Restrições

Transforme regras de negócio em garantias

Pré-requisitos

Conheça CREATE TABLE e INSERT. Use Python 3.11 ou superior com sqlite3 para rodar o exercício. O recorte é uma agenda de um único dia com sessões fixas de uma hora, iniciadas em horas cheias. Minutos desde meia-noite simplificam o laboratório.

Regras por linha [1]

NOT NULL exige presença. CHECK avalia uma condição, como duração igual a 60 minutos e estado pertencente ao conjunto previsto. Tanto no PostgreSQL quanto no SQLite, uma condição CHECK que resulta em NULL pode passar. Use NOT NULL junto da condição sempre que a existência do valor for obrigatória. Valores padrão ajudam a preencher registros. A restrição protege também os valores fornecidos explicitamente.

Regras entre linhas [2]

Para permitir várias reservas canceladas no histórico e somente uma ativa por sala e horário, use um índice único parcial. Sua chave compara sala e início somente nas linhas cujo estado é ativa. Com horários alinhados e duração fixa, duas sessões na mesma sala só se sobrepõem quando têm o mesmo início.

Separe a proteção do banco da orientação ao usuário

A interface pode ocultar horários ocupados, mas a agenda pode mudar entre a consulta e a gravação. Faça a gravação depender da regra no banco e converta um conflito esperado em uma resposta de negócio. Em uma operação com várias escritas, cuide da transação inteira e preserve o erro original nos registros internos.

Reveja o modelo quando o domínio mudar

Se a agenda passar a aceitar durações arbitrárias, comparar só o início deixa passar interseções como 10h–11h e 10h30–11h30. Em PostgreSQL, um tipo de intervalo com restrição de exclusão pode representar essa regra [1]. Para vários dias, inclua a data ou instantes completos. O esquema do exercício serve ao recorte de um dia.

Restrições

Exercício resolvido • Impeça uma segunda reserva ativa

Enunciado e dados de partida

A sala 7 tem uma reserva ativa das 10h às 11h. Tente criar uma segunda reserva no mesmo horário, criar uma sessão de 30 minutos, omitir o fim e introduzir um estado desconhecido. Depois cancele a primeira reserva e reutilize o horário. Preserve o histórico.

1. Salve agenda.sql em UTF-8

```
CREATE TABLE reservas (
  id INTEGER PRIMARY KEY,
  sala_id INTEGER NOT NULL,
  inicio_min INTEGER NOT NULL,
  fim_min INTEGER NOT NULL,
  estado TEXT NOT NULL CHECK (estado IN ('ativa', 'cancelada')),
  CHECK (inicio_min >= 0 AND fim_min <= 1440),
  CHECK (fim_min = inicio_min + 60 AND inicio_min % 60 = 0)
);
CREATE UNIQUE INDEX uma_reserva_ativa_por_horario
  ON reservas (sala_id, inicio_min)
  WHERE estado = 'ativa';
```

Leia os limites

A faixa diária começa em 0 e termina em 1440 minutos. Um início divisível por 60 garante hora cheia. A diferença fixa de 60 minutos também impede fim anterior ao início. O índice parcial exclui reservas canceladas da regra de unicidade. O tipo INTEGER no SQLite tem afinidade, com regras diferentes de um esquema PostgreSQL. O laboratório usa entradas inteiras ou NULL.

Calcule os dados de partida

10 × 60 = 600 e 11 × 60 = 660. O primeiro caso duplicado tem esses mesmos valores. A duração da reserva 3 é 690 − 660 = 30, violando a condição de 60 minutos.

Restrições

Execute e confira as recusas

2. Salve executar.py na mesma pasta

```
from pathlib import Path
import sqlite3

db = sqlite3.connect(':memory:')
db.executescript(Path('agenda.sql').read_text(encoding='utf-8'))

def reservar(dados):
    db.execute('INSERT INTO reservas VALUES (?, ?, ?, ?, ?)', dados)

reservar((1, 7, 600, 660, 'ativa'))
for tentativa in [
    (2, 7, 600, 660, 'ativa'),
    (3, 7, 660, 690, 'ativa'),
    (4, 7, 660, None, 'ativa'),
    (5, 7, 660, 720, 'desconhecida'),
]:
    try:
        reservar(tentativa)
    except sqlite3.IntegrityError as erro:
        print(tentativa[0], 'recusada:', erro.sqlite_errorname)
    else:
        raise AssertionError('Uma reserva invalida foi aceita')

db.execute("UPDATE reservas SET estado = 'cancelada' WHERE id = 1")
reservar((6, 7, 600, 660, 'ativa'))
print(db.execute('SELECT id, estado FROM reservas ORDER BY id').fetchall())
```

Resultados e critérios

Rode python executar.py. As reservas 2 a 5 devem ser recusadas por unicidade, CHECK de duração, NOT NULL e CHECK de estado, respectivamente. O estado final contém (1, cancelada) e (6, ativa). Após qualquer recusa, a tentativa ausente pode ser confirmada com SELECT.

Variação • Remova a condição do índice

Troque o índice parcial por um índice único sobre todas as linhas e repita o exercício. A tentativa 6 é recusada, pois a reserva cancelada ainda ocupa a chave. Volte à condição WHERE para conservar o histórico e liberar a vaga.

Referências primárias

[1] PostgreSQL • CHECK, NOT NULL e restrições de exclusão

[2] SQLite • Índices parciais únicos

Restrições

Proteja intervalos inválidos no banco

Combine verificações conforme a regra

CHECK (fim > inicio) verifica a ordem dos instantes quando ambos estão preenchidos. No PostgreSQL, uma expressão CHECK com resultado desconhecido passa, então declare NOT NULL para ambos quando o intervalo for obrigatório.

Exercício adicional

Uma reserva com início preenchido e fim nulo foi gravada apesar do CHECK. Que restrição está faltando?

Resolução

Depois de tratar as linhas antigas, aplique NOT NULL em fim e também em inicio se a regra exigir ambos. Teste fim nulo, fim anterior e um intervalo válido.

Ao assistir ao vídeo

Compare as responsabilidades de NOT NULL, UNIQUE e CHECK.

Vídeo complementar

PostgreSQL Constraints Explained (em inglês)