

Instituto Federal Sul-rio-grandense – Campus Camaquã
Disciplina: Banco de Dados II – Turma 2024
Professor: Vinícius Alves Hax
Assunto: Revisão sobre SQL

Instruções gerais: para os exercícios abaixo você pode conversar com seus colegas (em voz baixa) e até consultar a Internet. Mas NÃO use o chatGPT. Ele nem sempre estará lá :P

Para os exercícios de número 1 até 4 Utilize as tabelas e registros abaixo:

```
CREATE TABLE teacher(  
    id INT,  
    name VARCHAR(200),  
    side VARCHAR(200)  
);
```

```
CREATE TABLE student(  
    id SERIAL,  
    name VARCHAR(200),  
    teacher_id INT  
);
```

```
INSERT INTO teacher (id, name, side) VALUES (1, 'Palpatine', 'Dark');  
INSERT INTO teacher (id, name, side) VALUES (2, 'Yoda', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (3, 'Dookan', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (4, 'Qui-gon Jin', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (5, 'Obi Wan', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (6, 'Anakin', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (7, 'Luke', 'Light');  
INSERT INTO teacher (id, name, side) VALUES (8, 'Mace Windu', 'Light');
```

```
INSERT INTO student (name, teacher_id) VALUES ('Luke', 2);  
INSERT INTO student (name, teacher_id) VALUES ('Anakin', 5);  
INSERT INTO student (name, teacher_id) VALUES ('Anakin', 1);  
INSERT INTO student (name, teacher_id) VALUES ('Obi Wan', 4);  
INSERT INTO student (name, teacher_id) VALUES ('Qui-gon Jin', 3);  
INSERT INTO student (name, teacher_id) VALUES ('Ahsoka', 6);  
INSERT INTO student (name, teacher_id) VALUES ('Rei', 7);  
INSERT INTO student (name, teacher_id) VALUES ('Kylo Ren', 7);  
INSERT INTO student (name) VALUES ('Palpatine');
```

1) Mostre SOMENTE o nome dos professores que possuem PELO MENOS um aprendiz (student)

Para resolver esse exercício sem usar subqueries devemos usar o mecanismo dos JOINS. A questão é escolher o JOIN adequado para essa situação: left/right (essencialmente o mesmo porém com os nomes das tabelas trocadas), inner, full (também chamado de ‘full outer’) ou cross join.

Por eliminação: o cross join não faz sentido aqui pois geraria todas as possibilidades. O full join mostraria linhas de aprendizes que não tem um professor portanto também não faz sentido.

Podemos usar um inner join (pois queremos todos os professores com pelo menos um aluno):

```
SELECT DISTINCT(t.name) as t_name FROM  
teacher t INNER JOIN student s ON  
t.id = s.teacher_id;
```

Podemos evitar repetições de nomes usando a função DISTINCT.

Nesse caso, também usar o left join para chegar na mesma resposta:

```
SELECT DISTINCT(t.name) as t_name FROM  
teacher t LEFT JOIN student s  
ON t.id = s.teacher_id  
WHERE s.name IS NOT null;
```

Na resposta acima excluimos as linhas dos professores sem aluno com a cláusula WHERE. Quando usamos INNER JOIN é como se o próprio SQL já adicionasse o WHERE “automaticamente”.

2) Mostre SOMENTE os nomes dos professores que tem mais de um aluno



Aqui novamente temos mais de um caminho para obter a resposta. Primeiramente vamos usar a estratégia das subconsultas, até mesmo para exercitar esse mecanismo. Nas subconsultas (ou subqueries) usamos o resultado de uma query para montar outra. Observe o raciocínio acima para obter o que é pedido.

Primeiramente, como poderíamos obter a quantidade de alunos que cada professor tem?

Poderíamos usar a query abaixo:

```
SELECT teacher_id, count(*) as contador FROM student GROUP BY teacher_id;
```


Usando essa query nos dados acima obtemos o seguinte resultado:

	teacher_id integer 	contador bigint 
1	[null]	1
2	3	1
3	5	1
4	4	1
5	6	1
6	2	1
7	7	2
8	1	1

Podemos montar uma nova consulta, que utilize o resultado da query acima como se fosse uma tabela temporária, para filtrar somente as linhas onde o valor da coluna “contador” é maior do que um determinado limite, no caso maior do que 1.

```
SELECT teacher_id FROM
(
  SELECT teacher_id, count(*) as contador
  FROM student GROUP BY teacher_id
) AS subquery
WHERE contador >=2;
```

O resultado agora é mostrado na imagem abaixo:

	teacher_id integer 
1	7

Note que usamos uma query como entrada de outra. Esse é o conceito de suquery ou subconsulta.

Mas na verdade o que foi pedido foi o nome do professor e não o seu ID. Poderíamos usar uma linguagem de programação para com base no resultado acima fazer uma segunda consulta. Ou então podemos utilizar novamente o mecanismo de subconsultas.

```
SELECT DISTINCT(t.name) as t_name FROM
  teacher t LEFT JOIN student s
  ON t.id = s.teacher_id
  WHERE t.id IN
  (SELECT teacher_id FROM
    (
      SELECT teacher_id, count(*) as contador
      FROM student GROUP BY teacher_id
    ) AS subquery
  WHERE contador >=2);
```

Note que fazendo isso estamos potencialmente consumindo muitos recursos do nosso banco de dados pois precisamos fazer três buscas. Então, quando possível, as subconsultas devem ser evitadas. Mas é interessante saber que elas existem, se for necessário.

A cláusula WHERE não permite funções que trabalhem com dados agregados tal como count(), porém nesses casos existe o parâmetro HAVING.

Podemos obter então o mesmo resultado com o SQL abaixo:

```
SELECT teacher.name
FROM teacher JOIN student
ON teacher.id = student.teacher_id
GROUP BY teacher.name
```

HAVING COUNT(student.id) > 1;

Veja o link abaixo para mais detalhes sobre o parâmetro HAVING
https://www.w3schools.com/sql/sql_having.asp

Observe também como a solução com subconsultas pode ser testada: sempre de “dentro” pra “fora” e testando os resultados intermediários.

3) Mostre SOMENTE os nomes dos aprendizes que não possuem professor cadastrado

Note que essa consulta nem precisa do join para mostrar o que precisamos:

```
SELECT name FROM student  
WHERE teacher_id IS NULL;
```

Podemos adicionar a cláusula DISTINCT, se necessário, para evitar repetições.

4) Imagine que vai haver um campeonato de vôlei onde todos os professores (tabela teacher) se enfrentarão em um torneio onde cada partida vai ter dois participantes. Na primeira fase do torneio todos os participantes disputarão contra todos os outros. Gere todas as possibilidades de confrontos utilizando somente a linguagem SQL.

Qual o tipo de JOIN que usamos para gerar todas as combinações? Para isso temos o CROSS JOIN (chamado na matemática de produto cartesiano). Assim como podemos usar o CROSS join entre duas tabelas diferentes podemos fazer o produto cartesiano de uma tabela com ela mesma. Isso é bastante útil para resolver o que é pedido.

Então poderíamos usar

```
SELECT t1.name as participante1, t2.name as participante2  
FROM teacher as t1  
CROSS JOIN teacher as t2;
```

Porém a resposta acima tem um problema: geramos assim o confronto de uma pessoa com ela mesma, o que não faz sentido. Podemos resolver isso com uma cláusula WHERE:

```
SELECT t1.name as participante1, t2.name as participante2  
FROM teacher as t1  
CROSS JOIN teacher as t2  
WHERE t1.name != t2.name;
```

Porém ainda temos um problema: o par de confrontos se repete, pois por exemplo no CROSS JOIN ‘Palpatine’ e ‘Yoda’ são um par diferente de ‘Yoda’ e ‘Palpatine’. Se for um campeonato onde cada confronto deveria ser único o que poderíamos fazer?

Um ‘truque’ que podemos adotar baseia-se no fato de que cada registro tem uma chave e que ela é única. Podemos então por exemplo comparar as chaves e definir que o par só aparece na resposta se o id de um registro for menor do que o da segunda (se as colunas estivessem trocadas o resultado seria o mesmo nesse caso)

Por exemplo o SQL abaixo é a resposta que esperávamos:

```
SELECT t1.name as participante1, t2.name as participante2
FROM teacher as t1
CROSS JOIN teacher as t2
WHERE t1.name != t2.name
      AND t1.id < t2.id;
```

Para os exercícios abaixo utilize as tabelas abaixo:

```
CREATE TABLE professores ( id SERIAL PRIMARY KEY, nome VARCHAR(200) );
```

```
CREATE TABLE alunos ( matricula VARCHAR(100) PRIMARY KEY, nome VARCHAR(200) );
```

```
CREATE TABLE disciplinas ( id SERIAL PRIMARY KEY, nome VARCHAR(200) );
```

```
CREATE TABLE notas ( bimestre INT, ano INT, nota FLOAT, disciplina_id INT, aluno_matricula
VARCHAR(100), professor_id INT, CONSTRAINT fk_professor FOREIGN KEY(professor_id)
REFERENCES professores(id), CONSTRAINT fk_aluno FOREIGN KEY(aluno_matricula)
REFERENCES alunos(matricula), CONSTRAINT fk_disciplina FOREIGN KEY(disciplina_id)
REFERENCES disciplinas(id), PRIMARY KEY (disciplina_id, aluno_matricula, ano) );
```

5) Insira três professores na tabela professores com os seguintes nomes: "Carlos Silva", "Ana Souza" e "Roberto Costa".

```
INSERT INTO professores (nome) VALUES ('Carlos Silva'), ('Ana Souza'), ('Roberto Costa');
```

6) Insira três alunos na tabela alunos com as seguintes informações:

Matrícula: "2023001", Nome: "João Pereira"

Matrícula: "2023002", Nome: "Maria Oliveira"

Matrícula: "2023003", Nome: "Pedro Santos"

Matrícula: "2024001", Nome: "Marcos Silva"

```
INSERT INTO alunos (matricula, nome) VALUES
('2023001', 'João Pereira'),
('2023002', 'Maria Oliveira'),
('2023003', 'Pedro Santos'),
('2024001', 'Marcos Silva');
```

7) Insira três disciplinas na tabela disciplinas com os seguintes nomes: "Matemática", "História" e "Biologia".

```
INSERT INTO notas (bimestre, ano, nota, disciplina_id, aluno_matricula, professor_id) VALUES
(1, 2024, 8.5, (SELECT id FROM disciplinas WHERE nome='Matemática'), '2023001', (SELECT
id FROM professores WHERE nome='Carlos Silva')),
(1, 2024, 7.0, (SELECT id FROM disciplinas WHERE nome='História'), '2023002', (SELECT id
FROM professores WHERE nome='Ana Souza')),
(1, 2024, 9.0, (SELECT id FROM disciplinas WHERE nome='Biologia'), '2023003', (SELECT id
FROM professores WHERE nome='Roberto Costa'));
```

8) Insira os dados abaixo na tabela notas:

Bimestre: 1, Ano: 2023, Nota: 8.5, Disciplina: Matemática, Aluno: João Pereira, Professor: Carlos Silva

Bimestre: 1, Ano: 2023, Nota: 7.0, Disciplina: História, Aluno: Maria Oliveira, Professor: Ana Souza

Bimestre: 1, Ano: 2023, Nota: 7.0, Disciplina: História, Aluno: Marcos Silva, Professor: Ana Souza

Bimestre: 1, Ano: 2023, Nota: 9.0, Disciplina: Biologia, Aluno: Pedro Santos, Professor: Roberto Costa

Bimestre: 1, Ano: 2024, Nota: 9.0, Disciplina: Biologia, Aluno: Pedro Santos, Professor: Roberto Costa

```
INSERT INTO notas (bimestre, ano, nota, disciplina_id, aluno_matricula, professor_id) VALUES
(1, 2024, 8.5, (SELECT id FROM disciplinas WHERE nome='Matemática'), '2023001', (SELECT id FROM professores WHERE nome='Carlos Silva')),
(1, 2024, 7.0, (SELECT id FROM disciplinas WHERE nome='História'), '2023002', (SELECT id FROM professores WHERE nome='Ana Souza')),
(1, 2024, 9.0, (SELECT id FROM disciplinas WHERE nome='Biologia'), '2023003', (SELECT id FROM professores WHERE nome='Roberto Costa'));
```

9) Liste todos os professores cadastrados

```
SELECT * FROM professores;
```

10) Liste todas as notas dos alunos para a disciplina de História no primeiro bimestre de 2023

```
SELECT notas.nota, alunos.nome AS aluno_nome
FROM turmas
JOIN disciplinas ON notas.disciplina_id = disciplinas.id
JOIN alunos ON notas.aluno_matricula = alunos.matricula
WHERE disciplinas.nome = 'História' AND notas.bimestre = 1 AND notas.ano = 2023;
```

11) Liste todas as notas do aluno 'Pedro Santos'

```
SELECT notas.*, disciplinas.nome AS disciplina_nome, alunos.nome AS aluno_nome
FROM turmas
JOIN disciplinas ON notas.disciplina_id = disciplinas.id
JOIN alunos ON notas.aluno_matricula = alunos.matricula
JOIN professores ON notas.professor_id = professores.id
WHERE alunos.nome = 'Pedro Santos';
```

12) Atualize a nota do aluno "João Pereira" na disciplina "Matemática" para 9.0 no ano de 2024

```
SET nota = 9.0
WHERE aluno_matricula = '2023001' AND disciplina_id = (SELECT id FROM disciplinas
WHERE nome = 'Matemática') AND ano = 2024;
```

13) Mostre que retorna a nota média dos alunos em cada disciplina cadastrada ao lado do nome da disciplina

```
SELECT disciplinas.nome AS disciplina_nome, AVG(notas.nota) AS nota_media
FROM notas
JOIN disciplinas ON notas.disciplina_id = disciplinas.id
```

GROUP BY disciplinas.nome;

14) Exclua a disciplina 'Matemática'

DELETE FROM disciplinas WHERE nome = 'Matemática';