ao arquivo CSV.

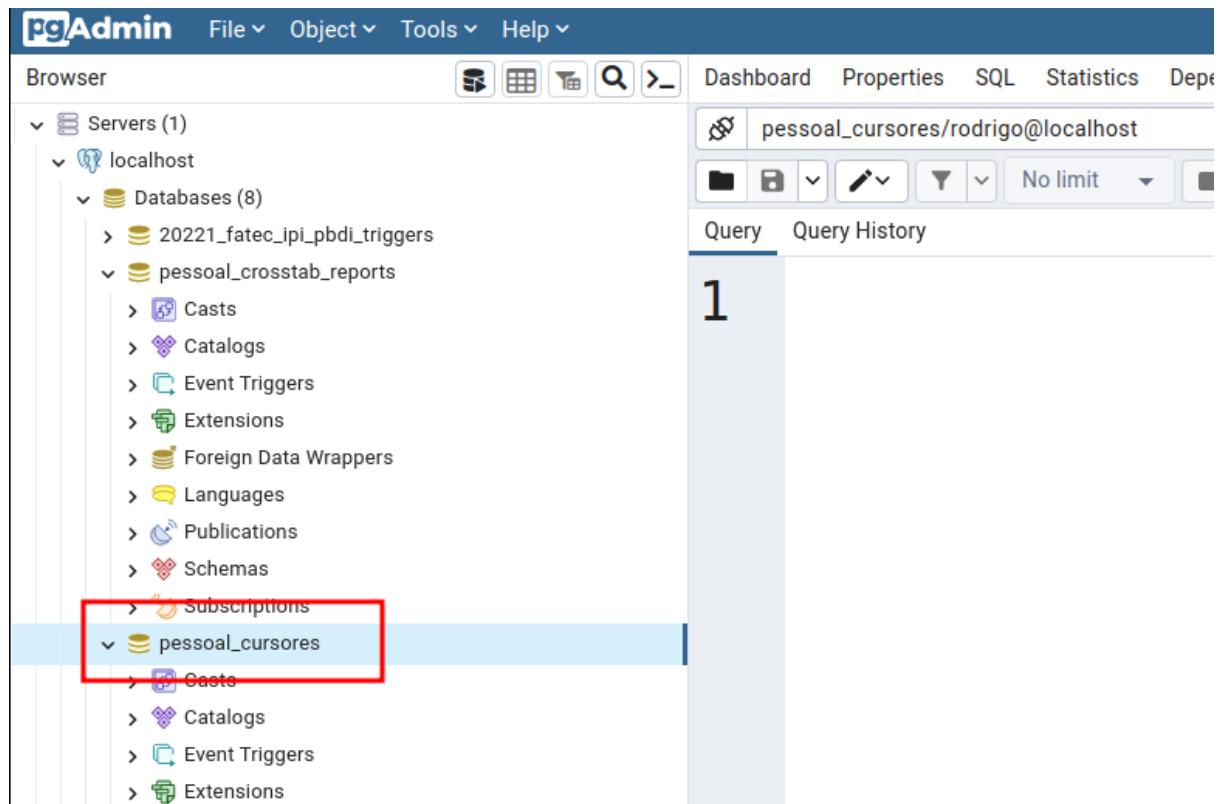
**(Nova base de dados no Postgre)** No pgAdmin, crie um database como na Figura 2.4.1.

Figura 2.4.1



Escolha um nome apropriado para a base. Ainda no pgAdmin, clique em **Tools >> Query Tool** para ter acesso ao editor. Lembre-se de manter a base de dados criada há pouco selecionada, como na Figura 2.4.2.

Figura 2.4.2



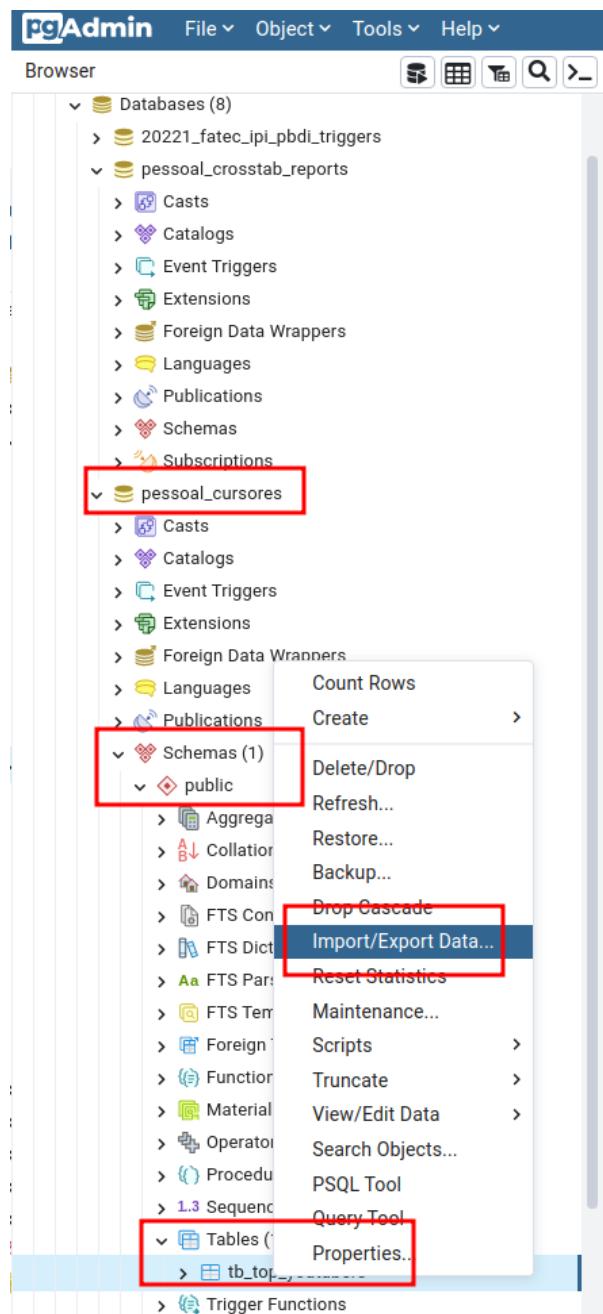
**(Criando uma tabela)** O Bloco de Código 2.4.1 mostra como criar uma tabela apropriada para abrigar os dados.

Bloco de Código 2.4.1

```
CREATE TABLE tb_top_youtubers(
    cod_top_youtubers SERIAL PRIMARY KEY,
    rank INT,
    youtuber VARCHAR(200),
    subscribers INT,
    video_views VARCHAR(200),
    video_count INT,
    category VARCHAR(200),
    started INT
);
```

**(Importando os dados)** Há diferentes formas para importar os dados de um arquivo .csv para uma base gerenciada pelo PostgreSQL. Uma delas é provida pelo próprio pgAdmin. Para utilizar esta funcionalidade, clique com o direito na tabela que acaba de criar (se necessário, clique com o direito em Tables e escolha Refresh para encontrá-la) e escolha a opção Import/Export Data.... Veja a Figura 2.4.3.

Figura 2.4.3

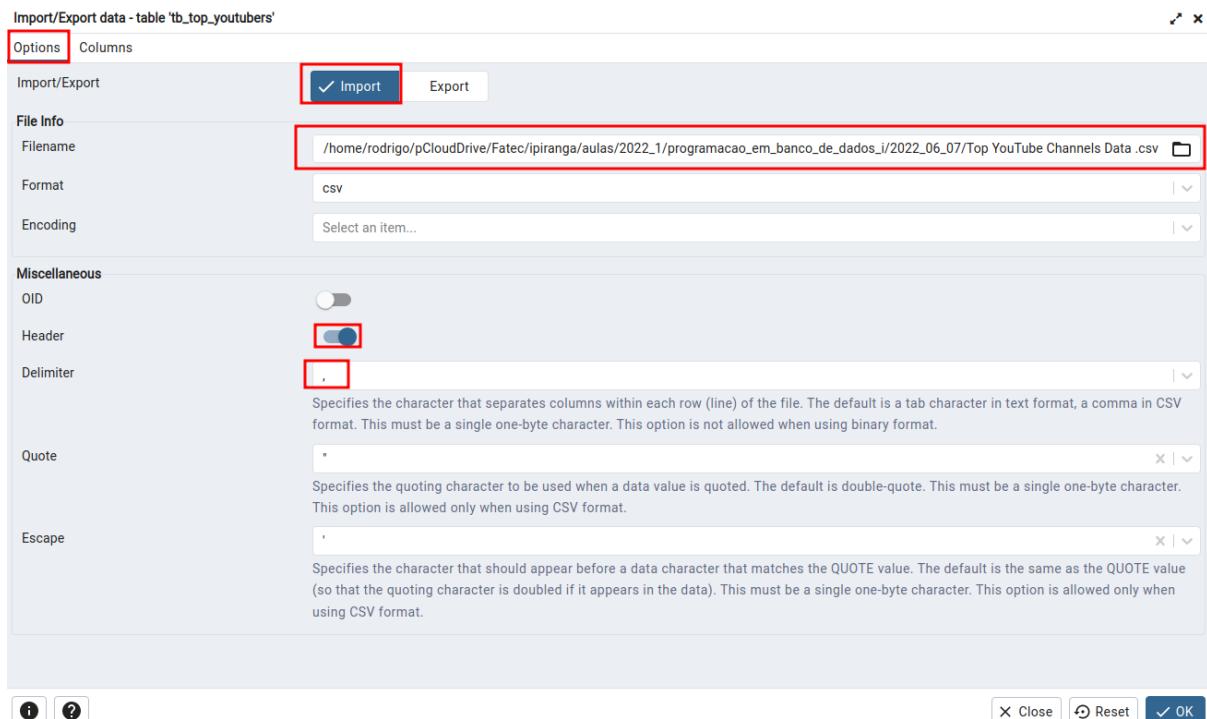


Na tela seguinte, escolha

- **Import/Export:** Import
- **Filename:** clique na pastinha e navegue até o diretório em que se encontra seu arquivo
- **Header:** Clique para informar que o arquivo possui cabeçalho
- **Delimiter:** ,

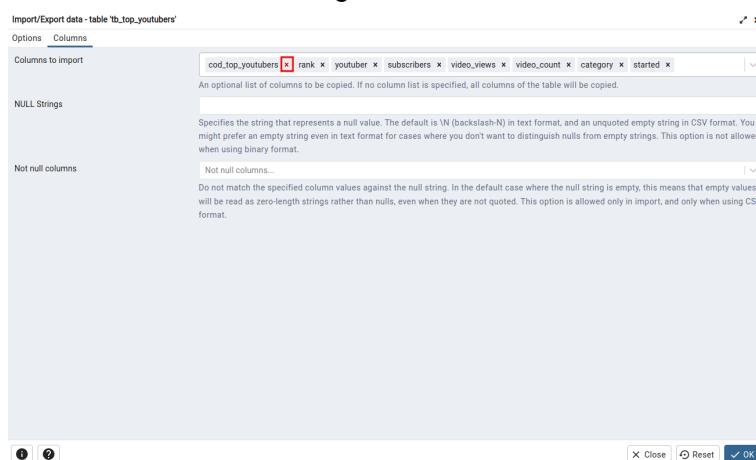
Veja a Figura 2.4.3.

Figura 2.4.3



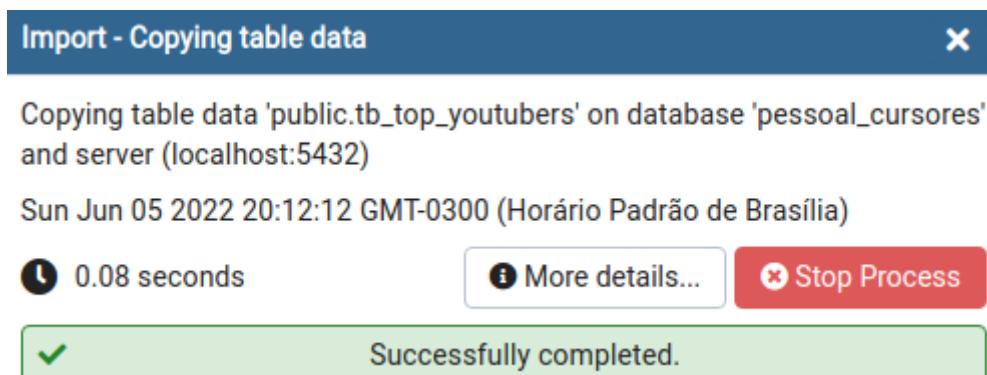
Ainda nesta tela, clique na aba **Columns**. Como mostra a Figura 2.4.4. clique no **x** próximo à coluna `cod_top_youtubers` para removê-la. Seu valor não será importado do CSV: ele será gerado automaticamente pelo PostgreSQL.

Figura 2.4.4



Clique em **OK**. A esperança é obter uma mensagem informativa como aquela que a Figura 2.4.5 exibe.

Figura 2.4.5



Apenas clique no **X** para fechá-la. Execute também um SELECT para verificar os dados.

**2.5 (Cursor não vinculado (unbound): Exibindo os nomes dos Youtubers)** Nesta seção vamos criar um Cursor para fazer a exibição dos nomes dos Youtubers. Dizemos que o Cursor em questão é “não vinculado” por não estar associado a nenhuma query. Veja o Bloco de Código 2.5.1.

**Nota.** Estamos fazendo uso de uma variável especial chamada **FOUND**. Leia mais sobre ela no Link 2.5.1.

Link 2.5.1

<https://www.postgresql.org/docs/current/plpgsql-statements.html>

A Tabela 2.5.1 mostra um resumo sobre a variável FOUND, trazido da documentação oficial.

Tabela 2.5.1

1. A SELECT INTO statement sets FOUND true if a row is assigned, false if no row is returned.
2. A PERFORM statement sets FOUND true if it produces (and discards) one or more rows, false if no row is produced.
3. UPDATE, INSERT, and DELETE statements set FOUND true if at least one row is affected, false if no row is affected.
4. A FETCH statement sets FOUND true if it returns a row, false if no row is returned.
5. A MOVE statement sets FOUND true if it successfully repositions the cursor, false otherwise.
6. A FOR or FOREACH statement sets FOUND true if it iterates one or more times, else false. FOUND is set this way when the loop exits; inside the execution of the loop, FOUND is not modified by the loop statement, although it might be changed by the execution of other statements within the loop body.
7. RETURN QUERY and RETURN QUERY EXECUTE statements set FOUND true if the query returns at least one row, false if no row is returned.
8. Other PL/pgSQL statements do not change the state of FOUND. Note in particular that EXECUTE changes the output of GET DIAGNOSTICS, but does not change FOUND.
9. FOUND is a local variable within each PL/pgSQL function; any changes to it affect only the current function.

### Bloco de Código 2.5.1

```
DO $$  
DECLARE  
    -1. declaração do cursor  
    -esse cursor é unbound por não ser associado a nenhuma query  
    cur_nomes_youtubers REFCURSOR;  
    -para armazenar o nome do youtuber a cada iteração  
    v_youtuber VARCHAR(200);  
BEGIN  
    -2. abertura do cursor  
    OPEN cur_nomes_youtubers FOR  
        SELECT youtuber  
        FROM  
        tb_top_youtubers;  
  
    LOOP  
        -3. Recuperação dos dados de interesse  
        FETCH cur_nomes_youtubers INTO v_youtuber;  
        --FOUND é uma variável especial que indica  
        EXIT WHEN NOT FOUND;  
        RAISE NOTICE '%', v_youtuber;  
  
    END LOOP;  
    -4. Fechamento do cursos  
    CLOSE cur_nomes_youtubers;  
END;  
$$
```

**2.6 (Cursor não vinculado (unbound) com query dinâmica: Exibindo os nomes dos youtubers que começaram a partir de um ano específico)** Vejamos como criar um cursor capaz de operar com uma query qualquer, especificada como uma string.

## Bloco de Código 2.6.1

```
DO $$  
DECLARE  
    cur_nomes_a_partir_de REFCURSOR;  
    v_youtuber VARCHAR(200);  
    v_ano INT := 2008;  
    v_nome_tabela VARCHAR(200) := 'tb_top_youtubers';  
BEGIN  
    OPEN cur_nomes_a_partir_de FOR EXECUTE  
        format  
        (  
            SELECT  
                youtuber  
            FROM  
                %s  
            WHERE started >= $1  
            '  
            ,  
            v_nome_tabela  
        )USING v_ano;  
    LOOP  
        FETCH cur_nomes_a_partir_de INTO v_youtuber;  
        EXIT WHEN NOT FOUND;  
        RAISE NOTICE '%', v_youtuber;  
    END LOOP;  
    CLOSE cur_nomes_a_partir_de;  
END;  
$$
```

**2.7 (Cursor vinculado (bound): Concatenando nome e número de inscritos)** Um Cursor é vinculado ou bound quando, no momento de sua declaração, já especificamos a query a que ficará associado. Neste exemplo, vamos montar uma string contendo os nomes e números de inscritos de cada canal. Veja o Bloco de Código 2.7.1.

### Bloco de Código 2.7.1

```
DO $$  
DECLARE  
    -cursor vinculado (bound)  
    cur_nomes_e_inscritos CURSOR FOR SELECT youtuber, subscribers FROM  
    tb_top_youtubers;  
    -capaz de abrigar uma tupla inteira  
    -tupla.youtuber nos dá o nome do youtuber  
    -tupla.subscribers nos dá o número de inscritos  
    tupla RECORD;  
    resultado TEXT DEFAULT '';  
BEGIN  
    OPEN cur_nomes_e_inscritos;  
    FETCH cur_nomes_e_inscritos INTO tupla;  
    WHILE FOUND LOOP  
        resultado := resultado || tupla.youtuber || ':' || tupla.subscribers || ' ';  
        FETCH cur_nomes_e_inscritos INTO tupla;  
    END LOOP;  
    CLOSE cur_nomes_e_inscritos;  
    RAISE NOTICE '%', resultado;  
END;  
  
$$
```

**2.8 (Cursor: Parâmetros nomeados e pela ordem)** Um cursor pode receber parâmetros. Eles podem ser especificados por ordem e também por nome. No Bloco de Código 2.8.1 exibimos os nomes dos youtubers que começaram a partir de 2010 e que têm, pelo menos, 60 milhões de inscritos. Ilustramos as duas formas de passagem de parâmetro.

## Bloco de Código 2.8.1

```
DO $$  
DECLARE  
    v_ano INT := 2010;  
    v_inscritos INT := 60_000_000;  
    cur_ano_inscritos CURSOR (ano INT, inscritos INT) FOR SELECT youtuber FROM  
    tb_top_youtubers WHERE started >= ano AND subscribers >= inscritos;  
    v_youtuber VARCHAR(200);  
BEGIN  
    --execute apenas um dos dois comandos OPEN a seguir  
    -- passando argumentos pela ordem  
    OPEN cur_ano_inscritos (v_ano, v_inscritos);  
    --passando argumentos por nome  
    OPEN cur_ano_inscritos (inscritos := v_inscritos, ano := v_ano);  
    LOOP  
        FETCH cur_ano_inscritos INTO v_youtuber;  
        EXIT WHEN NOT FOUND;  
        RAISE NOTICE '%', v_youtuber;  
    END LOOP;  
    CLOSE cur_ano_inscritos;  
END;  
$$
```

**2.9 (Cursor: UPDATE e DELETE)** O processamento realizado por meio de um cursor pode envolver operações UPDATE e DELETE. No Bloco de Código 2.9.1 ilustramos um cursor em que

- remove todas as duplas em que video\_count é desconhecido
- exibe as tuplas remanescentes na tabela, de baixo para cima

## Bloco de Código 2.9.1

```
DO $$  
DECLARE  
    cur_delete REFCURSOR;  
    tupla RECORD;  
BEGIN  
    -- scroll para poder voltar ao início  
    OPEN cur_delete SCROLL FOR  
        SELECT  
            *  
        FROM  
            tb_top_youtubers;  
    LOOP  
        FETCH cur_delete INTO tupla;  
        EXIT WHEN NOT FOUND;  
        IF tupla.video_count IS NULL THEN  
            DELETE FROM tb_top_youtubers WHERE CURRENT OF cur_delete;  
        END IF;  
    END LOOP;  
  
    -- loop para exibir item a item, de baixo para cima  
    LOOP  
        FETCH BACKWARD FROM cur_delete INTO tupla;  
        EXIT WHEN NOT FOUND;  
        RAISE NOTICE '%', tupla;  
    END LOOP;  
    CLOSE cur_delete;  
END;  
$$
```

## *Bibliografia*

**PostgreSQL: Documentation: 14: PostgreSQL 14.2 Documentation.** PostgreSQL, 2022. Disponível em <<https://www.postgresql.org/docs/current/index.html>>. Acesso em junho de 2022.