

Guia do desafio — sistema de chamados

Confira assim:

Material do aluno. Acompanha o desafio da aula 01 e conduz etapa por etapa.

Este guia **não traz o SQL pronto**. Ele mostra o raciocínio, o que conferir depois de cada passo e o erro que costuma aparecer ali. Se você copiar comando de algum lugar, copie do roteiro da aula e adapte — é assim que se aprende a escrever o próximo sozinho.

Antes de começar

Abra uma **janela de consulta nova** no pgAdmin, separada da que você usou no roteiro. Assim o SQL do desafio não se mistura com o dos exercícios da biblioteca.

Trabalhe no banco `biblioteca` mesmo — as tabelas novas convivem sem problema com `autor` e `livro`. Se preferir separar, crie um banco `chamados` e abra a janela nele; nesse caso confira o nome na aba antes de executar qualquer coisa.

Três coisas que ajudam:

- **Execute pouco de cada vez.** Selecione só o comando que você quer rodar e aperte `F5`. Mandar as trinta linhas de uma vez e receber um erro no meio é a forma mais rápida de se perder.
 - **Salve desde o começo.** `Ctrl + S` na janela de consulta, com o nome `aula01-seusobrenome.sql`, e vá salvando a cada etapa. Esse arquivo é a sua entrega, e máquina de laboratório trava.
 - **Erro não quebra nada.** Leia a mensagem antes de chamar alguém: o PostgreSQL costuma dizer exatamente o que está errado e onde. Corrija e execute de novo.
-

Etapa 0 — Entender o que está sendo pedido

Releia o enunciado e localize as três coisas que deram errado na empresa:

1. chamado atribuído a técnico que não trabalha mais lá;
2. cliente cadastrado duas vezes, com grafias diferentes;
3. relatório que não fecha porque os valores estão em texto livre.

Guarde essa lista. A pergunta 10 vai perguntar qual restrição sua impede cada um desses erros, e é muito mais fácil responder se você pensar nisso **enquanto** modela, e não depois.

Antes de escrever qualquer comando, responda para você mesmo: quais são as três tabelas, e qual delas depende das outras?

Etapa 1 — A tabela `cliente`

O enunciado pede: identificador próprio, nome obrigatório, cidade, e um campo de cadastro que não pode se repetir entre clientes.

Pense antes de escrever. Qual campo do mundo real serve para não deixar o mesmo cliente entrar duas vezes? Nome não serve, porque duas pessoas podem se chamar igual, e porque foi exatamente aí que a empresa errou. Procure algo que seja único por natureza.

Estrutura do comando:

```
CREATE TABLE cliente (  
    id      serial PRIMARY KEY,  
    -- nome, que não pode ficar vazio  
    -- cidade  
    -- o campo que não pode repetir  
);
```

```
INSERT INTO cliente (nome, cidade, /* seu campo */) VALUES ('Teste', 'Dourados', 'valor');  
SELECT * FROM cliente;
```

Depois tente inserir **outro** cliente com o mesmo valor naquele campo único. Se der erro, sua restrição está funcionando. Se entrar, falta alguma coisa.

Apague o cliente de teste antes de seguir:

```
DELETE FROM cliente WHERE nome = 'Teste';
```

Etapa 2 — A tabela `tecnico`

Identificador, nome obrigatório, especialidade, e um campo que diz se o técnico ainda está ativo.

Dois pontos merecem atenção:

O tipo do campo "ativo". Ele só tem duas respostas possíveis. Existe um tipo exato para isso, e não é `text` nem `integer`.

O valor inicial. O enunciado diz que ele deve começar como verdadeiro quando ninguém informar nada. Isso não se faz na aplicação: é uma restrição de tabela, e você viu qual na seção 5 da aula.

Confira assim: insira um técnico **sem** informar o campo ativo, e depois consulte a tabela. Se o valor apareceu como `true` sozinho, está certo.

```
INSERT INTO tecnico (nome, especialidade) VALUES ('Fulano', 'Elétrica');  
SELECT * FROM tecnico;
```

Etapa 3 — A tabela `chamado`

Esta é a última, e a ordem importa: ela aponta para as outras duas, então as outras duas precisam existir antes. Se você tentou criar `chamado` primeiro e tomou erro, é isso.

Precisa ter: identificador, descrição obrigatória, data de abertura, valor cobrado, e as ligações com cliente e com técnico. Mais uma restrição impedindo valor negativo.

Três decisões aqui:

O tipo do valor. Volte à seção 4 da aula, na demonstração do `0.1 + 0.2`. Você já sabe qual tipo não usar.

As ligações. Cada uma é uma coluna que guarda o identificador da outra tabela, mais a declaração que obriga aquele identificador a existir de verdade. A palavra-chave é `REFERENCES`.

A restrição do valor. `CHECK`, com a condição que faz sentido.

```
CREATE TABLE chamado (  
    id          serial PRIMARY KEY,  
    descricao   ...,  
    aberto_em   ...,  
    valor       ...,  
    cliente_id  ... REFERENCES ...,  
    tecnico_id  ... REFERENCES ...  
);
```

Confira assim: tente inserir um chamado apontando para um cliente que não existe, por exemplo o número 999. Precisa dar erro. Depois tente com valor negativo. Também precisa dar erro. São as duas restrições trabalhando.

Etapa 4 — Conferir a estrutura antes de povoar

Vale parar aqui e olhar o que você construiu:

```
SELECT table_name, column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name IN ('cliente', 'tecnico', 'chamado')
ORDER BY table_name, ordinal_position;
```

Olhe a coluna `is_nullable` : ela mostra quais campos aceitam vazio. Se um campo que deveria ser obrigatório aparece como `YES` , falta um `NOT NULL` .

Confira também se você tem, no conjunto das três tabelas, pelo menos um `NOT NULL` , um `UNIQUE` , um `CHECK` e um `DEFAULT` . O enunciado exige os quatro.

Etapa 5 — Os dados

Aqui tem uma armadilha, e ela é de propósito.

O enunciado pede quantidades mínimas, mas o que importa mesmo são **três condições**:

Condição	Por que ela existe
Um técnico sem nenhum chamado	A questão 2 só prova alguma coisa se existir alguém com zero
Um técnico inativo	A questão 8 pede a média só dos ativos; sem um inativo, o filtro não muda nada
Um chamado sem valor preenchido	A questão 6 conta os sem valor, e a 8 é afetada por ele

Se você inserir tudo bonitinho e completo, metade das consultas devolve o mesmo resultado com ou sem a cláusula que importa, e você não consegue saber se acertou.

Ordem de inserção: primeiro `cliente` e `tecnico` , depois `chamado` . Mesma lógica da criação.

Como deixar um valor vazio: não informe a coluna no `INSERT` , ou escreva `NULL` sem aspas. Aspas transformam em texto.

Confira assim, antes de ir para as consultas:

```
SELECT (SELECT count(*) FROM cliente) AS clientes,
       (SELECT count(*) FROM tecnico) AS tecnicos,
       (SELECT count(*) FROM chamado) AS chamados,
       (SELECT count(*) FROM chamado WHERE valor IS NULL) AS sem_valor,
       (SELECT count(*) FROM tecnico WHERE ativo = false) AS inativos;
```

Se `sem_valor` ou `inativos` vier zero, volte e ajuste os dados. Vale mais tempo aqui do que descobrir na questão 8.

Etapa 6 — As oito consultas

Resolva na ordem. Elas crescem em dificuldade e cada uma prepara a seguinte.

Q1 — Chamados com nomes em vez de números

O ponto de partida. Você precisa de duas junções: uma para o cliente e uma para o técnico. A mesma consulta pode ter mais de um `JOIN`.

Confira: o número de linhas precisa ser igual ao total de chamados. Se vier menos, alguma junção está descartando linhas.

Q2 — Quantos chamados cada técnico atendeu

A questão mais importante do desafio. O técnico sem chamado precisa aparecer com zero.

Duas decisões:

1. Qual `JOIN`? Reveja a seção 8 da aula: qual dos dois mantém as linhas que não casam?
2. Qual tabela fica à esquerda? Você quer todos os **técnicos**, então é a tabela de técnicos que precisa vir primeiro.

E mais um detalhe: `count(*)` conta linhas, e para o técnico sem chamado existe uma linha (com tudo nulo do lado direito), então ele devolveria 1. Conte uma coluna que venha da tabela de chamados, e não `*`, porque `count(coluna)` ignora nulo. É a diferença que a seção 9 mostrou.

Confira: o número de linhas precisa ser igual ao total de técnicos, e um deles precisa mostrar zero.

Q3 — Valor total por cliente

Agregação com `GROUP BY`, mais ordenação decrescente. Some a coluna de valor agrupando por cliente.

Cuidado: a soma ignora os valores nulos. Isso está certo e é o comportamento esperado.

Q4 — Chamados de clientes de uma cidade

Junção com filtro. Aqui o filtro é sobre linha, então é `WHERE`.

Q5 — Técnicos com mais de dois chamados

Igual à Q2, mas com filtro sobre o resultado da contagem. E aqui está o erro clássico: **não dá para usar `count()` dentro do `WHERE`**. Existe outra cláusula para filtrar depois do agrupamento, e ela está na seção 9.

Q6 — Quantos chamados estão sem valor

O enunciado pede **uma única consulta**. Há dois caminhos, ambos com uma consulta só: filtrar pelos nulos e contar, ou usar a diferença entre duas contagens.

Lembre: `= NULL` não funciona.

Q7 — O chamado mais caro, com o nome do cliente

Junção, ordenação decrescente pelo valor, e uma cláusula que devolve só a primeira linha.

Q8 — Média por técnico, só os ativos

Junta tudo: junção, agrupamento, filtro. Preste atenção em onde o filtro dos ativos entra — antes ou depois do agrupamento? Ele é sobre uma coluna da linha, não sobre o resultado de uma agregação.

Confira: o técnico inativo não pode aparecer no resultado.

Etapa 7 — As três perguntas

São de raciocínio e valem 3,0 dos 10. Não deixe para o fim com dois minutos de aula sobrando.

P1 — sobre o JOIN da Q2. Diga qual usou, por quê, e **o que aconteceria com o outro**. A segunda parte é a que vale: teste de verdade, troque o `JOIN` e veja a diferença no número de linhas. Depois descreva o que viu.

P2 — as três restrições. Volte à lista da Etapa 0. Para cada um dos três erros da empresa, aponte qual restrição sua o impede.

Um deles talvez não seja impedido pelo banco. Se for o caso, **diga isso** e explique por quê — a pergunta admite essa resposta, e reconhecer o limite da ferramenta vale ponto. Pense: uma restrição de banco consegue impedir que alguém digite o mesmo cliente com uma letra diferente?

P3 — o colega que quer `text` para o valor. Duas coisas para responder, e a maioria só escreve a primeira:

1. o que deixa de funcionar quando o número vira texto;
2. o que fazer com "a combinar". Existe uma forma correta de representar "não tem valor definido", e você usou ela na Etapa 5.

Etapa 8 — Montar a entrega

Seu arquivo `.sql` precisa ter, na ordem:

```
-- Nome completo
-- Projeto de Banco de Dados – Aula 01

-- ===== PARTE 1: estrutura =====
CREATE TABLE cliente (...);
CREATE TABLE tecnico (...);
CREATE TABLE chamado (...);

-- ===== PARTE 2: dados =====
INSERT INTO ...;

-- ===== PARTE 3: consultas =====
-- Q1. Todos os chamados com nome do cliente e do técnico
SELECT ...;
-- Resposta: 8 linhas

-- Q2. Quantos chamados cada técnico atendeu
SELECT ...;
-- Resposta: 3 linhas, sendo uma com zero

-- ===== PARTE 4: perguntas =====
-- P1. ...
-- P2. ...
-- P3. ...
```

Nome do arquivo: `aula01-seusobrenome.sql` . Envie pelo Classroom **antes de sair da sala**.

A correção é por evidência: o comando precisa estar lá, e a resposta precisa bater com o que ele devolveu. Resposta certa sem o comando que a produziu não pontua.

Antes de entregar, confira

- ☐ as três tabelas têm chave primária
- ☐ `chamado` tem chave estrangeira para as outras duas
- ☐ existe pelo menos um `NOT NULL` , um `UNIQUE` , um `CHECK` e um `DEFAULT`
- ☐ o valor cobrado **não** é `real` nem `float`
- ☐ existe um técnico sem chamado, um técnico inativo e um chamado sem valor

- ☐ a Q2 mostra o técnico com zero
- ☐ a Q5 usa a cláusula certa para filtrar contagem
- ☐ as oito consultas têm o resultado anotado em comentário
- ☐ as três perguntas estão respondidas
- ☐ seu nome está no topo do arquivo

Não terminou tudo? Entregue assim mesmo. Parcial vale mais que ausente, e mostra onde você travou — que é a informação que o professor usa para ajustar a próxima aula.