SQL

Consulte a agenda a partir de tabelas relacionadas

Pré-requisitos

Entenda linhas, colunas e identificadores. O exercício usa Python 3 com sqlite3, disponível no ambiente validado, e um banco em memória. O SQL apresentado também usa construções documentadas pelo PostgreSQL. O exercício armazena dias como texto ISO para manter a preparação curta. No projeto, escolha DATE ou um tipo temporal adequado à regra de negócio.

Comece pela granularidade

A pergunta é 'quantas reservas confirmadas cada unidade teve em um dia?'. A saída deve ter uma linha por unidade. A tabela unidades tem uma linha por unidade. A tabela reservas tem uma linha por reserva e referencia a unidade por unidade_id. Essa relação permite reunir eventos de agendamento com o cadastro das unidades.

Escolha o tipo de junção [1]

INNER JOIN retorna combinações que correspondem à condição da junção. LEFT JOIN também preserva linhas da tabela à esquerda sem correspondência, preenchendo colunas da direita com NULL. Parta de unidades e use LEFT JOIN para incluir unidades com zero confirmações.

Filtre na etapa adequada [2]

ON decide quais combinações contam como correspondências. WHERE filtra o resultado da junção. GROUP BY reúne linhas por unidade. COUNT(r.id) ignora o NULL da linha preservada sem reserva e produz zero. ORDER BY torna a apresentação previsível pelo nome. Um filtro após o agrupamento pode usar HAVING, por exemplo para listar só unidades com duas ou mais confirmações.

Evite contagem duplicada

Se você juntar também pagamentos, uma reserva com duas tentativas pode aparecer duas vezes. Confira a granularidade após cada junção. Agregar os pagamentos por reserva antes de juntar resolve casos em que você precisa usar informações desse relacionamento sem contar a mesma reserva repetidamente.

SQL

Exercício resolvido • Conte confirmações por unidade

Enunciado e dados de partida

Produza um relatório para 24/11/2026 com Centro, Sul e Norte. O Centro tem duas confirmações e um cancelamento. O Sul tem um cancelamento no dia e uma confirmação no dia seguinte. O Norte não tem reservas. Mostre também as unidades com total zero.

1. Salve dados.sql em UTF-8

```
CREATE TABLE unidades (
  id INTEGER PRIMARY KEY,
  nome TEXT NOT NULL
);
CREATE TABLE reservas (
  id INTEGER PRIMARY KEY,
  unidade_id INTEGER NOT NULL REFERENCES unidades(id),
  dia TEXT NOT NULL,
  status TEXT NOT NULL
);

INSERT INTO unidades VALUES (1, 'Centro'), (2, 'Sul'), (3, 'Norte');
INSERT INTO reservas VALUES
  (10, 1, '2026-11-24', 'confirmada'),
  (11, 1, '2026-11-24', 'confirmada'),
  (12, 1, '2026-11-24', 'cancelada'),
  (13, 2, '2026-11-24', 'cancelada'),
  (14, 2, '2026-11-25', 'confirmada');
```

2. Salve consulta.sql em UTF-8

```
SELECT u.nome, COUNT(r.id) AS confirmadas
FROM unidades AS u
LEFT JOIN reservas AS r
  ON r.unidade_id = u.id
 AND r.dia = '2026-11-24'
 AND r.status = 'confirmada'
GROUP BY u.id, u.nome
ORDER BY u.nome;
```

Confira a correspondência

A data e o estado fazem parte de ON. Assim o Sul e o Norte continuam no resultado mesmo sem correspondência válida. O agrupamento usa o identificador e o nome para manter separadas eventuais unidades de mesmo nome.

SQL

Execute a consulta e explique os zeros

3. Salve executar.py na mesma pasta

```
from pathlib import Path
import sqlite3

with sqlite3.connect(':memory:') as banco:
    banco.execute('PRAGMA foreign_keys = ON')
    banco.executescript(Path('dados.sql').read_text(encoding='utf-8'))
    sql = Path('consulta.sql').read_text(encoding='utf-8')
    for nome, total in banco.execute(sql):
        print(nome, total)
```

Execute e confira

Use python executar.py nessa pasta. Em sistemas onde o comando é python3, substitua python por python3. O banco existe somente durante a execução. A saída é 'Centro 2', 'Norte 0' e 'Sul 0', uma linha por unidade. No SQLite, o PRAGMA ativa a conferência das chaves estrangeiras para essa conexão.

Caso de erro • Mova as condições para WHERE [3]

Teste trocar o primeiro AND após ON por WHERE. A consulta passa a filtrar também as linhas produzidas sem correspondência. As comparações com NULL não passam no WHERE, e as reservas do Sul não satisfazem os filtros. Apenas Centro 2 aparece. Volte os filtros para ON para restabelecer a intenção.

Variação • Conte com COUNT(*)

Troque COUNT(r.id) por COUNT(*). Centro continua com 2, mas Norte e Sul aparecem com 1, pois cada unidade foi preservada pela junção em uma linha com NULL nos campos de reserva. Reponha COUNT(r.id) e confira os zeros. Os dois testes demonstram mecanismos diferentes de erro de contagem.

Critérios para conferir sua resposta

A saída tem três linhas, com a soma de duas confirmações. Nenhum cancelamento entra no total e a reserva do dia seguinte permanece fora da contagem. Confira o relatório sem os dados do Centro: as três unidades devem continuar aparecendo, agora com zero.

Referências primárias

[1] PostgreSQL • Junções entre tabelas

[2] PostgreSQL • Funções de agregação

[3] SQLite • Processamento de SELECT, JOIN e WHERE

SQL

Conte também os recursos sem reservas

Filtre antes da contagem

Para mostrar todas as salas e quantas reservas estão confirmadas, faça LEFT JOIN das salas com as reservas e inclua o filtro de confirmação na condição ON. Conte uma coluna não nula da tabela de reservas, como o identificador.

Exercício adicional

A sala A tem duas reservas confirmadas. A sala B tem uma cancelada. Que números a consulta deve mostrar?

Resolução

A deve aparecer com 2 e B com 0. COUNT(r.id) ignora o valor nulo da linha produzida para B. Um filtro de confirmação em WHERE eliminaria B do resultado.

Ao assistir ao vídeo

Observe como o lado esquerdo do JOIN é preservado na ausência de correspondência.

Vídeo complementar

SQL: como usar LEFT JOIN