Continuando estudo SQL

Chave Estrangeira:

Até agora fazemos chaves estrangeiras dessa maneira:

**CREATE** **TABLE** aluno\_curso (

aluno\_id **INTEGER**,

curso\_id **INTEGER**,

**PRIMARY** KEY (aluno\_id, curso\_id),

**FOREIGN** KEY (aluno\_id)

**REFERENCES** aluno (id),

**FOREIGN** KEY (curso\_id)

**REFERENCES** curso (id)

);

Mas também é possível fazer de uma maneira mais intuitiva:

CREATE TABLE aluno\_curso (

aluno\_id INTEGER NOT NULL REFERENCES aluno(id),

curso\_id INTEGER NOT NULL REFERENCES curso(id),

PRIMARY KEY (aluno\_id, curso\_id)

);

Criando a variável, e referenciando.

**Usando in no filtro:**

Quando queremos buscar um filtro, normalmente usamos o recurso WHERE e usamos um Sistema de igualdade para verificar se entra no filtro ou n.

SELECT \* FROM curso WHERE categoria\_id = 1 OR categoria\_id = 2;

Com o uso do IN, isso fica mais facilitado:

SELECT \* FROM curso WHERE categoria\_id IN (1,2);

Desta forma, o in, englobe todos os valores colocados reduzindo o codigo.

**SubQuerys**

**SELECT** categoria.nome **AS** categoria,

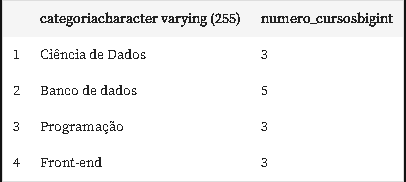
COUNT(curso.id) **as** numero\_cursos

**FROM** categoria

**JOIN** curso **ON** curso.categoria\_id = categoria.id

**GROUP** **BY** categoria;

Nesta query vemos o nome das categorias, e a quantidade das mesmas:



Mas e se eu quiser:

**SELECT** categoria,

numero\_cursos

**FROM** algum\_lugar

**WHERE** numero\_cursos > 3;

Aqui, eu quero fazer outra restrição onde eu busco o resultado de outra tabela, outro relatório, não existe uma tabela número de cursos, mas existe um SELECT que me da uma tabela de número de cursos, dessa forma basta unir os dois selects:

**SELECT** categoria,

numero\_cursos

**FROM** (

**SELECT** categoria.nome **AS** categoria,

COUNT(curso.id) **as** numero\_cursos

**FROM** categoria

**JOIN** curso **ON** curso.categoria\_id = categoria.id

**GROUP** **BY** categoria

) **AS** categoria\_cursos

**WHERE** numero\_cursos > 3;

Normalmente, é possível modificar a query inicial, e resolver o problema sem subquerys

**Funções do POSTGRE:**

**Manipulando strings:**

https://www.postgresql.org/docs/9.1/functions-string.html

**SELECT** (primeiro\_nome || ' ' || ultimo\_nome) **AS** nome\_completo **FROM** aluno;

O || tem a função de juntar as strings, mas existe uma finção que tem o mesmo objetivo, mas de maneira mais eficiente:

SELECT CONCAT('Vinicius', NULL, 'Dias');

A função concat permite fazer a junção das strings, mesmo se um dos parâmetros for NULL, o que n é permitido no SELECT

Assim como no python, a função UPPER tbm funciona para letras maiúsculas:

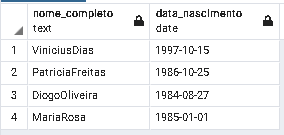
SELECT UPPER CONCAT('Vinicius', ' ', 'Dias');

**Funções de data:**

**https://www.postgresql.org/docs/9.1/functions-datetime.html**

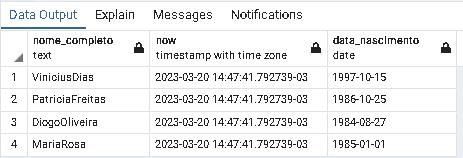
Now me da a data e hora atual:

**SELECT** (primeiro\_nome || ultimo\_nome) **AS** nome\_completo, data\_nascimento **FROM** aluno;



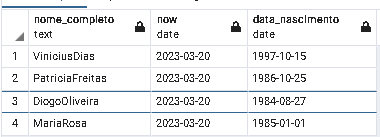
Com NOW():

SELECT (primeiro\_nome || ultimo\_nome) AS nome\_completo, NOW(), data\_nascimento FROM aluno;



Para transformar apenas em data use ::DATE no NOW:

**SELECT** (primeiro\_nome || ultimo\_nome) **AS** nome\_completo, NOW()::**DATE**, data\_nascimento **FROM** aluno;



Dessa forma, podemos descobrir a idade dos alunos:

**SELECT** (primeiro\_nome || ultimo\_nome) **AS** nome\_completo,

(NOW()::DATE - data\_nascimento) / 365 **FROM** aluno;

O cálculo (NOW()::DATE - data\_nascimento nos entrega um valor em dias, por isso deve ser dividido por 365, mas essa conta ficou feia, será que tem como fazer mais fácil?

Podemos usar a função AGE()

**SELECT** (primeiro\_nome || ultimo\_nome) **AS** nome\_completo,

AGE(data\_nascimento) **AS** idade

**FROM** aluno;

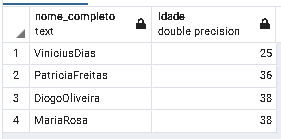


Para pegar apenas os anos podemos usar a função EXTRACT:

**SELECT** (primeiro\_nome || ultimo\_nome) **AS** nome\_completo,

EXTRACT(**YEAR** **FROM** AGE(data\_nascimento)) **AS** idade

**FROM** aluno;



Funções matemáticas:

<https://www.postgresql.org/docs/9.5/functions-math.html>

Conversão de dados:

<https://www.postgresql.org/docs/9.5/functions-formatting.html>

**Criar VIEW**

**VIEW nada mais é do que guardar uma tabela, e as vezes isso pode ajudar em facilidade mas perder em performance.**

Como nomear tabelas/relatórios, podendo pegar consultas inteiras, querys inteiras:

CREATE VIEW vw\_cursos\_por\_categoria

**AS** **SELECT** categoria.nome **AS** categoria,

COUNT(curso.id) **as** numero\_cursos

**FROM** categoria

**JOIN** curso **ON** curso.categoria\_id = categoria.id

**GROUP** **BY** categoria;

SELECT \* FROM vw\_cursos\_por\_categoria;

Podemos trazer tbm alguns filtros:

**CREATE** **VIEW** vw\_cursos\_programacao **AS** **SELECT** nome **FROM** curso **WHERE** categoria\_id = 2

SELECT vw\_cursos\_programacao WHERE nome = 'PHP'