Resumo QSL, no PostgreSQL

**Principais types utilizados:**

|  |  |
| --- | --- |
| **Types Números** | |
| integer | inteiro |
| real | número real, com até 6 dígitos de precisão |
| serial | se completa sozinho, tipo um contador |
| numeric (n,n) | pode definir o número de caracteres e o número de casas decimais |

|  |  |
| --- | --- |
| **Types Textos** | |
| varchar(n) | nome normal, com caráteres limitados(n), mas n precisos |
| char(n) | caracteres precisos, tem que ter o valor de n |
| text | sem restrição de caracteres |

|  |  |
| --- | --- |
| **Types datas** | |
| date | Data |
| Time | Hora |
| Timestamp | Data e hora |

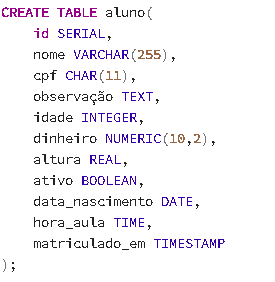
|  |  |
| --- | --- |
| **Types boolean** | |
| boolean | True false ou Null |

**Selecionando tabelas:**

SELECT \* FROM aluno; seleciona a tabela aluno

**Criando Tabela:**

CREATE TABLE aluno(

 id SERIAL,

nome VARCHAR(255),

cpf CHAR(11),

observação TEXT,

idade INTEGER,

dinheiro NUMERIC(10,2),

altura REAL,

ativo BOOLEAN,

data\_nascimento DATE,

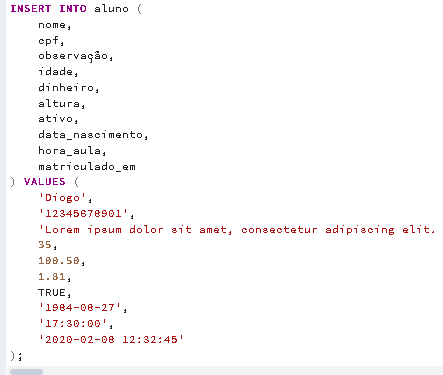
hora\_aula TIME,

matriculado\_em TIMESTAMP

);

**Inserindo valores em uma tabela:**

INSERT INTO aluno (

 nome,

cpf,

observação,

idade,

dinheiro,

altura,

ativo,

data\_nascimento,

hora\_aula,

matriculado\_em

) VALUES (

'Diogo',

'12345678901',

‘blablabla,

35,

100.50,

1.81,

TRUE,

'1984-08-27',

'17:30:00',

'2020-02-08 12:32:45'

);

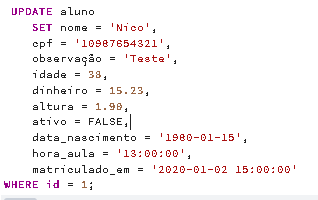
**Para modificar um valor, ou varios de um linha da tabela, é recomendado primeiro filtrar esse valor, como fazemos filtro?**

**SELECT** \* **FROM** aluno

**WHERE** id = 1

Traduzindo: selecione da tabela aluno, onde o id for = 1

Após isso fica mais fácil visualizar o que será substituído com o comando update:



**UPDATE** aluno

**SET** nome = 'Nico',

cpf = '10987654321',

observacao = 'Teste',

idade = 38,

dinheiro = 15.23,

altura = 1.90,

ativo = **FALSE**,

data\_nascimento = '1980-01-15',

hora\_aula = '13:00:00',

matriculado\_em = '2020-01-02 15:00:00'

**WHERE** id = 1;

Traduzindo: update (atualize na tabela alunos) set ( faça essas mudanças) where (onde o id= 1)

Após isso sempre dando SELECT \* FROM aluno; para verificar como ficou a mudança.

**Como deletar um registro em uma tabela?**

Assim como para selecionar um filtro é feito, um:

**SELECT** \*

**FROM** aluno

**WHERE** nome = 'Nico';

,

Para deletar um registro o procedimento é similar, apenas mudando o SELECT para DELETE:

DELETE\*

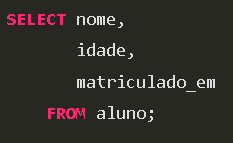
**FROM** aluno

**WHERE** nome = 'Nico';

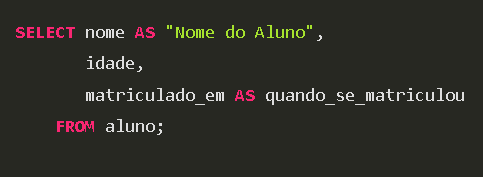
É NECESSARIO USAR O FILTRO WHERE PARA LIMITAR O DSEU DELETE, PARA EVITAR DELETAR A TABELA INTEIRA OU INVES DE APENAS 1 REGISTRO

**Como selecionar colunas especificas na tabela:**

Até agora estamos usando sempre o \* quando damos um select, mas se quisermos selecionar uma coluna especifica, no lugar do \* deve-se colocar o nome da coluna:



Também podemos fazer essa pesquisa mudando o nome da coluna:



Aprendendo mais com seleção:

Aumentando a tabela:

**INSERT** **INTO** aluno (nome) **VALUES** ('Vinícius Dias');

**INSERT** **INTO** aluno (nome) **VALUES** ('Nico Steppat');

**INSERT** **INTO** aluno (nome) **VALUES** ('João Roberto');

**INSERT** **INTO** aluno (nome) **VALUES** ('Diego');

**Quando usamos o filtro where, usamos referente a alguma coluna especifica e para isso temos:**

|  |  |
| --- | --- |
| = | Igual |
| < > ou != | Diferente |
| LIKE \_  (EX **LIKE** 'Di\_go')  (EX **LIKE** '\_iogo') | \_ogo (procura terminação em ogo (diogo))  Ou seja, não importando o caractere com o \_ mas mantendo o sufixo determinado. |
| NOT LIKE \_  (EX **NOT LIKE** 'Di\_go')  (EX **NOT LIKE** '\_iogo') | Mesma coisa que diferente, dessa forma, vai ignorer os sufixos destacados |
| LIKE %  (EX **LIKE** 'D%') iniciados com D  (EX **LIKE** '%s') Terminados com s  (EX **LIKE** '% %') com espaço no meio (nome composto)  (EX **LIKE** '%a% f%') Palavras que tenha “a” e “f” em sua composição | Da ora fazer filtros iniciados com, terminados com, ou com coisas entre os texto |
| IS NULL | Está nulo, contém uma uma célula nula, mostra as células que estão vazias |
| IS NOT NULL | Está preenchido, mostra as células preenchidas |

Ex: =

**SELECT** \*

**FROM** aluno

**WHERE** nome = 'Diogo';

Ex: < > !=

**SELECT** \*

**FROM** aluno

**WHERE** nome < > 'Diogo';

Obs: o Word n deixa colocar o < > junto, mas é junto na programação

**SELECT** \*

**FROM** aluno

**WHERE** nome != 'Diogo';

Ex: LIKE \_

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** 'Di\_go';

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** '\_iogo';

Ex: NOT LIKE

**SELECT** \*

**FROM** aluno

**WHERE** nome **NOT LIKE** 'Di\_go';

Ex: LIKE %

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** 'D%';

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** '% %';

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** '%i%a%';

Ex: IS NULL

**SELECT** \*

**FROM** aluno

**WHERE** cpf **IS** **NULL**;

Ex: IS NOT NULL

**SELECT** \*

**FROM** aluno

**WHERE** cpf **IS** **NOT NULL**;

**Para filtros em colunas contendo números, as operações matemáticas funcionam bem:**

|  |  |
| --- | --- |
| < > != | Diferente |
| = | igual |
| >= e <= | Maior igual e menor igual |
| >(maior) <(menor) | Maior e menor |
| BETWEEN  **BETWEEN** 10 **AND** 20; | ENTRE, inclusivo, pega todos os numero entre 10 e 20 incluindo eles (como no exemplo) |

Ex BETWEEN:

**SELECT** \*

**FROM** aluno

**WHERE** idade **BETWEEN** 10 **AND** 20;

Usando AND e OR:

Podemos também combinar os filtros mostrados anteriormente:

EX: AND (todas as condições são atendidas)

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** 'D%'

**AND** cpf **IS** **NOT** **NULL**;

EX: OR (Pelo menos umas das opções devem ser verdadeiras)

**SELECT** \*

**FROM** aluno

**WHERE** nome **LIKE** 'Diogo'

**OR** nome **LIKE** 'Rodrigo';

**Chave Primaria:**

Quando for criar um valor não nulo é valido já o fazer, para evitar locais com valores nulos indesejados:

Ex:

**CREATE** **TABLE** curso (

id **INTEGER** **NOT** **NULL**,

nome **VARCHAR**(255) **NOT** **NULL**

);

Ou:

Ex: o unique escrito no exemplo, impede que possa ser criado novos cursos com Ids iguais.

**CREATE** **TABLE** curso (

id **INTEGER** **NOT** **NULL** UNIQUE,

nome **VARCHAR**(255) **NOT** **NULL**

);

Mas o fato de ser único, é a mesma ideia de ser uma chave primária, dessa forma podemos fazer como primary key:

Chaves primarias não podem ser nulas e tem que ser únicas

**CREATE** **TABLE** curso (

id **INTEGER** **PRIMARY** KEY,

nome **VARCHAR**(255) **NOT** **NULL**

);

**Chave Estrangeira:**

**Criando tabela cruso:**

CREATE TABLE curso (

id INTEGER PRIMARY KEY,

nome VARCHAR(255) NOT NULL

);

INSERT INTO curso (id, nome) VALUES (1, 'HTML');

INSERT INTO curso (id, nome) VALUES (2, 'Javascript');

**Criando tebala aluno:**

CREATE TABLE aluno (

id SERIAL PRIMARY KEY,

nome VARCHAR(255) NOT NULL

);

INSERT INTO aluno (nome) VALUES ('Diogo');

INSERT INTO aluno (nome) VALUES ('Vinícius');

SELECT \* FROM curso;

SELECT \* FROM aluno;

Agora podemos criar uma tabela que relaciona as duas tabelas:

**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)

);

Para criar uma chave estrangeira primeiro deve descrever ela : FOREIGN KEY (curso\_id),

E depois referenciar para a tabela inicial: REFERENCES curso (id)

**Como puxar várias tabelas em conjunto, com informações conjuntas:**

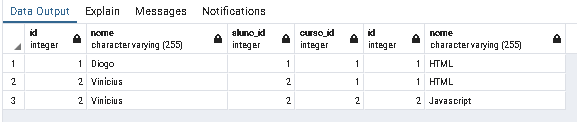
SELECT \*

FROM aluno

JOIN aluno\_curso ON aluno\_curso.aluno\_id = aluno.id

JOIN curso ON curso.id = aluno\_curso.curso\_id

Traduzindo: Pega a tabela aluno, junta com a tabela aluno\_curso, onde aluno\_curso.aluno\_id = aluno.id, eeeeee junto com a tabela curso onde curso.id = aluno\_curso.curso\_id



Se quiser pegar apenas o nome, poderia ser feito assim:

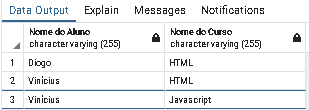
SELECT aluno.nome as "Nome do Aluno",

curso.nome as "Nome do Curso"

FROM aluno

JOIN aluno\_curso ON aluno\_curso.aluno\_id = aluno.id

JOIN curso ON curso.id = aluno\_curso.curso\_id



**Mas e se tiver um curso sem aluno matriculado, ou um aluno sem curso? Eles não aparecem na tabela anterior, por só mostra os que estão cumprindo o id das duas.**

**Como poderíamos verificar isso?**

**SELECT** aluno.nome **as** "Nome do Aluno",

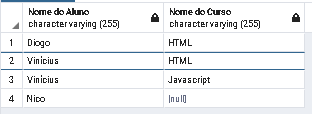
curso.nome **as** "Nome do Curso"

**FROM** aluno

LEFT **JOIN** aluno\_curso **ON** aluno\_curso.aluno\_id = aluno.id

LEFT **JOIN** curso **ON** curso.id = aluno\_curso.curso\_id

Com left join, nos trazemos todos os dados ta tabela inicial (aluno por isso a esqueda, mesmo que no curso esteja null



Ou seja, eu tenho dado na tabela de aluno, e não de curso.

**O contrário também pode ser verificado:**

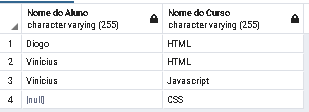
**SELECT** aluno.nome **as** "Nome do Aluno",

curso.nome **as** "Nome do Curso"

**FROM** aluno

RIGHT **JOIN** aluno\_curso **ON** aluno\_curso.aluno\_id = aluno.id

RIGHT **JOIN** curso **ON** curso.id = aluno\_curso.curso\_id



**E se quiser todos os dados?**

**SELECT** aluno.nome **as** "Nome do Aluno",

curso.nome **as** "Nome do Curso"

**FROM** aluno

FULL **JOIN** aluno\_curso **ON** aluno\_curso.aluno\_id = aluno.id

FULL **JOIN** curso **ON** curso.id = aluno\_curso.curso\_id



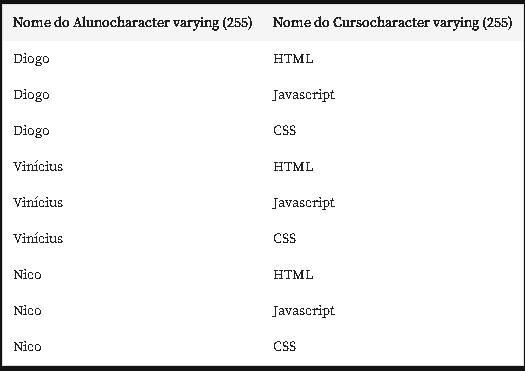
**Caso queria crusar as tabelas:**

**SELECT** aluno.nome **as** "Nome do Aluno",

curso.nome **as** "Nome do Curso"

**FROM** aluno

CROSS **JOIN** curso



**Neste momento se tentarmos deletar um aluno de uma tabela, vai impedir pq em outra taberla já tem o aluno, para que quando for deletado da tabela aluno, delete tbm da tabela aluno\_curso basta:**

CREATE TABLE aluno\_curso (

aluno\_id INTEGER,

curso\_id INTEGER,

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

FOREIGN KEY (aluno\_id)

REFERENCES aluno (id)

ON DELETE CASCADE,

FOREIGN KEY (curso\_id)

REFERENCES curso (id)

);

Agora se deletar, deleta nos 2:

DELETE FROM aluno WHERE id = 1

Também é possível fazer um UPDATE:

CREATE TABLE aluno\_curso (

aluno\_id INTEGER,

curso\_id INTEGER,

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

FOREIGN KEY (aluno\_id)

REFERENCES aluno (id)

ON DELETE CASCADE

ON UPDATE CASCADE,

FOREIGN KEY (curso\_id)

REFERENCES curso (id)

);

Assim quando tentar dar update entre tabelas com relacionamentos, vai permitir:

UPDATE aluno

SET id = 10

WHERE id = 2;

Quando quisermos ordenar usamos order by:

CREATE TABLE funcionarios(

id SERIAL PRIMARY KEY,

matricula VARCHAR(10),

nome VARCHAR(255),

sobrenome VARCHAR(255)

);

SELECT \*

FROM funcionarios

ORDER BY nome;

Se quiser em ordem decrescente, adicione um desc:

SELECT \*

FROM funcionarios

ORDER BY nome DESC;

Tambem poderia ordenar pela posição:

Coluna 1 = 1 ,coluna 2 = 2 e etc...

**SELECT** \*

**FROM** funcionarios

**ORDER** **BY** 3

Como limitar consulta: trazer os 5 primeiros da tabela.

**SELECT** \*

**FROM** funcionarios

**ORDER** **BY** id

LIMIT 5;

Caso queria começar a consulta a partir de uma coluna:

O offset faz com que comece a partir da linha 2 ao invez da 1

**SELECT** \*

**FROM** funcionarios

**ORDER** **BY** id

**OFFSET** 1;

**Função de agregação**

-- COUNT - Retorna a quantidade de registros

-- SUM - Retorna a soma dos registros

-- MAX - Retorna o maior valor dos registros

-- MIN - Retorna o menor valor dos registros

-- AVG - Retorna a média dos registros

**EX:**

SELECT COUNT (id),

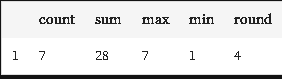
SUM(id),

MAX(id),

MIN(id),

ROUND(AVG(id),0)

FROM funcionarios;



E quando tem nome repetidos em uma tabela, como mostrar apenas 1:

**SELECT** **DISTINCT**

nome

**FROM** funcionarios

**ORDER** **BY** nome;

Caso queria tirar as repetições e mostrar a quantidade de repetições ou usar uma função de agregação deve se usar o grup by:

**SELECT**

nome,

sobrenome,

COUNT(\*)

**FROM** funcionarios

**GROUP** **BY** nome, sobrenome

**ORDER** **BY** nome;

Quando quiser fazer fazer um filtro, com grup by, não se usa o where, se usa having:

**SELECT** nome

**FROM** funcionarios

**GROUP** **BY** nome

**HAVING** COUNT(id) > 1;