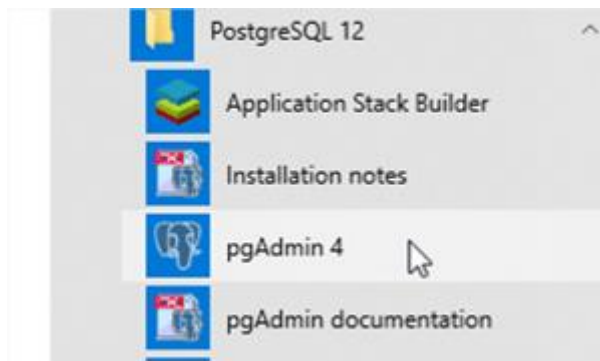
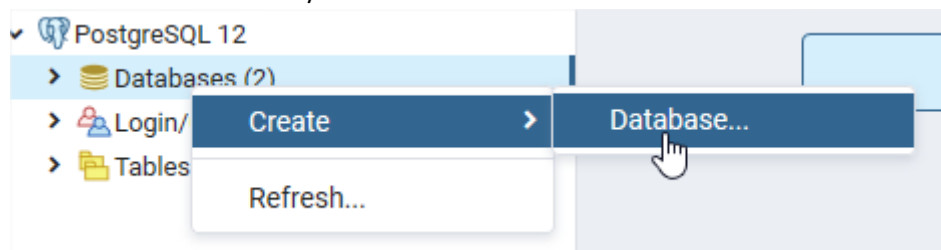


# Configurando o PostgreSQL 12

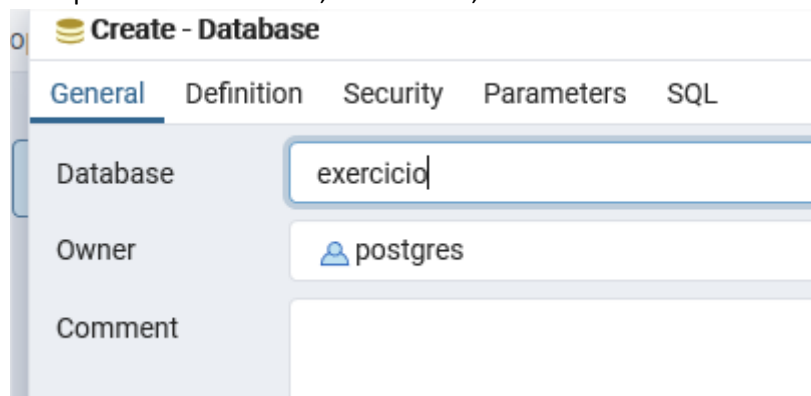
- 1) Baixe a versão 12 do PostgreSQL e instale em sua máquina de acordo com o seu sistema operacional (<https://www.enterprisedb.com/downloads/postgres-postgresql-downloads>).
- 2) Na instalação coloque uma senha para o usuário postgres. Guarde essa senha!
- 3) Para inicializar o PostgreSQL vá no menu iniciar e abra o PgAdmin 4



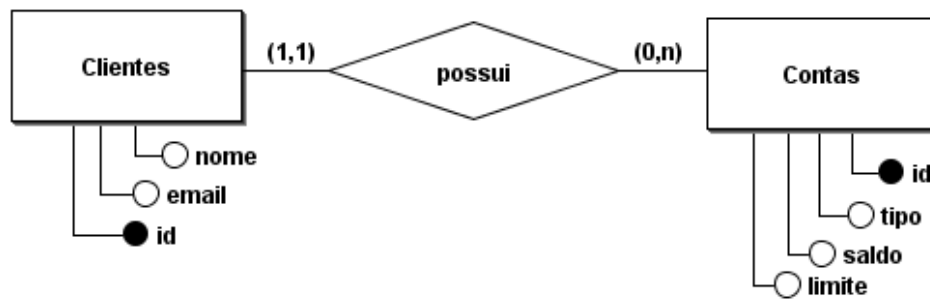
- 4) Ao iniciar, ele irá pedir a senha que você definiu no momento da instalação
- 5) Vamos criar um esquema que irá receber nossas tabelas. Clique com o botão direito sobre databases e create/database



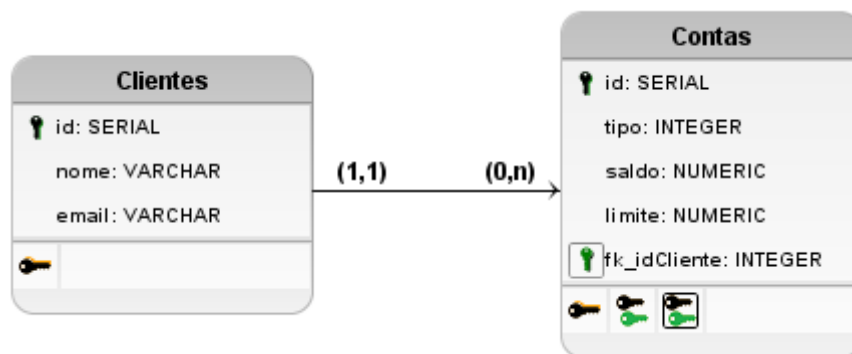
- 6) Coloque como “exercicio”, sem acento, e mande salvar.



- 7) Pronto, temos um banco, chamado “exercício” que irá receber as nossas tabelas. De acordo com a tarefa proposta pela Bete o modelo conceitual será:

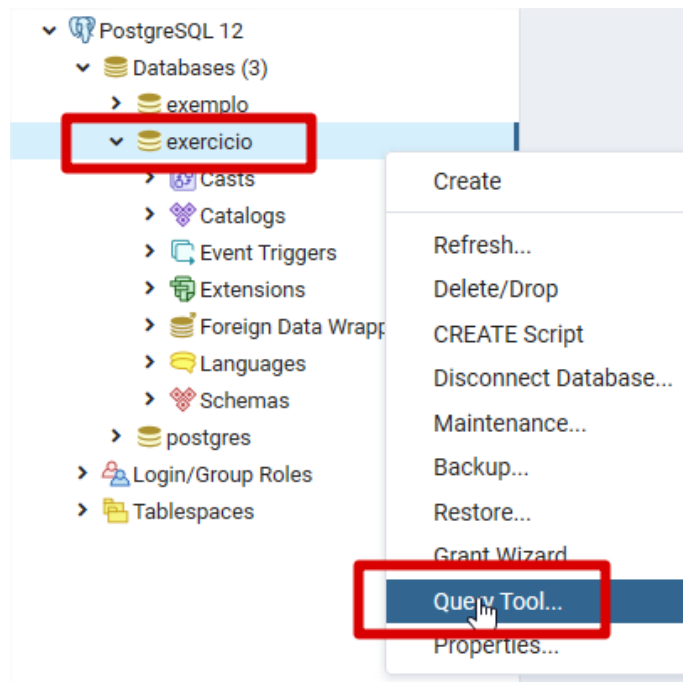


- 8) Vimos que Clientes possui uma chave primária chamada id, ela será autoincremental (1,2,3,4 ....) assim como em Contas. Percebemos também que é um relacionamento do tipo 1 para n. E em nossas conversões de conceitual para lógico o lado n irá receber a chave estrangeira da relação. Ficando o modelo lógico igual a imagem abaixo:



Percebe-se que temos a inclusão de um novo atributo na tabela Contas. Um fk (Foreign Key – chave estrangeira de Clientes dentro da tabela Contas). Perceba também que definimos tipos a nossos atributos. O tipo Serial no PostgreSQL é numérico e autoincremental, enquanto Varchar é tipo texto, O tipo Integer é numérico, não aceitando casas decimais. Já o Numeric permite salvar valores com casas decimais/ponto flutuante, ex. 500.55.

- 9) Nosso próximo passo é voltar no PostgreSQL e gerar o modelo físico do mesmo. Para isso, clique com o botão direito sobre o esquema “exercício” e escolha Query Tool



- 10) Vamos montar os Scripts conforme abaixo. Perceba que definimos uma estrutura padrão de SQL para a criação de novas tabelas a CREATE TABLE. Basicamente, temos que definir um nome para ela e adicionar os atributos com seus respectivos tipos. Por fim, adicionamos a palavra reservada PRIMARY KEY no atributo chave.

```
1 CREATE TABLE nome_tabela (  
2     atributo1 tipo_atributo PRIMARY KEY,  
3     atributo2 tipo_atributo,  
4     atributo3 tipo_atributo  
5 );
```

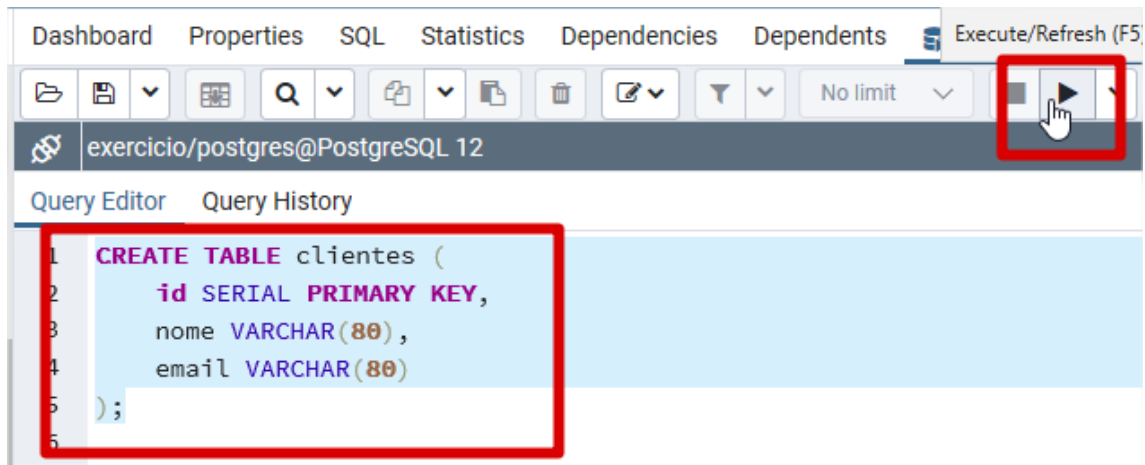
Para o nosso exemplo, o script ficará assim:

```
1 CREATE TABLE clientes (  
2     id SERIAL PRIMARY KEY,  
3     nome VARCHAR(80),  
4     email VARCHAR(80)  
5 );  
6  
7 CREATE TABLE contas (  
8     id SERIAL PRIMARY KEY,  
9     tipo INTEGER,  
10    saldo NUMERIC,  
11    limite NUMERIC,  
12    fk_idCliente INTEGER,  
13    FOREIGN KEY (fk_idCliente) REFERENCES clientes (id)  
14 );
```

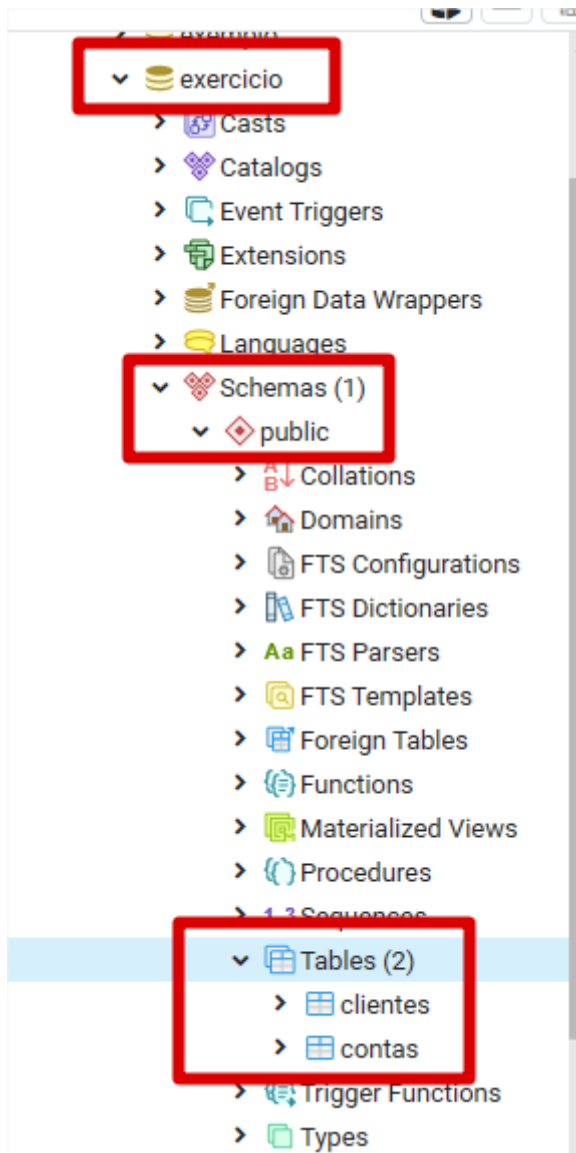
A diagram with a red arrow pointing from the 'id' attribute in the 'clientes' table (line 2) to the 'fk\_idCliente' attribute in the 'contas' table (line 12). Another red arrow points from the 'REFERENCES clientes (id)' part of line 13 to the 'id' attribute in the 'clientes' table.

Veja que na linha 13 temos a definição da chave estrangeira da tabela Clientes.

Para executar qualquer instrução em SQL temos que selecionar o script e rodar, conforme a imagem abaixo.



11) Pronto! As nossas tabelas estão persistidas. Para visualizá-las é só acessar o seguinte caminho no PostgreSQL



- 12) De acordo com trabalho proposto pela Bete, são realizadas algumas operações sobre essas tabelas, como inserção, alteração, deleção e seleção.
- 13) O primeiro será de inserção. Ou seja, iremos incluir um registro a nossa tabela. Para isso, tem-se o seguinte comando abaixo. Veja que os valores (VALUES) são respectivamente na ordem que você definiu anteriormente na linha, conforme as setas. Note que não inserimos o id, como ele é auto incremento o próprio banco irá se encarregar de inserir.

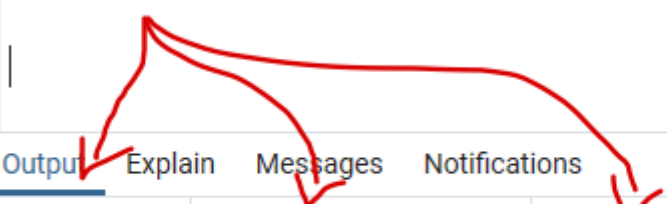
```
INSERT INTO clientes (nome,email) VALUES ('MARIA','MARIA@GMAIL.COM');
```



- 14) Como inserimos, temos que ver se realmente o dado foi inserido na tabela Clientes, para isso teremos que usar o SELECT. O "\*" quer dizer: traga todos os atributos da tabela Clientes. Como não definimos nenhuma regra, esse comando também irá trazer todos os registros (linhas) da tabela.

```
SELECT * FROM clientes;
```

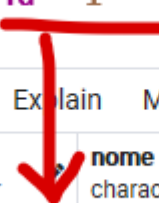
Table Output			Explain	Messages	Notifications
id	nome	email			
[PK] integer	character varying (80)	character varying (80)			
1	MARIA	MARIA@GMAIL.COM			



Também, podemos definir quais colunas serão retornadas. Além disso, podemos definir restrições na busca usando a cláusula Where.

```
SELECT id,nome FROM clientes  
WHERE id = 1
```

Table Output		Explain	Messages	Notifications
id	nome			
[PK] integer	character varying (80)			
1	MARIA			



Nesse exemplo, selecione as colunas id e nome onde o id seja igual a 1.

- 15) Para inserir em conta, temos que já ter um registro de clientes, pois a chave estrangeira requer uma referência a tabela Cliente. Veja no exemplo. Inserimos uma conta comum a cliente maria (ID = 1)

Query Editor   Query History

```

1  --Tipo de conta são: 0 = conta comum e 1 = conta especial
2  INSERT INTO contas(tipo, saldo, limite, fk_idcliente) VALUES (0,500.55,1000,1);
3
4

```

Data Output   Explain   Messages   Notifications

	id [PK] integer	nome character varying (80)	email character varying (80)
1	1	MARIA	MARIA@GMAIL.COM

**TABELA CLIENTES**

Para visualizarmos os dados da tabela contas podemos dar um select \* from contas. Veja que em fk\_idcliente registrou-se o id da tabela Clientes.

```
SELECT * FROM contas
```

Data Output   Explain   Messages   Notifications

id [PK] integer	tipo integer	saldo numeric	limite numeric	fk_idcliente integer
1	0	500.55	1000	1

- 16) Podemos também alterar algum registro. Para isso, usaremos o UPDATE. No exemplo abaixo, percebeu-se que o saldo estava errado do usuário 1. Desejamos alterar o saldo de 500.55 para 1500.55. Veja o exemplo:

```

1  UPDATE contas
2  SET saldo = 1500.55
3  WHERE fk_idcliente = 1
4

```

Data Output   Explain   Messages   Notifications

id [PK] integer	tipo integer	saldo numeric	limite numeric	fk_idcliente integer	
1	1	0	500.55	1000	1

Após a execução ficará assim:

```
select * from contas
```

Data Output   Explain   Messages   Notifications

id [PK] integer	tipo integer	saldo numeric	limite numeric	fk_idcliente integer
1	0	1500.55	1000	1

- 17) Podemos também remover um registro. Imagine que o usuário 1 queira cancelar sua conta e que deseje removê-la. Para isso, usamos o DELETE com alguma condição.

1

DELETE FROM

contas

2

WHERE id = 1;

3

4

SELECT \* FROM

contas;

Data Output

Explain

Messages

Notifications

<div>id</div> <div>[PK] integer</div>	<div>tipo</div> <div>integer</div>	<div>saldo</div> <div>numeric</div>	<div>limite</div> <div>numeric</div>	<div>fk_idcliente</div> <div>integer</div>

Veja que deletamos o registro com id = 1. Após a seleção notamos que não há mais o registro na tabela.