
PENSARBEM
EDITORA
SQL
One
Do primeiro SELECT à aplicação em produção.
Thiago Nogueira

INFRA ONE 11

SQL One — Prévia

Perguntas Certas para Bancos de Verdade

Thiago Nogueira

PENSARBEM

SQL One — Prévia
© 2027 Thiago Nogueira
Pensarbem

Prévia · 1ª edição

Composto em IBM Plex Serif, IBM Plex Sans e IBM Plex Mono. Diagramado por uma pipeline de tipografia programática.

Sumário

Antes de começar 1
A linguagem das perguntas 4
O que este livro não faz 4
Quem aparece 4
Como as consultas aparecem 5

1 A planilha de 1,2 milhão de linhas 9
1.1 Um banco de dados é um conjunto de tabelas 12
1.2 As tabelas da Santa Rita 13
1.3 A primeira consulta 15
1.4 Escolher as colunas 16

2 Uma linguagem de 1974 19
2.1 Antes das tabelas 21
2.2 O artigo de 1970 21
2.3 SEQUEL 22
2.4 Declarar o resultado 23
2.5 De laboratório a padrão 23
2.6 Por que sobreviveu 25

3 Montando a bancada 27
3.1 O que vai ser instalado 29
3.2 Caminho 1: instalar no sistema 30
3.3 Caminho 2: um container 30
3.4 Criar o banco da Santa Rita 31
3.5 Os comandos do psql 32
3.6 Um cliente gráfico 32

4 SELECT e FROM 37
4.1 Expressões 39
4.2 Textos e nomes 39
4.3 Apelidos 39
4.4 DISTINCT 39
4.5 A ordem em que o banco lê 39

5 WHERE 40
5.1 Comparar 43
5.2 AND, OR e NOT 43
5.3 Períodos com data e hora 43
5.4 IN e LIKE 43

6 NULL, o valor que não é valor 44
6.1 Ausente, e não vazio 47
6.2 IS NULL 47
6.3 Três valores 47
6.4 A campanha que perdeu três clientes 47
6.5 COALESCE 47

7 Ordenar e limitar 48
7.1 ORDER BY 51
7.2 Onde fica o NULL 51
7.3 Ordenar por uma expressão 51
7.4 LIMIT e os empates 51
7.5 Paginar 51

8 Funções de linha 52
8.1 Funções de texto 55
8.2 Funções de número 55
8.3 CASE 55
8.4 Converter tipos 55

9 Datas e horas 56
9.1 Os tipos de data e hora 59
9.2 Contas com datas 59
9.3 Os pedaços de uma data 59
9.4 Escrever uma data para leitura 59
9.5 Fuso horário 59

10 Agrupar 60
10.1 Agregar 63
10.2 A média de quem 63
10.3 GROUP BY 63
10.4 O que pode aparecer no SELECT 63
10.5 HAVING 63

11 Do resumo ao relatório 64
11.1 Agrupar por uma expressão 67
11.2 Contar com condição 67
11.3 Subtotais e total 67
11.4 Juntar os textos de um grupo 67

12 Chaves 71
12.1 A chave primária 73
12.2 Números e não CPFs 73
12.3 A chave estrangeira 73
12.4 Uma tabela que aponta para si mesma 73
12.5 Seguir um fio 73

13 JOIN 74
13.1 Juntar duas tabelas 77
13.2 Nomes que se repetem 77
13.3 Três tabelas 77
13.4 O quadro em reais 77
13.5 A vírgula antiga 77

14 As linhas que faltam 78
14.1 LEFT JOIN 81
14.2 Contar o que veio do outro lado 81
14.3 A condição no ON e no WHERE 81
14.4 Achar o que não existe 81
14.5 RIGHT e FULL JOIN 81

15 O faturamento que dobrou 82
15.1 Para quantos 85
15.2 A junção que multiplica 86
15.3 Conferir antes de confiar 88
15.4 Uma tabela com ela mesma 89
15.5 O produto cartesiano 90

16 Subconsultas 94
16.1 Uma consulta como valor 97
16.2 Uma consulta como lista 97
16.3 Existe ou não existe 97
16.4 NOT IN e o NULL 97
16.5 Uma consulta como tabela 97

17 Conjuntos 98
17.1 UNION e UNION ALL 101
17.2 As regras do empilhamento 101
17.3 INTERSECT 101
17.4 EXCEPT 101

18 WITH 102
18.1 Dar nome a uma subconsulta 105
18.2 Usar o mesmo passo duas vezes 105
18.3 WITH RECURSIVE 105
18.4 Como a recursão para 105

19 Funções de janela 106
19.1 Agregar sem agrupar 109
19.2 Numerar e classificar 109
19.3 Olhar a linha vizinha 109
19.4 Acumular 109

20 Do balcão ao diagrama 113
20.1 Uma tabela que é várias 115
20.2 Entidades e identificadores 115
20.3 Cardinalidade 115
20.4 O diagrama 115

21 Tipos de dados 116
21.1 Dinheiro não é float 119
21.2 Inteiros e os seus limites 119
21.3 Texto 119
21.4 Datas que chegaram como texto 119
21.5 Verdadeiro, falso e números que se geram sozinhos 119

22 Normalização 120
22.1 As anomalias 123
22.2 Primeira forma normal 123
22.3 Dependências e a segunda forma normal 123
22.4 Terceira forma normal 123
22.5 Montar e conferir 123
22.6 Quando desnormalizar 123

23 Restrições 124
23.1 CHECK 127
23.2 Nomes para as regras 127
23.3 UNIQUE e as maiúsculas 127
23.4 DEFAULT 127
23.5 O que acontece ao apagar 127

24 O esquema muda 128
24.1 Acrescentar uma coluna 131
24.2 Renomear 131
24.3 Mudar o tipo 131
24.4 Apagar tabelas e colunas 131
24.5 Migrações 131

25 INSERT, UPDATE, DELETE 135
25.1 INSERT 138
25.2 UPDATE 139
25.3 O UPDATE sem WHERE 140
25.4 Atualizar a partir de outra tabela 142
25.5 DELETE 143

26 Transações 146
26.1 Tudo ou nada 149
26.2 Um erro no meio 149
26.3 SAVEPOINT 149

26.4 O que as outras sessões veem 149

27 A madrugada das 3h 150
27.1 O que uma transação vê 153
27.2 REPEATABLE READ 153
27.3 A atualização perdida 153
27.4 O deadlock 153

28 Cargas 154
28.1 COPY 157
28.2 A tabela de carga 157
28.3 ON CONFLICT 157
28.4 MERGE 157

29 Índices 161
29.1 O que o banco faz sem índice 163
29.2 O índice 163
29.3 A ordem das colunas 163
29.4 Condições que o índice entende 163
29.5 O preço de um índice 163

30 Ler um plano 164
30.1 Custos e estimativas 167
30.2 Quando a estimativa erra 167
30.3 Três jeitos de juntar 167
30.4 Ordenar e cortar 167
30.5 Um roteiro de leitura 167

31 Visões, funções e gatilhos 168
31.1 Visões 171
31.2 Visões materializadas 171
31.3 Funções 171
31.4 Gatilhos 171

32 Quem pode ver o quê 172
32.1 Papéis 175
32.2 Privilégios por coluna 175
32.3 Segurança por linha 175
32.4 Injeção de SQL 175
32.5 Dados sensíveis 175

33 Backup 176
33.1 pg_dump 179
33.2 Restaurar 179
33.3 O sábado, consertado 179
33.4 Backup físico e o ponto no tempo 179

33.5 O ensaio 179

34 PostgreSQL e MySQL 183
34.1 A mesma base no MySQL 185
34.2 Textos que se comparam diferente 185
34.3 Números e o NULL 185
34.4 O que muda na escrita 185
34.5 Transações e estrutura 185

35 SQLite, SQL Server e Oracle 186
35.1 SQLite: um banco num arquivo 189
35.2 Tipos que não prometem 189
35.3 Chaves desligadas e grupos frouxos 189
35.4 SQL Server e Oracle 189

36 Quinze de março 190
36.1 O estoque, lote por lote 193
36.2 A margem de cada loja 193
36.3 Números que se conferem 193

A base da Santa Rita 194
36.4 As tabelas 197
36.5 Os dados 197
36.6 A amostra da Rede Bem-Estar 197
36.7 Criar o banco 197
36.8 A base no MySQL 197

Referência rápida 198
36.9 A ordem de leitura 201
36.10 O esqueleto de uma consulta 201
36.11 Condições 201
36.12 Junções 201
36.13 Agregação e janelas 201
36.14 Escrever dados 201
36.15 Estrutura 201
36.16 Transações 201
36.17 Comandos do psql 201
36.18 Cinco bancos 201
36.19 As armadilhas e as defesas 201

Onde praticar 202
36.20 As três bancadas 205
36.21 Recomeçar a base 205
36.22 Trabalhar em cópias 205
36.23 Uma base maior 205

Respostas 206

Glossário 214

Antes de começar

Um e-mail com três pedidos, trinta e oito milhões de reais esperando a resposta, um milhão de vendas guardadas — e uma única pessoa na empresa que sabe perguntar alguma coisa a elas.

Assunto: Financiamento — documentação para a reunião de 15/03

> *Prezada Cláudia, para a análise de crédito precisamos de três informações: a posição do estoque, lote por lote, indicando o que está vencido; a margem de cada loja; e o histórico de vendas dos últimos três anos. Os números serão conferidos por amostragem com os documentos fiscais.*

O e-mail tinha seis linhas, e dele dependia um financiamento de **38 milhões de reais**. Quem o recebeu foi a Cláudia Menezes, diretora financeira da **Drogarias Santa Rita** — 48 farmácias no centro-oeste de Minas Gerais, um centro de distribuição em Itaúna, quase seiscentos funcionários. O dinheiro serviria para comprar a Rede Bem-Estar, onze lojas no sul do estado, num acordo fechado em setembro de 2026. A reunião no banco ficou marcada para **15 de março de 2027**.

As três respostas existiam. Cada cupom vendido numa das 48 lojas — a dipirona, a fralda, o remédio de pressão, a caixa de antibiótico com a receita retida no balcão — virava linhas num banco de dados: a loja, a hora, o cliente, quem atendeu, cada produto, de que lote saiu, quanto custou e quanto foi o desconto. Em três anos, um milhão de vendas e mais de dois milhões e meio de itens. Os números do banco estavam todos lá.

O problema era outro: na empresa toda, uma pessoa só sabia perguntar alguma coisa àqueles dados. O Seu Osvaldo montou o banco em 2011 "para tirar relatório" e, desde então, tira. O resto da Santa Rita decide com uma planilha que o financeiro exporta toda segunda-feira: um milhão e duzentas mil linhas, que travam o computador de quem abre, e uma fórmula de margem que ninguém, nem o autor, consegue mais explicar.

Quando a Cláudia perguntou quanto tempo a planilha levaria para responder à primeira pergunta do banco, o Seu Osvaldo pensou um pouco antes de dizer:

— Depende de quem for perguntar.

Em dezembro, chega à Santa Rita a Bruna, analista de dados que faz milagre em planilha e nunca escreveu uma consulta. Até 15 de março, ela vai ter de aprender a perguntar.

A Santa Rita, o financiamento e a fórmula de margem são inventados. As consultas e as saídas, não: todas rodaram de verdade.

A linguagem das perguntas

Os números que o banco quer estão no banco de dados da Santa Rita. Para tirá-los de lá, existe uma linguagem que tem mais de cinquenta anos, funciona em quase todo banco de dados que você vai encontrar — o da farmácia, o do aplicativo de banco, o do sistema da prefeitura, o do seu celular — e quase não mudou na essência desde os anos 1980: **SQL**.

SQL não é uma linguagem para escrever programas. É uma linguagem para fazer **perguntas** a dados guardados em tabelas: quais lojas venderam mais em dezembro, quantos lotes vencem antes de abril, quanto cada cliente gastou no ano. Você diz *o que* quer; o banco de dados decide *como* buscar. É essa troca — você descreve o resultado, outro programa acha o caminho — que faz uma consulta de três linhas responder, em meio segundo, o que uma planilha de um milhão de linhas não responde em uma tarde.

O que este livro não faz

Ele não ensina SQL como lista de comandos. Cada assunto aparece quando a Santa Rita precisa dele, numa pergunta de verdade, com a consulta ingênua primeiro, a saída que ela produz e o motivo de essa saída estar errada — ou certa por acaso. Boa parte dos erros caros com SQL não dá mensagem de erro: a consulta roda, devolve um número, e o número está errado. O livro dedica a eles o mesmo espaço que dedica às consultas que funcionam.

E ele não pula etapas. Nenhuma palavra aparece numa consulta antes de ser explicada. Os exemplos rodam numa base de dados que você mesmo vai criar, com scripts SQL que estão no fim do livro, e que dá os mesmos números que aparecem no texto. É um recorte do banco da Santa Rita que o Seu Osvaldo separou para a Bruna: três lojas e as vendas de dezembro de 2026. Pequeno de propósito — cada resultado pode ser conferido olhando as linhas.

Quem aparece

Bruna Teixeira tem 25 anos e chegou à Santa Rita em dezembro, como analista de dados. Faz milagre numa planilha, sabe montar uma tabela dinâmica de olhos fechados e nunca escreveu uma consulta. Quer entender por que o número dá o que dá.

Seu Osvaldo Ribeiro tem 62 anos e se aposentou no ano passado, no papel. Continua vindo três dias por semana. Sabe de cabeça o nome das tabelas, o que cada coluna quer dizer e qual delas mente. Nunca escreveu uma linha de documentação.

Dra. Lúcia Amaral é a farmacêutica responsável técnica da rede. Responde à vigilância sanitária por cada caixa de remédio controlado e por cada lote vencido numa prateleira. Não aceita "mais ou menos".

Cláudia Menezes é a diretora financeira. Quer o número na segunda, às oito da manhã, e desconfia de qualquer número que ela não consegue reproduzir.

Rafael Costa é o gerente de TI. Tem um painel novo a cada trimestre e promete prazos que ainda não sabe quanto custam.

Dona Zilda gerencia a loja 1, no centro de Divinópolis, há dezenove anos. Conhece cada cliente pelo nome e cada erro do sistema por um apelido.

Como as consultas aparecem

Uma consulta aparece assim:

```
SELECT 48 AS lojas, 'Divinópolis' AS sede;
```

E o resultado aparece logo abaixo, exatamente como o **psql**, o programa de terminal do PostgreSQL, o mostra:

```
 lojas |    sede
-------+-------------
    48 | Divinópolis
(1 row)
```

O resultado de uma consulta é sempre uma **tabela**: colunas com nome, em cima, e linhas embaixo. A última linha, entre parênteses, é o psql contando quantas linhas o resultado tem; ela não faz parte dos dados. As mensagens do psql e do PostgreSQL aparecem em inglês, porque é assim que eles vêm instalados na imagem oficial; uma mensagem de erro em inglês também é a que você vai colar numa busca quando precisar de ajuda.

Os comandos que você digita no terminal aparecem com o nome do programa na frente, sem o símbolo do prompt:

```
psql -d santarita
```

Scripts — arquivos com várias instruções SQL, como os que criam a base — aparecem com o nome do arquivo em cima.

> **O ESSENCIAL**
> Toda consulta que aparece com a sua saída foi executada de verdade — no PostgreSQL, sobre a base que os scripts do apêndice A1 criam, e, na parte final, também no MySQL e no SQLite —, e a saída mostrada é a saída real. Quando uma consulta se comportar de outro jeito em outro banco, o texto avisa; o SQL Server e o Oracle aparecem lado a lado na sintaxe.

A planilha da segunda-feira continua circulando. Até março, é com ela que a Santa Rita decide o que comprar.

PARTE 1

A planilha de 1,2 milhão de linhas

Quarenta e oito farmácias, uma planilha que trava o Excel e um banco que alguém montou em 2011 para tirar relatório.

NESTA PARTE

1 A planilha de 1,2 milhão de linhas
2 Uma linguagem de 1974
3 Montando a bancada

CAPÍTULO

1

A planilha de 1,2 milhão de linhas

Uma margem de 31% que ninguém consegue refazer, um arquivo que trava o Excel de quem abre, e um homem que diz que a resposta está numa tabela de três linhas.

AO FIM DESTE CAPÍTULO
Entender o que é um banco de dados relacional — tabelas, colunas, linhas e tipos —, conhecer as tabelas da Santa Rita e escrever as primeiras consultas com SELECT e FROM.

ENQUANTO ISSO, NA EMPRESA **Segunda, oito da manhã**

Na segunda, 4 de janeiro, a Cláudia Menezes abriu a reunião de diretoria com um número.

— Margem bruta de dezembro: 31%. Melhor mês do ano.

A Bruna, três semanas de empresa, recebeu a tarefa de refazer a conta para a apresentação do banco. Pediu a planilha. Veio por e-mail: um arquivo de 410 megabytes, um milhão e duzentas mil linhas, catorze abas e uma coluna chamada `MARGEM_FINAL_v3`, com uma fórmula de três linhas que terminava em `PROCV`.

O computador dela levou onze minutos para abrir o arquivo. Quando ela mexeu num filtro, a tela ficou branca, e o Excel perguntou se ela queria esperar ou fechar.

O Seu Osvaldo, na mesa ao lado, viu a tela branca e girou a cadeira.

— De onde sai essa planilha? — perguntou ela.

— Do banco. O financeiro exporta tudo toda segunda e faz a conta lá dentro. — Ele puxou o teclado dela. — Vamos perguntar direto para quem tem os dados.



FIGURA 1.1 Uma planilha de um milhão de linhas é um banco ao qual ninguém consegue perguntar.

1.1 Um banco de dados é um conjunto de tabelas

A planilha da segunda-feira tem um problema que não é o tamanho: ela mistura tudo numa folha só. Cada linha é um item vendido, e na mesma linha estão o nome da loja, a cidade, o nome do produto, o preço, o nome do cliente. A loja do Centro aparece cento e tantas mil vezes, escrita de novo em cada linha. Se ela mudar de nome, são cento e tantas mil células para corrigir.

Um **banco de dados relacional** guarda cada coisa num lugar só. As lojas ficam numa tabela de lojas, uma linha por loja. Os produtos, numa tabela de produtos. As vendas, numa tabela de vendas, e cada venda diz de que loja é pelo número da loja, não pelo nome. A Santa Rita tem onze tabelas assim:

TABELA	UMA LINHA PARA CADA
`lojas`	loja da rede
`funcionarios`	pessoa que trabalha na rede
`categorias`	categoria de produto
`fornecedores`	laboratório ou distribuidora
`produtos`	produto do catálogo
`lotes`	lote de fabricação de um produto
`estoque`	lote guardado numa loja
`clientes`	cliente cadastrado
`vendas`	cupom fiscal
`itens_venda`	produto vendido num cupom
`receitas`	receita de remédio controlado

TABELA 1.1 As tabelas da Drogarias Santa Rita.

Uma tabela tem **colunas**, cada uma com um nome e um **tipo** — número inteiro, texto, data, valor em dinheiro —, e tem **linhas**, cada uma com um valor para cada coluna. A tabela de lojas, completa:

lojas

id	nome	cidade	uf	inaugurada	metragem
1	Centro	Divinópolis	MG	1978-05-02	150
2	Bom Pastor	Divinópolis	MG	2006-01-26	260
3	Itaúna	Itaúna	MG	2012-08-15	**NULL**

FIGURA 1.2 A tabela lojas: seis colunas, três linhas. Destacadas, a coluna cidade, a linha do Bom Pastor e um campo sem valor, a metragem de Itaúna.

Cada linha é uma loja, e cada coluna, uma informação sobre todas as lojas. O cruzamento de uma linha com uma coluna é um **valor**: a cidade do Bom Pastor é Divinópolis. Ao contrário de uma planilha, o tipo da coluna é uma regra: numa coluna de data só entra data, e um banco de dados recusa "31/02" ou "a combinar". E um campo pode ficar sem valor, como a metragem de Itaúna, que ninguém informou; o banco chama essa ausência de **NULL**.

Banco de dados relacional Um programa que guarda dados em tabelas com colunas de tipo definido, ligadas umas às outras por valores em comum, e que responde a perguntas escritas em SQL. O **PostgreSQL**, que a Santa Rita usa, é um deles; o MySQL, o SQL Server, o Oracle e o SQLite são outros.

O banco de dados roda num computador, o **servidor**, e quem quer os dados se conecta a ele com um programa **cliente**. O cliente mais simples do PostgreSQL é o **psql**, que roda num terminal: você digita uma consulta, ele manda para o servidor e mostra o resultado.

1.2 As tabelas da Santa Rita

O Seu Osvaldo abriu o psql no computador da Bruna e se conectou ao banco `santarita`, que ele preparou para ela: um recorte com três lojas e as vendas de dezembro.

```
psql -d santarita
```

O psql mostra um **prompt** com o nome do banco e espera:

```
santarita=#
```

As linhas que começam com barra invertida são comandos do próprio psql, e não SQL. O `\dt` lista as tabelas do banco:

NORUN sql

```
\dt
```

```
               List of tables
 Schema |     Name     | Type  |  Owner
--------+--------------+-------+----------
 public | categorias   | table | postgres
 public | clientes     | table | postgres
 public | estoque      | table | postgres
 public | fornecedores | table | postgres
 public | funcionarios | table | postgres
 public | itens_venda  | table | postgres
 public | lojas        | table | postgres
 public | lotes        | table | postgres
 public | produtos     | table | postgres
 public | receitas     | table | postgres
 public | vendas       | table | postgres
(11 rows)
```

Para ver as colunas de uma tabela, `\d` e o nome dela:

NORUN sql

```
\d lojas
```

```
                      Table "public.lojas"
    Column   |     Type     | Collation | Nullable | Default
-------------+--------------+-----------+----------+---------
 id          | integer      |           | not null |
 nome        | text         |           | not null |
 cidade      | text         |           | not null |
 uf          | character(2) |           | not null |
 inaugurada  | date         |           | not null |
 metragem    | integer      |           |          |
Indexes:
    "lojas_pkey" PRIMARY KEY, btree (id)
```

A tabela `lojas` tem seis colunas. `id` é um número inteiro que identifica cada loja; `nome` e `cidade` são texto; `inaugurada` é uma data; `metragem`, um inteiro. `character(2)` é um texto de exatamente dois caracteres, a sigla do estado. A coluna *Nullable* diz quais colunas podem ficar sem valor: todas são `not null`, menos a metragem. As linhas de baixo, *Indexes*, falam de uma estrutura que o banco mantém para achar cada loja pelo `id` depressa; por ora, basta saber que ela existe e que o `\d` a mostra.

Como as tabelas se ligam

Uma venda não guarda o nome da loja: guarda o número dela, na coluna `loja_id`. O item de uma venda guarda o número da venda e o do produto. Cada número desses é um fio entre duas tabelas:

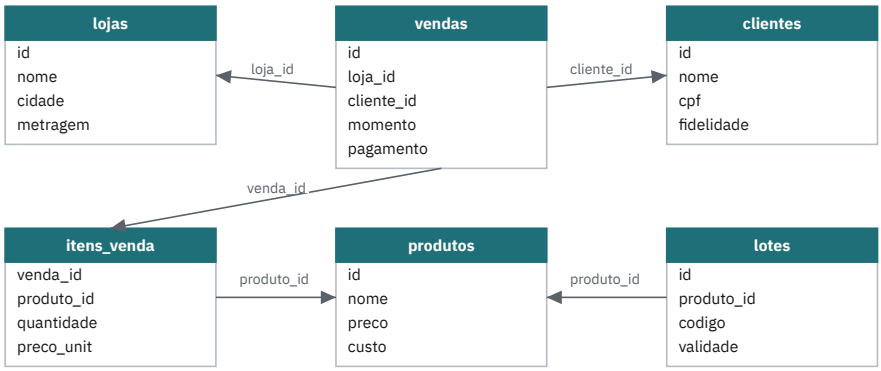


FIGURA 1.3 Seis das onze tabelas da Santa Rita, com algumas das colunas. Cada linha entre duas tabelas é uma coluna que guarda o número de uma linha da outra.

A venda 1 tem `loja_id` igual a 1: foi feita no Centro. Para saber o nome da loja de uma venda, o banco precisa olhar as duas tabelas, e uma pergunta que cruza tabelas é uma consulta como outra qualquer — só que maior. As primeiras consultas olham uma tabela de cada vez.

1.3 A primeira consulta

Para ver as lojas, a Bruna escreveu, com o Seu Osvaldo ditando:

```
SELECT * FROM lojas;
```

```
 id |    nome    |   cidade    | uf | inaugurada | metragem
----+------------+-------------+----+------------+----------
  1 | Centro     | Divinópolis | MG | 1978-05-02 |      150
  2 | Bom Pastor | Divinópolis | MG | 2006-01-26 |      260
  3 | Itaúna     | Itaúna      | MG | 2012-08-15 |
(3 rows)
```

A consulta tem duas partes. `FROM lojas` diz **de onde** vêm os dados: da tabela `lojas`. `SELECT *` diz **o que** mostrar: o asterisco quer dizer "todas as colunas". O ponto e vírgula no fim avisa ao psql que a consulta terminou e pode ser enviada ao servidor.

O resultado é outra tabela, com as colunas na ordem em que foram criadas. A loja de Itaúna não tem metragem: o campo está vazio. É um

valor ausente, e não um zero — a loja tem algum tamanho, só não foi informado.

As palavras `SELECT` e `FROM` são **palavras-chave** da linguagem, e o SQL não diferencia maiúsculas de minúsculas nelas: `select * from lojas;` dá o mesmo resultado. Este livro escreve as palavras-chave em maiúsculas e os nomes de tabelas e colunas em minúsculas, para que se veja de longe o que é da linguagem e o que é da Santa Rita.

1.4 Escolher as colunas

Quase nunca se quer todas as colunas. No lugar do asterisco, entram os nomes das colunas, separados por vírgula, na ordem em que devem aparecer:

```sql
SELECT nome, preco FROM produtos;
```

```
               nome              | preco
---------------------------------+--------
 Losartana 50 mg 30 comp.        |  22.27
 Dipirona 500 mg 10 comp.        |   6.49
 Amoxicilina 500 mg 21 cáps.     |  24.97
 Clonazepam 2 mg 30 comp.        |  18.90
 Protetor solar FPS 50 120 ml    |  68.57
 Sérum vitamina C 30 ml          | 129.90
 Fralda infantil M pacote 60     |  84.23
 Fralda geriátrica pacote 30     |  72.40
 Loratadina 10 mg 12 comp.       |  14.80
 Aferição de pressão arterial    |   5.00
(10 rows)
```

O psql alinha o texto à esquerda e os números à direita, e o preço aparece sempre com duas casas, porque a coluna foi criada como valor em dinheiro com duas casas decimais. O ponto é o separador decimal: SQL escreve números como o inglês.

Um nome de coluna errado não passa:

QUEBRADO sql

```sql
SELECT nome, preço FROM produtos;
```

```
ERROR:  column "preço" does not exist
LINE 1: SELECT nome, preço FROM produtos;
                     ^
HINT:  Perhaps you meant to reference the column
"produtos.preco".
```

A mensagem tem três partes. A primeira linha, `ERROR`, diz o que deu errado: *column does not exist*, a coluna não existe. A segunda e a terceira mostram a consulta e apontam, com o acento circunflexo, onde está o problema. A última, `HINT`, é uma dica: o PostgreSQL percebeu que existe uma coluna de nome parecido. A coluna se chama `preco`, sem cedilha; o banco sugere, mas não troca sozinho. As mensagens vêm em inglês, e ler as três partes, na ordem, resolve a maioria dos erros.

Esquecer o ponto e vírgula também acontece, e não dá erro: o psql só espera. O prompt muda de `=#` para `-#`, que quer dizer "continuo esperando o resto da consulta".

```
santarita=# SELECT nome FROM lojas
santarita-#
```

Basta digitar `;` e Enter para ele enviar.

> **• O ESSENCIAL**
> Um banco de dados relacional guarda cada coisa numa tabela própria, com colunas de nome e tipo definidos e uma linha por registro. O psql é o cliente de terminal do PostgreSQL: `\dt` lista as tabelas e `\d` mostra as colunas de uma delas. A consulta mais simples é `SELECT` seguido das colunas — ou `*`, para todas — e `FROM` seguido da tabela, terminada por ponto e vírgula. O resultado é sempre uma tabela, e um campo vazio é um valor ausente, não um zero.

ENQUANTO ISSO, NA EMPRESA **Três linhas**

A Bruna olhou as três linhas da tabela de lojas e depois a planilha, ainda branca, na outra metade da tela.

— Então a planilha é isso tudo junto, repetido.

— Repetido e copiado e colado — disse o Seu Osvaldo. — Com uma conta no meio que alguém fez em 2019 e ninguém mais sabe se está certa.

— E os 31%?

Ele deu de ombros.

— A Cláudia vai querer o número. Você vai dar o número. Mas primeiro aprende a perguntar.

RESUMO

- Um banco relacional guarda cada coisa numa tabela: colunas com nome e tipo, uma linha por registro; as tabelas se ligam por valores, como o número da loja numa venda.
- O servidor guarda os dados; o psql é um cliente de terminal. `\dt` lista as tabelas; `\d tabela` mostra as colunas.
- `SELECT colunas FROM tabela;` — o asterisco pede todas as colunas; palavras-chave não diferenciam maiúsculas de minúsculas.
- Um nome de coluna errado dá `ERROR: column ... does not exist`, com a posição marcada; sem ponto e vírgula, o psql espera o resto.
- Um campo vazio no resultado é um valor ausente, não zero.

1 Escreva a consulta que mostra o nome, a cidade e o programa de fidelidade de cada cliente. Quantos clientes aparecem sem programa?

2 ·· Mostre o número, a hora e a forma de pagamento de cada venda. Olhando o resultado, em que ordem os cupons parecem estar, e por que não dá para contar com essa ordem?

3 ·· A consulta abaixo tem dois erros. Rode, leia a mensagem, conserte o primeiro, rode de novo e conserte o segundo.

```
SELECT nome, cidad FROM cliente;
```

CAPÍTULO

2

Uma linguagem de 1974

Um matemático da IBM que achava que ninguém deveria precisar saber onde os dados ficam, dois pesquisadores que queriam uma linguagem que um contador conseguisse ler, e um nome que teve de mudar porque já era de uma fábrica de aviões.

AO FIM DESTE CAPÍTULO
Entender de onde vêm as tabelas e a linguagem SQL, o que o modelo relacional promete, por que SQL é declarativa, e por que uma linguagem de cinquenta anos continua sendo a forma mais comum de perguntar coisas a dados.

ENQUANTO ISSO, NA EMPRESA **Tecnologia dos anos 70**

Na terça, 5 de janeiro, o Rafael levou à sala da TI um consultor de camisa social e tênis branco. Diego Prado, de uma empresa de "dados e inteligência artificial" de Belo Horizonte.

— A Santa Rita está presa num banco relacional dos anos setenta — disse o Diego, ligando o projetor. — O mercado todo foi para data lake. Vocês jogam tudo num lago, o modelo de IA lê, e ninguém precisa mais escrever consulta.

O Seu Osvaldo, no fundo da sala, tomou um gole de café.

— Dos anos setenta não — disse ele. — De 1974. O banco é de 2011.

Depois da reunião, a Bruna foi até a mesa dele.

— Ele tem razão? SQL é velho?

— É mais velho que eu na empresa — disse o Seu Osvaldo. — E que o Excel, e que a internet. — Ele abriu uma gaveta e tirou um livro com a lombada descolando. — Por isso mesmo funciona em todo lugar. Pergunta para ele em que linguagem o lago dele responde as consultas.

2.1 Antes das tabelas

Nos anos 1960, um banco de dados era um programa que guardava registros e os **ligava por ponteiros**: o registro de um cliente apontava para o primeiro pedido, que apontava para o segundo, que apontava para o terceiro. O IMS, da IBM, de 1968, organizava tudo numa hierarquia de pai e filho — foi feito para controlar as peças do programa Apollo. Os bancos do modelo de **rede**, padronizados pelo grupo CODASYL a partir de 1969, permitiam ligações em várias direções.

Os dois modelos funcionavam, e alguns funcionam até hoje. Mas tinham um custo escondido: para perguntar alguma coisa, o programador precisava saber o caminho. "Quais clientes de Itaúna compraram antibiótico em dezembro?" virava um programa que começava num registro, seguia ponteiros numa ordem definida e ia guardando o que encontrava no caminho. Se a pergunta mudasse, o programa mudava. Se a forma de guardar os dados mudasse — um índice novo, uma ligação a mais —, os programas antigos quebravam.

2.2 O artigo de 1970

Em junho de 1970, Edgar F. Codd, um matemático inglês que trabalhava no laboratório de pesquisa da IBM em San José, na Califórnia, publicou

um artigo de onze páginas na revista *Communications of the ACM*: "A Relational Model of Data for Large Shared Data Banks". A ideia central cabia numa frase: quem usa os dados não deveria precisar saber como eles estão guardados.

Para isso, Codd propôs representar todos os dados de um único jeito: como **relações** — o nome matemático do que hoje chamamos de tabela. Uma relação tem colunas com nome e um conjunto de linhas, cada linha com um valor para cada coluna. Nada de ponteiros: um pedido não aponta para o cliente; ele tem uma coluna com o código do cliente, e a ligação entre os dois é só a igualdade entre esse código e o código que está na tabela de clientes.

Relação O nome que Codd deu, tirado da matemática dos conjuntos, ao que os bancos de dados chamam de **tabela**: um conjunto de linhas, todas com as mesmas colunas. Por ser um conjunto, as linhas não têm uma ordem própria, e duas linhas idênticas não fazem sentido.

Duas consequências dessa escolha atravessam este livro todo. A primeira: como uma tabela é um conjunto, ela **não tem ordem**. Se você quer as linhas numa ordem, precisa pedir. A segunda: como as ligações são feitas por valores, e não por ponteiros, qualquer pergunta que se escreva com esses valores pode ser respondida — inclusive as que ninguém tinha previsto quando os dados foram guardados.

Codd também mostrou que as perguntas a um conjunto de tabelas podiam ser escritas com um pequeno número de operações — escolher linhas, escolher colunas, combinar duas tabelas por um valor em comum, unir, subtrair —, a **álgebra relacional**. Era matematicamente elegante e quase ninguém fora da pesquisa conseguia ler.

2.3 SEQUEL

Em 1974, dois colegas de Codd no mesmo laboratório, Donald Chamberlin e Raymond Boyce, apresentaram uma linguagem para escrever essas perguntas com palavras em inglês em vez de símbolos matemáticos: **SEQUEL**, *Structured English Query Language*. A meta declarada era que uma pessoa sem formação em programação — um contador, um analista de negócio — conseguisse ler uma consulta e entender o que ela pedia.

Boyce morreu naquele mesmo ano, aos 27 anos. Chamberlin continuou o trabalho dentro do **System R**, o projeto da IBM que, entre 1974 e 1979, construiu um banco de dados relacional de verdade para provar

que a ideia de Codd podia ser rápida. Foi no System R que apareceu uma peça que todo banco de dados moderno tem: o **otimizador**, a parte do banco que lê a pergunta e escolhe, entre vários caminhos possíveis, o mais barato para respondê-la.

No meio do caminho, a linguagem teve de trocar de nome: "SEQUEL" já era marca registrada de uma fabricante de aviões inglesa, a Hawker Siddeley. Ficou **SQL** — que muita gente ainda pronuncia "síquel", em inglês, por causa do nome antigo, e que muita gente pronuncia letra por letra. As duas pronúncias estão certas.

2.4 Declarar o resultado

A diferença entre SQL e o jeito como a Bruna trabalhava até agora é de natureza, e vale a pena vê-la antes da primeira consulta. Para achar, na planilha da segunda-feira, os produtos que custam mais de cem reais, ela segue passos: ordena a coluna de preço, desce até o primeiro valor acima de cem, seleciona dali até o fim, copia os nomes para outra aba. Um programa escrito numa linguagem comum faria o mesmo: percorrer a lista do começo ao fim, um produto de cada vez, testar o preço, guardar o nome. Quem escreve decide **como** achar.

Em SQL, você descreve **o que** quer — os nomes dos produtos com preço acima de cem —, e o banco de dados decide como achar. Se houver um índice por preço, ele usa o índice; se a tabela estiver dividida em vários discos, ele lê os discos ao mesmo tempo; se os dados mudarem de lugar amanhã, a mesma pergunta continua funcionando. É isso que se chama de linguagem **declarativa**, e foi essa a promessa do artigo de 1970: o programa diz o que quer, não onde os dados estão.

A consequência prática é que uma consulta SQL não tem passos. Ela não é lida de cima para baixo como um programa; é lida por completo, como uma descrição. Quem já programou estranha isso no começo, e a maior parte dos erros dos primeiros meses vem de ler uma consulta como se ela fosse uma lista de passos.

2.5 De laboratório a padrão

A IBM demorou a vender o que tinha inventado. Quem chegou primeiro ao mercado foi uma empresa pequena da Califórnia, fundada por Larry Ellison, Bob Miner e Ed Oates, que leu os artigos do System R e lançou em 1979 o primeiro banco de dados relacional comercial com SQL. O

produto se chamava Oracle, e a empresa acabaria adotando o nome. A IBM lançou o SQL/DS em 1981 e o DB2 em 1983.

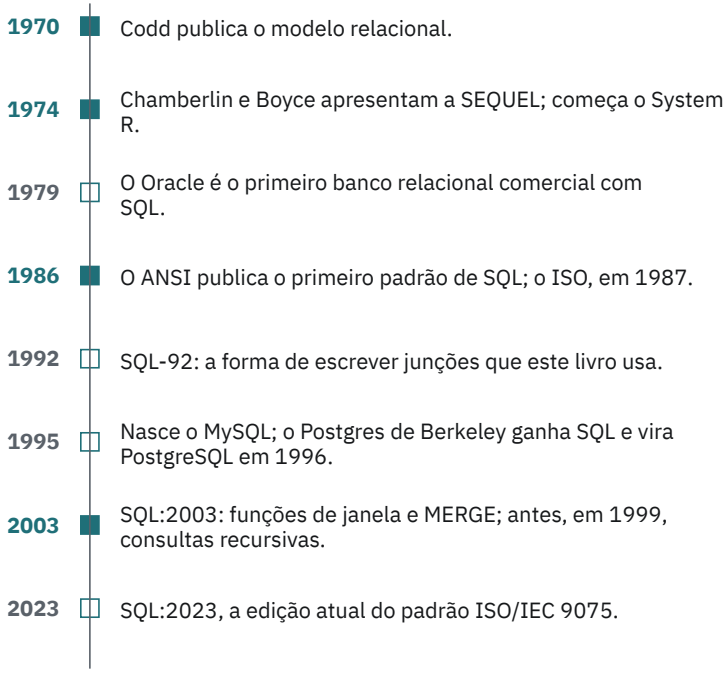


FIGURA 2.1 Do artigo de Codd ao padrão de 2023.

Em 1986, o instituto americano de normas, o ANSI, publicou o primeiro padrão da linguagem, e no ano seguinte a organização internacional, a ISO, adotou o mesmo texto. Desde então, o padrão ISO/IEC 9075 ganhou várias edições, e cada uma acrescentou coisas que os bancos já faziam cada um do seu jeito: a forma de escrever junções de 1992, as consultas recursivas e os gatilhos de 1999, as funções de janela de 2003, o JSON de 2016.

O padrão é um documento, não um programa. Nenhum banco de dados implementa o padrão completo, e todos implementam coisas que o padrão não tem. Por isso o mesmo SQL básico roda no PostgreSQL, no MySQL, no SQLite, no SQL Server e no Oracle, e uma consulta mais sofisticada pode precisar de ajustes de um para o outro. Este livro escreve as consultas na forma do padrão sempre que o PostgreSQL aceita, e diz o nome do banco quando uma sintaxe é de um banco só.

2.6 Por que sobreviveu

De tempos em tempos alguém anuncia o fim do SQL. No fim dos anos 2000, a onda foi o **NoSQL**: bancos de documentos, de chave e valor, de grafos, feitos para escalar em muitos servidores e livres do esquema fixo das tabelas. Vários deles resolvem bem problemas que um banco relacional resolve mal, e continuam em uso. Mas, uma década depois, quase todos tinham ganhado alguma linguagem de consulta parecida com SQL, e os sistemas de análise de dados mais usados — os que leem os "lagos" de que o Diego falou — respondem às perguntas em SQL.

A razão é a mesma de 1970. Uma pergunta escrita em SQL não depende de como os dados estão guardados, e por isso sobrevive a mudanças de servidor, de disco, de formato e de fornecedor. As tabelas da Santa Rita estão num PostgreSQL de 2011; a consulta que a Bruna escrever este mês vai rodar no banco que vier depois dele.

> • **O ESSENCIAL**
> O modelo relacional, de Codd, em 1970, representa todos os dados como tabelas — relações —, ligadas por valores e não por ponteiros, e sem ordem própria. A SQL nasceu da SEQUEL de Chamberlin e Boyce, em 1974, no System R da IBM, para que perguntas a essas tabelas pudessem ser escritas em palavras. É uma linguagem declarativa: descreve o resultado, e o otimizador do banco escolhe o caminho. O padrão ISO/IEC 9075, de 1986/1987 até a edição de 2023, define o núcleo comum que PostgreSQL, MySQL, SQLite, SQL Server e Oracle implementam, cada um com os seus acréscimos.

ENQUANTO ISSO, NA EMPRESA **A linguagem do lago**

Na sexta, o Diego mandou a proposta por e-mail: trinta e duas páginas, um diagrama colorido com um lago no meio e o preço na última folha.

A Bruna leu até a página onze, onde estava a arquitetura. Embaixo do lago, numa caixa pequena, a ferramenta que os analistas usariam para consultar os dados.

— "Consultas em SQL padrão" — leu ela em voz alta.

O Seu Osvaldo não tirou os olhos da tela.

— Então começa a aprender agora — disse. — Que o lago custa duzentos e quarenta mil, e o SQL é de graça.

RESUMO

- Os bancos dos anos 1960 ligavam registros por ponteiros; perguntar exigia saber o caminho, e mudar a forma de guardar quebrava os programas.
- Codd, em 1970, propôs representar tudo como relações — tabelas —, ligadas por valores; uma tabela é um conjunto e não tem ordem.
- SEQUEL, de Chamberlin e Boyce, em 1974, virou SQL no System R, onde nasceu o otimizador; o Oracle foi o primeiro produto comercial, em 1979.
- SQL é declarativa: descreve o resultado, não os passos.
- O padrão ISO/IEC 9075 vai de 1986/1987 a 2023; os bancos implementam o núcleo comum e acrescentam o seu.

1 Pelo que o capítulo conta, o que acontece com um programa escrito para um banco do modelo de rede quando se acrescenta uma ligação nova entre dois tipos de registro? E com uma consulta SQL, quando se acrescenta um índice?

2 Codd chamou as tabelas de relações porque elas são conjuntos. Dê dois exemplos de consequências práticas disso para quem vai escrever consultas.

CAPÍTULO

3

Montando a bancada

Um servidor de banco de dados no seu computador em dez minutos, dois scripts para criar a Santa Rita, e os seis comandos do psql que você vai usar todo dia.

AO FIM DESTE CAPÍTULO
Instalar o PostgreSQL — direto no sistema ou num container —, criar o banco da Santa Rita com os scripts do apêndice A1, e usar o psql e um cliente gráfico para conversar com ele.

ENQUANTO ISSO, NA EMPRESA **O computador do Seu Osvaldo**

Na quarta, 6 de janeiro, a Bruna pediu ao Rafael acesso ao banco da Santa Rita.

— Produção não — disse o Rafael. — Você aprende no seu computador. Depois a gente vê permissão.

O Seu Osvaldo concordou, o que era raro.

— Ele está certo. Quem aprende SQL em produção um dia apaga uma tabela em produção. — Ele mandou para ela dois arquivos. — Instala o PostgreSQL aí e roda esses dois. É o recorte de dezembro.

3.1 O que vai ser instalado

Para seguir o livro, você precisa de um **servidor PostgreSQL**, que guarda os dados e responde às consultas, e de um cliente para mandar as consultas a ele. O servidor e o cliente podem estar no mesmo computador, e na bancada estão. O livro usa o **PostgreSQL 18**; as consultas da maior parte dos capítulos rodam igual nas versões a partir da 13.

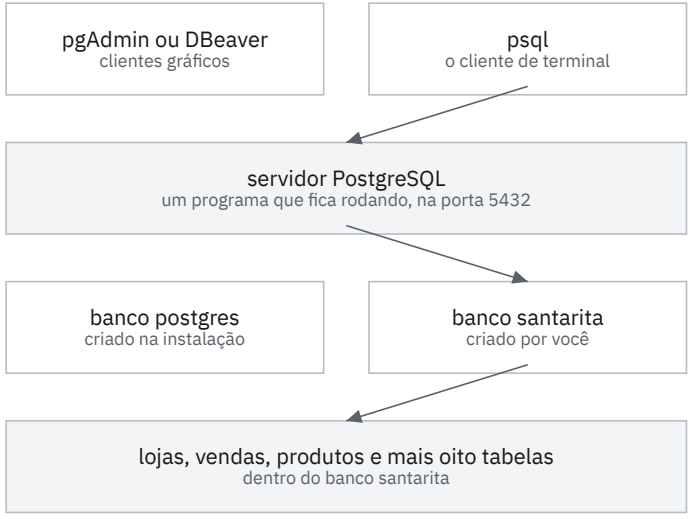


FIGURA 3.1 As peças da bancada. Os clientes mandam consultas ao servidor pela porta 5432; o servidor guarda vários bancos, e cada banco, as suas tabelas.

Um servidor é um programa que fica rodando à espera de pedidos, como um balcão de farmácia aberto. O cliente faz o pedido — uma

consulta — e recebe a resposta. Por isso o banco continua funcionando com o psql fechado: o psql é só um dos balcões por onde se chega a ele.

Há dois caminhos. O primeiro instala o PostgreSQL direto no sistema. O segundo roda o PostgreSQL dentro de um **container** do Docker, um ambiente isolado que se cria e se apaga com um comando, sem mexer no resto do computador. Escolha um.

3.2 Caminho 1: instalar no sistema

SISTEMA	COMO INSTALAR
Windows	o instalador do site oficial, em postgresql.org/download, que traz o servidor, o psql e o pgAdmin
macOS	o Postgres.app, que roda o servidor como um aplicativo, ou `brew install postgresql@18`
Linux	o pacote `postgresql` da distribuição, ou o repositório oficial do PostgreSQL para a versão mais nova

TABELA 3.1 Onde instalar o PostgreSQL em cada sistema.

Durante a instalação, o PostgreSQL cria um usuário administrador chamado `postgres` e pede uma senha para ele. Anote a senha. No Windows, o psql fica no menu Iniciar como *SQL Shell (psql)*; nos outros sistemas, roda no terminal.

3.3 Caminho 2: um container

Com o Docker instalado e aberto, um comando cria o servidor:

```
docker run --name santarita -e POSTGRES_PASSWORD=santarita \
       -p 5432:5432 -d postgres:18
```

O `--name` dá nome ao container; o `-e` define a senha do usuário `postgres`; o `-p 5432:5432` liga a porta 5432 do seu computador, a porta padrão do PostgreSQL, à do container; o `-d` deixa o servidor rodando em segundo plano. O psql já vem dentro do container:

```
docker exec -it santarita psql -U postgres
```

Para parar o servidor, `docker stop santarita`; para ligá-lo de novo, `docker start santarita`. Os dados continuam lá enquanto o container existir.

3.4 Criar o banco da Santa Rita

Um servidor guarda vários **bancos de dados**, cada um com as suas tabelas. O da Santa Rita se chama `santarita`, e é criado com `CREATE DATABASE`, dentro do psql conectado como `postgres`:

NORUN sql

```sql
CREATE DATABASE santarita;
```

```
CREATE DATABASE
```

O psql responde com o nome do comando quando ele dá certo. Para entrar no banco novo, `\c`:

NORUN sql

```
\c santarita
```

```
You are now connected to database "santarita" as user "postgres".
```

Os dois scripts do apêndice A1 — `esquema.sql`, que cria as tabelas, e `santarita.sql`, que as preenche — rodam com `\i` e o caminho do arquivo:

NORUN sql

```
\i esquema.sql
\i santarita.sql
```

O psql mostra o nome de cada comando executado: onze `CREATE TABLE` e uma sequência de `INSERT`. Para conferir, a mesma consulta da Bruna:

```sql
SELECT nome, cidade FROM lojas;
```

```
     nome     |   cidade
--------------+-------------
 Centro       | Divinópolis
 Bom Pastor   | Divinópolis
 Itaúna       | Itaúna
(3 rows)
```

Se as três lojas aparecerem, a bancada está pronta.

3.5 Os comandos do psql

Os comandos que começam com barra invertida são do psql, e não do SQL: não terminam com ponto e vírgula e não vão para o servidor. Estes são os que o livro usa:

COMANDO	O QUE FAZ
`\dt`	lista as tabelas do banco
`\d tabela`	mostra as colunas de uma tabela
`\c banco`	conecta a outro banco
`\i arquivo`	executa os comandos de um arquivo
`\x`	alterna a exibição em lista, uma coluna por linha
`\q`	sai do psql

TABELA 3.2 Os comandos do psql usados no livro.

O `\x` ajuda quando uma tabela tem colunas demais para a largura da tela: em vez de uma linha por registro, mostra um bloco por registro, com uma coluna em cada linha. Um segundo `\x` volta ao normal.

```
\x
SELECT * FROM clientes WHERE id = 1;
```

```
1  Expanded display is on.
2  -[ RECORD 1 ]+----------------------
3  id           | 1
4  nome         | Maria Aparecida Souza
5  cpf          | 34812099595
6  nascimento   | 1948-03-12
7  cidade       | Divinópolis
8  fidelidade   | ouro
9  desconto_pct | 10.0
```

O `WHERE id = 1` pede só o cliente de número 1; filtrar linhas é o assunto da parte seguinte do livro.

3.6 Um cliente gráfico

O psql é o cliente que todo PostgreSQL tem, e é com ele que o livro mostra os resultados. Para o dia a dia, muita gente prefere um cliente gráfico, com uma árvore das tabelas à esquerda e um editor de consultas com cores:

- o **pgAdmin**, feito pelo próprio projeto PostgreSQL, que vem com o instalador do Windows;

- o **DBeaver**, gratuito, que conversa com PostgreSQL, MySQL, SQLite, SQL Server, Oracle e muitos outros.

Os dois se conectam ao servidor com os mesmos dados: o endereço (`localhost`, o próprio computador), a porta (5432), o usuário (`postgres`), a senha e o banco (`santarita`). As consultas são as mesmas; muda só a moldura em volta do resultado.

O ESSENCIAL

O PostgreSQL pode ser instalado direto no sistema ou rodar num container do Docker; nos dois casos há um usuário administrador, `postgres`, e o servidor atende na porta 5432. Um servidor guarda vários bancos: `CREATE DATABASE santarita` cria o da Santa Rita, `\c` conecta a ele e `\i` executa os scripts do apêndice A1. Os comandos com barra invertida são do psql e não vão para o servidor. Clientes gráficos como o pgAdmin e o DBeaver mandam as mesmas consultas, com outra moldura.

RESUMO

- Servidor guarda e responde; cliente manda as consultas. O livro usa o PostgreSQL 18 e o psql.
- Instalar no sistema (instalador oficial, Postgres.app, pacote da distribuição) ou rodar `docker run ... postgres:18`.
- `CREATE DATABASE santarita`, `\c santarita`, `\i esquema.sql` e `\i santarita.sql` criam a base do livro.
- `\dt`, `\d`, `\c`, `\i`, `\x` e `\q` são comandos do psql, sem ponto e vírgula.
- pgAdmin e DBeaver são clientes gráficos para o mesmo servidor.

1 Com a base criada, use `\d` para descobrir quais colunas da tabela `produtos` podem ficar sem valor.

2 Ligue o `\x` e mostre todas as colunas da loja de número 3. O que o psql mostra na metragem?

PARTE 2

Perguntar a uma tabela

Selecionar, filtrar, ordenar, calcular e agrupar — e o NULL, que não é zero, nem vazio, nem falso.

NESTA PARTE

4 SELECT e FROM
5 WHERE
6 NULL, o valor que não é valor
7 Ordenar e limitar
8 Funções de linha
9 Datas e horas
10 Agrupar
11 Do resumo ao relatório

CAPÍTULO

4

SELECT e FROM

Uma coluna chamada ?column?, uma loja de 260 m² que perde dois metros quadrados numa divisão, uma ficha que some por completo por causa de um campo vazio e as quatro formas de pagamento que a farmácia aceita.

AO FIM DESTE CAPÍTULO

Calcular colunas novas com expressões e entender como o tipo decide o resultado de uma conta, distinguir textos de nomes, dar apelidos às colunas, eliminar linhas repetidas com DISTINCT e saber em que ordem o banco lê uma consulta.

Capítulo disponível na versão completa

CAPÍTULO

5
WHERE

A Dra. Lúcia quer os lotes vencidos da loja do Centro, e recebe um da loja do Bom Pastor de brinde; o relatório da primeira semana de dezembro perde o dia 7 todo; e um cliente de Itaúna some porque a cidade foi digitada com letra minúscula.

AO FIM DESTE CAPÍTULO
Escolher linhas com WHERE: comparações de números, textos e datas, condições compostas com AND, OR e NOT e a precedência entre elas, períodos com data e hora, listas com IN e padrões de texto com LIKE.

Capítulo disponível na versão completa

CAPÍTULO

6

NULL, o valor que não é valor

Uma campanha para todos os clientes que não são do programa ouro que não chega a três deles, uma consulta que procura o vazio com um sinal de igual e não acha nada, e a lógica de três valores que explica as duas coisas.

AO FIM DESTE CAPÍTULO
Entender o que é NULL, por que uma comparação com NULL não é nem verdadeira nem falsa, procurar ausências com IS NULL, trocar NULL por um valor com COALESCE e não perder linhas por causa de uma condição com <> ou NOT.

Capítulo disponível na versão completa

CAPÍTULO

7

Ordenar e limitar

O ranking das maiores lojas da rede começa pela única que não tem metragem, o top 3 de produtos sai diferente a cada execução, e a segunda página do relatório repete um cliente da primeira.

AO FIM DESTE CAPÍTULO

Ordenar resultados com ORDER BY, por uma ou várias colunas e por expressões, saber onde o NULL vai parar, limitar o número de linhas com LIMIT e FETCH FIRST, lidar com empates e paginar sem perder nem repetir linhas.

Capítulo disponível na versão completa

CAPÍTULO

8

Funções de linha

O jurídico quer o CPF escondido nos relatórios, a política de preços quer tudo terminado em noventa centavos e baixa o preço de um serviço sem querer, e um CPF guardado como número perde o zero da frente.

AO FIM DESTE CAPÍTULO
Usar funções que trabalham linha a linha — de texto e de número —, classificar linhas com CASE, converter valores de um tipo para outro e entender o que o banco faz quando um valor não cabe no tipo pedido.

Capítulo disponível na versão completa

CAPÍTULO

9

Datas e horas

Quantos dias faltam para cada lote vencer, uma receita que vale trinta dias e não um mês, um relatório em ordem de data que põe 2028 antes de 2025, e um painel que mostra vendas às onze da noite numa loja que fecha às nove.

AO FIM DESTE CAPÍTULO
Conhecer os tipos de data e hora, fazer contas com datas e durações, tirar pedaços de uma data e arredondá-la para o começo da semana ou do mês, formatá-la para leitura e entender o que o fuso horário muda — e o que não muda — num instante guardado.

Capítulo disponível na versão completa

CAPÍTULO

10

Agrupar

O desconto médio dos clientes é 5,2% ou 3,25%, dependendo de quem conta como cliente; o faturamento de dezembro cabe numa linha; e o PostgreSQL se recusa a mostrar uma forma de pagamento que ele não saberia qual escolher.

AO FIM DESTE CAPÍTULO

Resumir muitas linhas em poucas com as funções de agregação — COUNT, SUM, AVG, MIN e MAX —, entender como elas tratam o NULL e o conjunto vazio, agrupar com GROUP BY, filtrar grupos com HAVING e acompanhar uma consulta completa na ordem em que o banco a lê.

Capítulo disponível na versão completa

CAPÍTULO

11

Do resumo ao relatório

Vendas por semana numa semana que começou em novembro, um quadro com as lojas nas linhas e as formas de pagamento nas colunas, um zero que não existia em lugar nenhum e uma linha de total que o GROUP BY sozinho não dá.

AO FIM DESTE CAPÍTULO
Agrupar por uma expressão, como a semana de cada venda, contar e somar com condição para montar um quadro com FILTER e CASE, acrescentar subtotais e total com ROLLUP e juntar os textos de um grupo com string_agg.

Capítulo disponível na versão completa

PARTE 3

Juntar tabelas

Chaves, junções que acham, junções que perdem, junções que multiplicam, subconsultas e funções de janela.

NESTA PARTE

12 Chaves
13 JOIN
14 As linhas que faltam
15 O faturamento que dobrou
16 Subconsultas
17 Conjuntos
18 WITH
19 Funções de janela

CAPÍTULO

12
Chaves

Uma loja nova que o banco recusa porque o número já tem dono, uma venda numa loja que não existe, um CPF que não serve para identificar ninguém e a primeira vez que a Bruna segue um fio de uma tabela até outra — à mão.

AO FIM DESTE CAPÍTULO
Entender o que identifica uma linha — a chave primária, simples ou composta —, por que a Santa Rita usa números e não o CPF, como uma chave estrangeira liga duas tabelas e impede referências a linhas que não existem, e seguir uma ligação de uma tabela a outra.

Capítulo disponível na versão completa

CAPÍTULO

13

JOIN

Quinze vendas que ganham o nome da loja, uma coluna chamada id que o banco não sabe de qual tabela é, o faturamento em reais que a Cláudia pediu numa consulta só, e quatro cupons que somem quando entram os clientes.

AO FIM DESTE CAPÍTULO
Juntar tabelas com JOIN e ON, entender a junção como a escolha dos pares de linhas que satisfazem uma condição, qualificar nomes e usar apelidos de tabela, encadear três tabelas e montar o quadro de faturamento da Santa Rita.

Capítulo disponível na versão completa

CAPÍTULO

14

As linhas que faltam

Quatro cupons que voltam com o cliente em branco, uma loja que vendeu zero em débito e aparece com um, a mesma condição que dá três linhas no ON e duas no WHERE, um pacote de fraldas que nunca foi vendido e três caixas de dipirona que não deviam ter saído.

AO FIM DESTE CAPÍTULO
Manter as linhas sem par com LEFT, RIGHT e FULL JOIN, contar corretamente o que veio do outro lado, entender por que uma condição muda de sentido no ON e no WHERE e achar o que não existe com a anti-junção.

Capítulo disponível na versão completa

CAPÍTULO

15

O faturamento que dobrou

Um relatório de 2019 que vende o dobro de losartana que a farmácia vendeu, um serviço que some do faturamento por categoria, uma tabela juntada com ela mesma para achar o chefe e trinta combinações de loja e produto, das quais nove nunca aconteceram.

AO FIM DESTE CAPÍTULO
Prever quantas linhas uma junção devolve pela cardinalidade das ligações, reconhecer a junção que multiplica linhas e infla somas, conferir uma junção antes de confiar nela, juntar uma tabela com ela mesma e usar o produto cartesiano de propósito.

ENQUANTO ISSO, NA EMPRESA **A consulta de 2019**

Na segunda, 25 de janeiro, o Rafael achou num servidor antigo a consulta que gerava a planilha da segunda-feira. Mandou para a Bruna com um comentário: “foi o estagiário de 2019, mas funcionava”.

Era um `SELECT` de quarenta linhas, com vírgulas entre seis tabelas. A Bruna rodou só o pedaço do faturamento por categoria no recorte de dezembro e comparou com o dela.

— Anti-hipertensivos, 296,64. O meu dá 148,32. — Ela fez a conta duas vezes. — É exatamente o dobro.



FIGURA 15.1 Uma junção no lugar errado vende o dobro do que a farmácia vendeu.

15.1 Para quantos

Antes de somar qualquer coisa depois de uma junção, vale responder a uma pergunta: **para cada linha de uma tabela, quantas linhas da outra combinam?** É a **cardinalidade** da ligação. Entre vendas e lojas, é “muitas para uma”: uma venda tem uma loja, uma loja tem muitas vendas. Entre produtos e lotes, é “uma para muitas”, e “muitas” quer dizer quantas a base tiver:

```sql
SELECT p.id, count(lt.id) AS lotes
FROM produtos p
LEFT JOIN lotes lt ON lt.produto_id = p.id
GROUP BY p.id
ORDER BY p.id;
```

```
 id | lotes
----+-------
  1 |     2
  2 |     2
  3 |     2
  4 |     1
  5 |     1
  6 |     1
  7 |     1
  8 |     1
  9 |     1
 10 |     0
(10 rows)
```

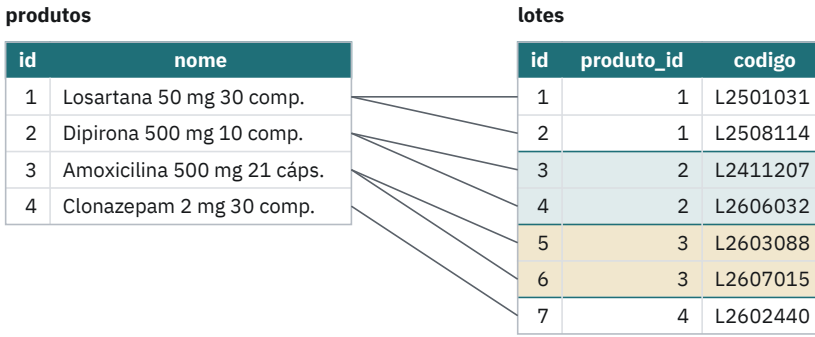


FIGURA 15.2 Os quatro primeiros produtos e os seus lotes. A losartana, a dipirona e a amoxicilina têm dois lotes cada; o clonazepam, um. Quem segue o fio de um produto para os lotes pode encontrar mais de uma linha.

A regra das junções da Santa Rita sai daí. Seguindo uma chave estrangeira **na direção dela** — do item para o produto, da venda para a loja —, cada linha encontra exatamente uma, e o número de linhas não muda. Seguindo **contra** ela — do produto para os lotes, da loja para as vendas —, cada linha encontra quantas houver: nenhuma, uma ou várias.

15.2 A junção que multiplica

A consulta de 2019 juntava os itens vendidos aos lotes pelo **produto**: para saber a categoria, ela passava por `lotes`, que também tem o `produto_id`. Os cinco itens de losartana de dezembro:

```
SELECT i.venda_id, i.item, lt.codigo,
       i.quantidade * i.preco_unit - i.desconto AS valor
FROM itens_venda i
JOIN lotes lt ON lt.produto_id = i.produto_id
WHERE i.produto_id = 1
ORDER BY i.venda_id, lt.codigo;
```

```
 venda_id | item |  codigo   | valor
----------+------+-----------+-------
        1 |    1 | L2501031 | 40.09
        1 |    1 | L2508114 | 40.09
        3 |    2 | L2501031 | 22.27
        3 |    2 | L2508114 | 22.27
        7 |    1 | L2501031 | 21.60
        7 |    1 | L2508114 | 21.60
       13 |    2 | L2501031 | 21.16
       13 |    2 | L2508114 | 21.16
       14 |    1 | L2501031 | 43.20
       14 |    1 | L2508114 | 43.20
(10 rows)
```

Dez linhas para cinco itens. O `ON` só exige que o lote seja do mesmo produto, e a losartana tem dois lotes: cada item forma par com os dois. Pela definição da junção, os dois pares satisfazem a condição, e os dois ficam.

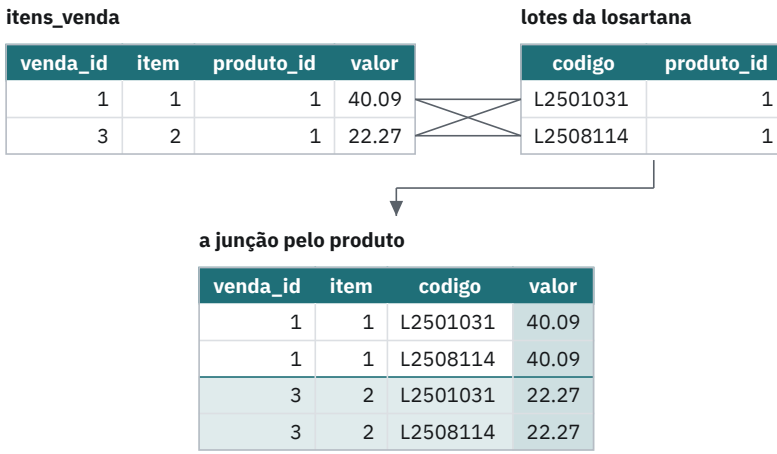


FIGURA 15.3 Dois itens de losartana e os dois lotes de losartana. Pela condição lt.produto*id* = *i.produto*id, os quatro pares combinam: cada item aparece duas vezes, com o mesmo valor.

Uma soma sobre esse resultado soma cada item duas vezes. O relatório por categoria de 2019, reescrito com `JOIN`:

```
SELECT c.nome AS categoria,
       sum(i.quantidade * i.preco_unit - i.desconto)
           AS faturamento
FROM itens_venda i
JOIN produtos p ON p.id = i.produto_id
JOIN categorias c ON c.id = p.categoria_id
JOIN lotes lt ON lt.produto_id = p.id
GROUP BY c.nome
ORDER BY faturamento DESC;
```

```
        categoria       | faturamento
------------------------+-------------
 Fraldas                |      412.73
 Dermocosméticos        |      378.48
 Anti-hipertensivos     |      296.64
 Genéricos              |      145.82
 Isentos de prescrição  |      141.97
 Controlados            |       36.85
(6 rows)
```

Três erros num resultado que parece normal. Anti-hipertensivos — só a losartana — e genéricos — só a amoxicilina — dobraram, porque os dois produtos têm dois lotes. Isentos de prescrição cresceram mais da metade: a dipirona, com dois lotes, dobrou, e a loratadina, com um, não. E a categoria Serviços sumiu: a aferição de pressão não tem lote, e a junção interna com `lotes` a descartou. Nenhuma mensagem de erro, e o total do relatório, 1.412,49, passa por um mês bom.

> ↯ **ERRO COMUM**
> Uma junção que multiplica não dá erro e não dá aviso. Ela dá um número maior, e um número maior num relatório de vendas raramente é questionado. A única defesa é conferir: contar as linhas antes e depois de cada junção, e comparar o total com um total conhecido.

15.3 Conferir antes de confiar

A conferência tem dois números. O primeiro é a contagem de linhas: se a junção não deveria multiplicar nada, o resultado tem o mesmo número de linhas da tabela de partida. O segundo é o total: o faturamento de dezembro é 1.156,04 em qualquer agrupamento.

```
SELECT count(*) AS linhas,
       count(DISTINCT (i.venda_id, i.item)) AS itens
FROM itens_venda i
JOIN lotes lt ON lt.produto_id = i.produto_id;
```

```
 linhas | itens
--------+-------
     38 |    25
(1 row)
```

`(i.venda_id, i.item)` é a chave primária dos itens escrita como um par, e `count(DISTINCT ...)` conta os pares diferentes. Trinta e oito linhas para 25 itens: há itens repetidos. E 25 itens, e não 28: três sumiram. As duas contagens contam a história completa da consulta errada.

O conserto não é um `DISTINCT` no fim — ele esconderia as duplicatas de uma soma, mas também juntaria dois itens diferentes de mesmo valor. O conserto é a junção certa. Para a categoria, a consulta não precisa dos lotes: o caminho é item → produto → categoria, sempre na direção das chaves.

```
SELECT c.nome AS categoria,
       sum(i.quantidade * i.preco_unit - i.desconto)
           AS faturamento
FROM itens_venda i
JOIN produtos p ON p.id = i.produto_id
JOIN categorias c ON c.id = p.categoria_id
GROUP BY c.nome
ORDER BY faturamento DESC;
```

```
        categoria       | faturamento
------------------------+-------------
 Fraldas                |      412.73
 Dermocosméticos        |      378.48
 Anti-hipertensivos     |      148.32
 Isentos de prescrição  |       92.00
 Genéricos              |       72.91
 Controlados            |       36.85
 Serviços               |       14.75
(7 rows)
```

Sete categorias, e a soma é 1.156,04. Quando o lote importa — para o custo, por exemplo, que é do lote e não do produto —, a junção certa é pelo lote do item, `lt.id = i.lote_id`: cada item foi vendido de um lote só, e o fio vai na direção da chave.

15.4 Uma tabela com ela mesma

A autorreferência de `funcionarios` pede uma junção da tabela com ela mesma: a mesma tabela aparece duas vezes no `FROM`, com dois apelidos, e cada apelido faz um papel. `f` é o funcionário; `c`, o chefe.

```
SELECT f.nome, c.nome AS chefe
FROM funcionarios f
JOIN funcionarios c ON f.chefe_id = c.id
ORDER BY f.id;
```

```
        nome       |      chefe
-------------------+-----------------
 Osvaldo Ribeiro | Cláudia Menezes
 Lúcia Amaral    | Cláudia Menezes
 Bruna Teixeira  | Osvaldo Ribeiro
 Zilda Moreira   | Cláudia Menezes
 Tânia Freitas   | Cláudia Menezes
 Marcos Rocha    | Cláudia Menezes
 Ana Mendes      | Zilda Moreira
 Bruno Dias      | Tânia Freitas
 Letícia Nunes   | Marcos Rocha
(9 rows)
```

Nove linhas para dez funcionários: a Cláudia, sem chefe, não forma par. Os apelidos aqui não são uma comodidade: sem eles, `funcionarios.nome` não diria de qual das duas cópias é o nome. A **autojunção** obedece a tudo o que vale para as outras — com `LEFT JOIN`, a Cláudia voltaria, com o chefe em branco.

15.5 O produto cartesiano

`CROSS JOIN` junta sem condição nenhuma: todos os pares, sem `ON`. Com as três lojas e dois produtos:

```
SELECT l.nome AS loja, p.nome AS produto
FROM lojas l
CROSS JOIN produtos p
WHERE p.id IN (2, 6)
ORDER BY l.id, p.id;
```

```
     loja     |          produto
--------------+--------------------------
 Centro       | Dipirona 500 mg 10 comp.
 Centro       | Sérum vitamina C 30 ml
 Bom Pastor | Dipirona 500 mg 10 comp.
 Bom Pastor | Sérum vitamina C 30 ml
 Itaúna       | Dipirona 500 mg 10 comp.
 Itaúna       | Sérum vitamina C 30 ml
(6 rows)
```

Três vezes dois, seis. É o **produto cartesiano**, o conjunto de que toda junção parte antes de o `ON` escolher. Sem o `WHERE`, seriam três vezes dez, trinta pares.

Na maior parte das vezes, ele aparece por engano. Na forma antiga, com vírgula, esquecer a condição no `WHERE` não dá erro, dá o produto:

```
SELECT count(*) FROM vendas, lojas;
```

```
 count
-------
    45
(1 row)
```

Quinze vendas vezes três lojas. Uma soma sobre isso triplicaria o faturamento, e numa consulta de seis tabelas, como a de 2019, um fio esquecido multiplica tudo pelo tamanho da tabela que ficou solta.

De propósito, o produto cartesiano serve para montar uma **grade**: todas as combinações que poderiam existir, para depois ver quais existiram. As combinações de loja e produto ativo que não venderam nada em dezembro:

```
SELECT l.nome AS loja, p.nome AS produto
FROM lojas l
CROSS JOIN produtos p
LEFT JOIN (itens_venda i JOIN vendas v ON v.id = i.venda_id)
       ON v.loja_id = l.id AND i.produto_id = p.id
WHERE p.ativo
GROUP BY l.nome, p.nome
HAVING count(i.item) = 0
ORDER BY l.nome, p.nome;
```

```
    loja    |           produto
------------+-----------------------------
 Bom Pastor | Aferição de pressão arterial
 Bom Pastor | Loratadina 10 mg 12 comp.
 Bom Pastor | Losartana 50 mg 30 comp.
 Bom Pastor | Protetor solar FPS 50 120 ml
 Centro     | Sérum vitamina C 30 ml
 Itaúna     | Clonazepam 2 mg 30 comp.
 Itaúna     | Dipirona 500 mg 10 comp.
 Itaúna     | Fralda infantil M pacote 60
 Itaúna     | Sérum vitamina C 30 ml
(9 rows)
```

A grade tem 27 combinações de loja e produto ativo. Os parênteses juntam itens e vendas primeiro, e o resultado dessa junção entra no `LEFT JOIN` como se fosse uma tabela só, com o `ON` falando das duas. As combinações sem nenhum item são as que a grade tem e as vendas

não têm: nove, e o Bom Pastor sem uma caixa de losartana vendida em dezembro é uma pergunta para a gerente.

> **O ESSENCIAL**
> Antes de juntar, pergunte a cardinalidade: seguindo uma chave estrangeira na direção dela, cada linha encontra uma; contra ela, quantas houver. Uma junção por uma coluna que não identifica a linha do outro lado multiplica as linhas, e toda soma depois dela infla, sem erro. Confira contando as linhas e as chaves (`count(DISTINCT (a, b))`) e comparando com um total conhecido; conserte a junção, e não o resultado com `DISTINCT`. A autojunção usa a mesma tabela com dois apelidos. `CROSS JOIN` dá todos os pares: por engano, é o fio esquecido; de propósito, é uma grade de combinações.

ENQUANTO ISSO, NA EMPRESA **Vinte e quatro**

O Rafael rodou na base toda, a de produção, as duas versões da margem de dezembro: a de 2019, com a junção pelos lotes do produto, e a da Bruna, com o lote de cada item.

A de 2019 deu 31%, o número do slide da Cláudia. A da Bruna deu 24%.

A Cláudia leu os dois números três vezes e marcou uma reunião para as cinco da tarde, com a porta fechada.

RESUMO

- Cardinalidade: na direção da chave estrangeira, uma linha; contra ela, quantas houver.
- Juntar por uma coluna que não identifica a linha do outro lado multiplica linhas e infla somas, sem erro.
- Confira: linhas antes e depois, `count(DISTINCT (chave))`, total conhecido. Conserte a junção, não o resultado.
- Autojunção: a mesma tabela duas vezes, com apelidos diferentes.
- `CROSS JOIN` dá todos os pares; a vírgula sem condição faz o mesmo por engano; de propósito, monta uma grade.

1 Mostre os funcionários do escritório — os que não têm loja — com o nome do chefe, sem perder a Cláudia. No lugar do chefe dela, escreva `(ninguém)`.

2 Sem rodar, diga quantas linhas devolve `SELECT * FROM itens_venda i JOIN lotes l` usando só a contagem de itens por produto e de lotes por produto.

3 ... Quais funcionários ganham mais de 60% do salário do próprio chefe? Mostre o nome, o salário, o chefe e o salário do chefe.

CAPÍTULO

16

Subconsultas

Uma consulta dentro da outra para achar os produtos acima da média, a fatia de cada loja no faturamento, a última compra de cada cliente e um relatório de lotes parados que responde nenhum — porque, no meio de uma lista de doze números, havia um NULL.

AO FIM DESTE CAPÍTULO
Usar uma consulta dentro de outra — como valor, como lista, como teste de existência e como tabela —, distinguir a subconsulta que roda uma vez da que roda para cada linha e entender por que NOT IN com um NULL na lista nunca é verdadeiro.

Capítulo disponível na versão completa

CAPÍTULO

17

Conjuntos

As cidades em que a Santa Rita está presente, contadas uma vez e contadas quinze; os produtos que o Centro e o Bom Pastor vendem em comum, e os que só o Centro vende; e um ORDER BY que o banco recusa porque não sabe a que consulta ele pertence.

AO FIM DESTE CAPÍTULO

Empilhar os resultados de duas consultas com UNION e UNION ALL, achar o que é comum com INTERSECT e o que sobra com EXCEPT, conhecer as regras que as consultas empilhadas precisam seguir e montar um relatório em pedaços.

Capítulo disponível na versão completa

CAPÍTULO

18
WITH

Uma consulta de quarenta linhas que vira três consultas com nome, os cupons acima do ticket médio calculados sem repetir a conta, uma árvore de categorias percorrida até as folhas e o organograma da Santa Rita montado por uma consulta que chama a si mesma.

AO FIM DESTE CAPÍTULO
Dar nome a subconsultas com WITH para escrever consultas longas em passos legíveis, reaproveitar um resultado intermediário, percorrer hierarquias com WITH RECURSIVE e saber como uma recursão começa, avança e para.

Capítulo disponível na versão completa

CAPÍTULO

19

Funções de janela

Cada cupom ao lado do total da sua loja sem perder nenhuma linha, um empate de 90,72 que a numeração resolve no cara ou coroa, o maior cupom de cada loja, os dias entre uma compra e outra e a curva do faturamento de dezembro somada dia a dia.

AO FIM DESTE CAPÍTULO
Calcular agregações e numerações sobre um conjunto de linhas sem juntar as linhas num grupo — OVER, PARTITION BY, ORDER BY da janela —, classificar com ROW*NUMBER, RANK e DENSE*RANK, olhar a linha vizinha com LAG e LEAD, acumular com molduras e saber em que momento da consulta a janela é calculada.

Capítulo disponível na versão completa

PARTE 4

Modelar

Do balcão da farmácia ao diagrama, dos tipos às restrições, e o esquema que muda sem fechar a loja.

NESTA PARTE

20 Do balcão ao diagrama
21 Tipos de dados
22 Normalização
23 Restrições
24 O esquema muda

CAPÍTULO

20

Do balcão ao diagrama

Uma tabela de 214 colunas que é, na verdade, seis tabelas; uma loja que fica em duas cidades; um laboratório escrito de dois jeitos; e o desenho, feito com consultas, do banco que a Rede Bem-Estar nunca teve.

AO FIM DESTE CAPÍTULO

Ler uma tabela achatada e achar nela as entidades, os atributos e os identificadores; descobrir com consultas a cardinalidade de cada ligação — um para um, um para muitos, muitos para muitos — e desenhar o modelo num diagrama de entidades e relacionamentos.

Capítulo disponível na versão completa

CAPÍTULO

21

Tipos de dados

Dez moedas de dez centavos que somam 0,9999999999999999, um contador que estoura no número 2.147.483.648, uma cidade que perde o fim do nome sem ninguém avisar e o 1º de dezembro que vira 12 de janeiro porque o servidor lê datas como um americano.

AO FIM DESTE CAPÍTULO

Escolher o tipo de cada coluna sabendo o que ele aceita, o que ele recusa e o que ele faz em silêncio: numeric e float para valores com casas, os inteiros e os seus limites, text, varchar e char, datas que chegam como texto, boolean e colunas de identidade.

Capítulo disponível na versão completa

CAPÍTULO

22

Normalização

Uma loja que muda de cidade em três linhas e esquece uma, um laboratório que desaparece do cadastro quando dois cupons são cancelados, uma loja nova que não pode ser cadastrada antes de vender alguma coisa — e as seis tabelas que tornam os três erros impossíveis.

AO FIM DESTE CAPÍTULO

Reconhecer as anomalias de uma tabela redundante, entender as dependências entre colunas e as três primeiras formas normais, decompor a tabela da Bem-Estar em tabelas normalizadas, carregá-las e conferir que nada se perdeu — e saber quando vale a pena desfazer a normalização.

Capítulo disponível na versão completa

CAPÍTULO

23

Restrições

Menos duas caixas de dipirona vendidas e aceitas, uma regra que não pode nascer enquanto houver uma linha que a desobedece, um laboratório cadastrado duas vezes por causa das maiúsculas e um cupom cancelado que leva os itens junto.

AO FIM DESTE CAPÍTULO

Pôr as regras do negócio dentro do banco com CHECK, UNIQUE, NOT NULL e DEFAULT, dar nomes às regras para que as recusas digam o que aconteceu, acrescentar uma regra a uma tabela que já tem dados e escolher o que uma chave estrangeira faz quando a linha apontada é apagada.

Capítulo disponível na versão completa

CAPÍTULO

24

O esquema muda

Uma coluna nova que não pode ser obrigatória numa tabela que já tem linhas, um nome trocado que derruba o painel do Diego num sábado, uma tabela que o banco se recusa a apagar e outra que volta depois de apagada — e o caderno de mudanças que impede a mesma alteração de ser feita duas vezes.

AO FIM DESTE CAPÍTULO
Mudar uma tabela que já tem dados com ALTER TABLE — acrescentar, renomear, mudar o tipo e tirar colunas —, apagar tabelas com DROP sabendo o que depende delas, desfazer uma mudança de estrutura com ROLLBACK e registrar as mudanças como migrações numeradas.

Capítulo disponível na versão completa

PARTE 5

Mudar os dados

Inserir, alterar, apagar, e fazer tudo isso sem que a madrugada das três horas trave.

NESTA PARTE

25 INSERT, UPDATE, DELETE
26 Transações
27 A madrugada das 3h
28 Cargas

CAPÍTULO

25

INSERT, UPDATE, DELETE

Um produto novo que diz o próprio número de volta, um reajuste de 4,5% que mostra o preço de antes e o de depois, o sábado de 2019 em que 3.104 produtos passaram a custar zero real e a baixa dos lotes vencidos, apagada com uma junção.

AO FIM DESTE CAPÍTULO
Escrever dados com INSERT, UPDATE e DELETE, usar RETURNING para ver o que cada comando fez, atualizar e apagar a partir de outra tabela, e criar os hábitos que impedem o UPDATE sem WHERE.

ENQUANTO ISSO, NA EMPRESA **O sábado de 2019**

Na segunda, 8 de fevereiro, a Cláudia mandou a planilha do reajuste anual: 4,5% nos genéricos e nos anti-hipertensivos, a partir de abril.

— Isso é um `UPDATE` — disse a Bruna.

O Seu Osvaldo abriu uma pasta no computador, chamada `NUNCA MAIS`, e mostrou uma foto de tela, tirada com o celular. Uma consulta de uma linha e uma resposta: `UPDATE 3104`.

— Sábado, 9 de março de 2019, oito da manhã. Era para reajustar um produto. Das oito às onze, as 48 lojas venderam 3.104 produtos a zero real.

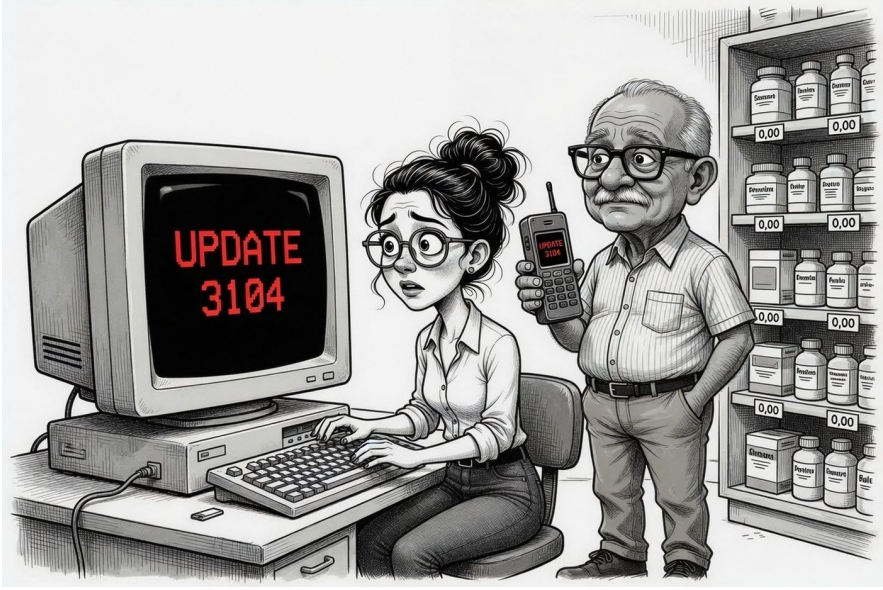


FIGURA 25.1 Um UPDATE sem WHERE não avisa: ele só responde quantas linhas.

> ◆ **DICA**
> Daqui em diante, os capítulos mudam os dados de verdade — preços, estoque, vendas —, e cada um precisa começar da base como ela é. O jeito é trabalhar numa **cópia**: `createdb` com `-T` cria um banco novo igual a outro, e o banco `santarita` fica intacto para o capítulo seguinte. Com o psql fechado — a cópia não pode ser feita com alguém conectado ao original —, no terminal:
>
> ```
> createdb -T santarita capitulo25
> psql -d capitulo25
> ```
>
> Cada capítulo das partes 5 e 6 começa assim, com o seu número.

25.1 INSERT

`INSERT INTO tabela (colunas) VALUES (valores)` acrescenta linhas, e já apareceu várias vezes. Ele tem uma cláusula que diz de volta o que entrou, `RETURNING`. O omeprazol, que entra no catálogo da Santa Rita em março:

```
INSERT INTO produtos (id, nome, categoria_id, fornecedor_id,
                      preco, custo, controlado, tarja, ativo)
VALUES (11, 'Omeprazol 20 mg 28 cáps.', 2, 1, 19.90, 8.40,
        false, NULL, true)
RETURNING id, nome, preco;
```

```
 id |           nome           | preco
----+--------------------------+-------
 11 | Omeprazol 20 mg 28 cáps. | 19.90
(1 row)

INSERT 0 1
```

O `RETURNING` devolve uma tabela com as colunas pedidas das linhas que entraram, como um `SELECT` feito na mesma hora. Numa tabela com identidade, é o jeito de saber que número o banco deu à linha nova. A lista de colunas depois do nome da tabela é opcional e vale a pena escrevê-la sempre: sem ela, o `INSERT` depende da ordem das colunas na tabela, e quebra no dia em que alguém acrescentar uma.

Uma coluna obrigatória deixada de fora, sem `DEFAULT`, recusa a linha:

QUEBRADO sql

```
INSERT INTO produtos (id, nome, preco, custo, controlado,
                      ativo)
VALUES (12, 'Teste', 1, 1, false, true);
```

```
ERROR:  null value in column "categoria_id" of relation
"produtos" violates not-null constraint
DETAIL:  Failing row contains (12, Teste, null, null, 1.00,
1.00, f, null, t).
```

25.2 UPDATE

`UPDATE tabela SET coluna = valor WHERE condição` muda as linhas que a condição escolhe. O valor novo pode ser uma expressão sobre o valor antigo, e o `RETURNING` pode mostrar os dois: no PostgreSQL 18, `old` é a linha como era, e `new`, como ficou. O reajuste da Cláudia, nas categorias 2 e 3:

```
UPDATE produtos
SET preco = round(preco * 1.045, 2)
WHERE categoria_id IN (2, 3)
RETURNING id, old.preco AS antes, new.preco AS depois;
```

```
 id | antes | depois
----+-------+--------
  1 | 22.27 |  23.27
  3 | 24.97 |  26.09
 11 | 19.90 |  20.80
(3 rows)

UPDATE 3
```

A losartana, a amoxicilina e o omeprazol, que acabou de entrar. A resposta do psql, `UPDATE 3`, conta as linhas mudadas, e é o número mais importante de qualquer `UPDATE`: ele diz se o comando fez o que se pensava.

UPDATE produtos SET preco = round(preco * 1.045, 2) WHERE categoria_id IN (2, 3)

id	categoria_id	preco antes	preco depois
1	3	22.27	**23.27**
2	5	6.49	6.49
3	2	24.97	**26.09**
4	4	18.90	18.90
5	7	68.57	68.57
6	7	129.90	129.90
7	9	84.23	84.23
8	9	72.40	72.40
9	5	14.80	14.80
10	10	5.00	5.00
11	2	19.90	**20.80**

FIGURA 25.2 O UPDATE calcula a condição em cada linha, como um SELECT, e só muda as que passam. As outras oito ficam como estavam.

Um `SET` pode mudar várias colunas de uma vez, separadas por vírgula, e o `RETURNING *` mostra a linha toda:

```
UPDATE lojas SET metragem = 210, nome = 'Itaúna Centro'
WHERE id = 3
RETURNING *;
```

```
 id |      nome       | cidade | uf | inaugurada | metragem
----+-----------------+--------+----+------------+----------
  3 | Itaúna Centro | Itaúna | MG | 2012-08-15 |      210
(1 row)

UPDATE 1
```

E um `UPDATE` que não acha nenhuma linha não é erro:

```
UPDATE produtos SET ativo = false WHERE id = 99;
```

```
UPDATE 0
```

Zero linhas mudadas, e nenhuma mensagem de alerta. Um `UPDATE 0` onde se esperava um `UPDATE 1` é um erro de quem escreveu, e só o número avisa.

25.3 O UPDATE sem WHERE

O comando do sábado de 2019 era um `UPDATE produtos SET preco = ...` com o `WHERE` na linha seguinte. Quem rodou selecionou só a primeira

linha no editor e apertou o botão de executar. Sem `WHERE`, um `UPDATE` muda **todas** as linhas da tabela. Para ver sem estragar nada, dentro de uma transação, que o `ROLLBACK` desfaz:

```
BEGIN;
UPDATE produtos SET preco = 0;
SELECT count(*) AS sem_preco FROM produtos WHERE preco = 0;
ROLLBACK;
SELECT count(*) AS sem_preco FROM produtos WHERE preco = 0;
```

```
BEGIN
UPDATE 11
 sem_preco
-----------
          11
(1 row)

ROLLBACK
 sem_preco
-----------
           0
(1 row)
```

`UPDATE 11`: os onze produtos do catálogo, a zero real. Depois do `ROLLBACK`, nenhum. Na Santa Rita de 2019, o comando rodou fora de transação, e cada comando fora de uma transação se confirma sozinho, na hora: não havia `ROLLBACK` a dar.

SET preco = 0 WHERE id = 7

id	preco
1	23.27
2	6.49
3	26.09
…	…
7	0.00
…	…
11	20.80

SET preco = 0

id	preco
1	0.00
2	0.00
3	0.00
…	…
7	0.00
…	…
11	0.00

FIGURA 25.3 O mesmo UPDATE, com e sem a segunda linha. Sem o WHERE, a condição não existe, e toda linha passa.

Os hábitos que a Santa Rita adotou depois daquele sábado, e que valem para qualquer `UPDATE` ou `DELETE`:

HÁBITO	POR QUÊ
escrever o `WHERE` antes do `SET`	uma linha selecionada pela metade fica sem condição, e o comando falha em vez de mudar tudo
rodar antes um `SELECT count(*)` com o mesmo `WHERE`	o número que o `UPDATE` deve dar fica conhecido antes
fazer dentro de `BEGIN`	o `UPDATE 3104` aparece antes do `COMMIT`, e ainda dá para desfazer
conferir com `RETURNING`	as linhas mudadas aparecem na tela, com o antes e o depois

TABELA 25.1 Os hábitos contra o UPDATE sem WHERE.

> ◆ **DICA**
> O psql tem uma variável que transforma o terceiro hábito em regra: `\set AUTOCOMMIT off`. Com ela, todo comando abre uma transação sozinho, e nada se confirma até um `COMMIT` explícito. Vale para a sessão, e pode ir no arquivo de configuração do psql de quem mexe em produção.

25.4 Atualizar a partir de outra tabela

O custo do sérum de vitamina C nunca foi cadastrado, mas o lote dele tem o custo de compra. `UPDATE ... FROM` junta a tabela atualizada com outra, e a condição da junção vai no `WHERE`:

```
UPDATE produtos p
SET custo = lt.custo_unit
FROM lotes lt
WHERE lt.produto_id = p.id AND p.custo IS NULL
RETURNING p.id, p.nome, p.custo;
```

```
 id |          nome          | custo
----+------------------------+-------
  6 | Sérum vitamina C 30 ml | 77.94
(1 row)

UPDATE 1
```

Só o sérum: a aferição, sem custo, não tem lote, e a junção não acha par para ela.

> ↯ **ERRO COMUM**
> Se a junção do `UPDATE ... FROM` achar **duas** linhas para o mesmo produto — dois lotes com custos diferentes —, o PostgreSQL atualiza o produto com uma delas, qualquer uma, sem erro e sem aviso. É a cardinalidade de novo, agora escrevendo. Antes de um `UPDATE ... FROM`, confira com um `SELECT` que a junção dá uma linha por linha a atualizar.

25.5 DELETE

`DELETE FROM tabela WHERE condição` apaga as linhas que a condição escolhe, e aceita `RETURNING` como os outros dois:

```
DELETE FROM produtos WHERE id = 11 RETURNING id, nome;
```

```
 id |           nome
----+--------------------------
 11 | Omeprazol 20 mg 28 cáps.
(1 row)

DELETE 1
```

O omeprazol era um teste, e sai. A fralda geriátrica, inativa, não: ela tem lotes, e a chave estrangeira recusa.

QUEBRADO sql

```
DELETE FROM produtos WHERE id = 8;
```

```
ERROR:  update or delete on table "produtos" violates foreign
key constraint "lotes_produto_id_fkey" on table "lotes"
DETAIL:  Key (id)=(8) is still referenced from table "lotes".
```

Por isso um produto que sai de linha quase nunca é apagado: ele fica `ativo = false`, com o histórico de vendas completo apontando para ele.

`DELETE ... USING` apaga a partir de uma junção, como o `UPDATE ... FROM`. A Dra. Lúcia autorizou a baixa do estoque dos lotes vencidos — as caixas vão para a incineração, e saem do estoque:

```
DELETE FROM estoque e
USING lotes lt
WHERE lt.id = e.lote_id
  AND lt.validade < '2027-01-04'
RETURNING e.loja_id, e.lote_id, e.quantidade;
```

```
 loja_id | lote_id | quantidade
---------+---------+------------
       1 |       3 |          6
       2 |      11 |          3
(2 rows)

DELETE 2
```

As seis caixas de dipirona do Centro e as três fraldas do Bom Pastor, os dois lotes que a lista da vigilância tinha. O `RETURNING` aqui não é só conferência: é o registro do que foi incinerado, que a Dra. Lúcia precisa guardar.

> ⋆ **VOCÊ SABIA**
> `TRUNCATE tabela` apaga todas as linhas de uma tabela de uma vez, muito mais depressa que um `DELETE` sem `WHERE`, porque não olha as linhas uma a uma. Ele não aceita `WHERE` — é tudo ou nada — e é o comando das tabelas de carga, esvaziadas antes de cada importação. No PostgreSQL, também ele pode ser desfeito dentro de uma transação.

> • **O ESSENCIAL**
> `INSERT INTO t (colunas) VALUES (...)`, com a lista de colunas sempre; `UPDATE t SET col = expr WHERE ...`; `DELETE FROM t WHERE ....` `RETURNING` devolve as linhas afetadas — no PostgreSQL 18, com `old` e `new` no `UPDATE`. A resposta do psql conta as linhas: `INSERT 0 1`, `UPDATE 3`, `DELETE 2`; `UPDATE 0` não é erro. Sem `WHERE`, `UPDATE` e `DELETE` valem para a tabela toda. `UPDATE ... FROM` e `DELETE ... USING` usam uma junção, e uma junção que acha duas linhas atualiza com qualquer uma. Os hábitos: o `WHERE` primeiro, o `SELECT count(*)` antes, o `BEGIN` em volta, o `RETURNING` para conferir.

ENQUANTO ISSO, NA EMPRESA **Seiscentos e quarenta**

A Bruna preparou o reajuste para a base de produção: um `SELECT count(*)` com o `WHERE` das duas categorias, que deu 640, e o `UPDATE` dentro de um `BEGIN`. Mandou os dois para o Seu Osvaldo antes de rodar. Ele rodou o `SELECT`, anotou o número num papel e passou o teclado de volta para ela.

— Roda. Se o `UPDATE` não disser 640, você dá `ROLLBACK` e a gente conversa.

Disse 640. Ela digitou `COMMIT` devagar, letra por letra.

RESUMO

- `INSERT INTO t (colunas) VALUES (...) RETURNING ...`: escreva a lista de colunas; `RETURNING` mostra o que entrou.
- `UPDATE t SET a = x, b = y WHERE ... RETURNING old.a, new.a.`
- A resposta do psql conta as linhas afetadas; `UPDATE 0` não é erro.
- Sem `WHERE`, todas as linhas. Hábitos: `WHERE` primeiro, `SELECT count(*)` antes, `BEGIN`, `RETURNING`.
- `UPDATE ... FROM` e `DELETE ... USING`: com junção; confira a cardinalidade antes.
- Apagar o que outros apontam é recusado; produto fora de linha fica inativo.

1 A Dona Zilda vendeu duas caixas de dipirona do lote 4 no balcão, fora do sistema. Baixe as duas caixas do estoque do Centro e mostre a quantidade de antes e a de depois.

2 ·· Sem rodar, diga o que acontece com o preço do sérum, e por quê.

NORUN sql

```sql
UPDATE produtos SET preco = preco * 1.1 WHERE id = 6
RETURNING preco;
```

3 ··· O Rafael quer apagar os clientes que não compraram nada em dezembro. Escreva o `SELECT` que mostra quem seriam eles, antes de qualquer `DELETE`, e diga por que o `DELETE` não deve ser escrito.

CAPÍTULO

26

Transações

Cinco caixas de dipirona que aparecem em Itaúna sem ter saído do Centro, uma transferência que para no meio e não deixa rastro, um COMMIT que responde ROLLBACK e dois terminais olhando a mesma linha ao mesmo tempo.

AO FIM DESTE CAPÍTULO
Agrupar comandos numa transação com BEGIN, COMMIT e ROLLBACK, entender o que acontece quando um comando falha no meio dela, desfazer só uma parte com SAVEPOINT e ver, com duas sessões abertas, o que uma transação mostra às outras antes e depois de confirmada.

Capítulo disponível na versão completa

CAPÍTULO

27

A madrugada das 3h

Uma soma que muda no meio da mesma transação, uma fotografia do estoque que não muda nem quando o estoque muda, um pedido automático que grava um número velho, e o deadlock que derrubava o job das três da manhã duas vezes por semana.

AO FIM DESTE CAPÍTULO
Entender o que uma transação vê das outras em cada nível de isolamento, reconhecer a leitura não repetível e a atualização perdida, bloquear linhas com SELECT FOR UPDATE, provocar e ler um deadlock e escrever transações que não caem nele.

Capítulo disponível na versão completa

CAPÍTULO

28

Cargas

Um arquivo de sete linhas que o banco recusa por completo por causa da palavra dois, uma loja BE-04 que não existe, um cupom que chega pela segunda vez, um preço que entra ou atualiza conforme já exista e uma tabela de preços sincronizada num comando só.

AO FIM DESTE CAPÍTULO
Carregar arquivos CSV com COPY, receber os dados numa tabela de carga e validá-los com consultas antes de passá-los às tabelas de verdade, inserir sem duplicar com ON CONFLICT, sincronizar uma tabela com outra com MERGE e escrever cargas que podem rodar duas vezes sem estrago.

Capítulo disponível na versão completa

PARTE 6

Desempenho e operação

Índices, planos de execução, visões, gatilhos, permissões e o backup que ninguém tinha testado.

NESTA PARTE

29 Índices
30 Ler um plano
31 Visões, funções e gatilhos
32 Quem pode ver o quê
33 Backup

CAPÍTULO

29
Índices

Trezentas mil vendas lidas para achar seis, um índice que transforma catorze milissegundos em menos de um, uma função em volta da coluna que faz o banco ignorar o índice, uma pergunta larga demais para ele valer a pena e o preço que todo índice cobra a cada INSERT.

AO FIM DESTE CAPÍTULO
Entender por que uma consulta lê a tabela toda, criar índices B-tree e ver no plano de execução quando o banco os usa, escolher a ordem das colunas de um índice composto, escrever condições que o índice consegue aproveitar e pesar o que um índice custa.

Capítulo disponível na versão completa

CAPÍTULO

30

Ler um plano

Um banco que acha que vai encontrar uma linha e encontra cem mil, três jeitos diferentes de juntar as mesmas duas tabelas, um ranking que ordena trezentas mil vendas para mostrar cinco e o roteiro de cinco perguntas que a Bruna colou no monitor.

AO FIM DESTE CAPÍTULO

Ler um plano de execução com custos e estimativas, comparar o que o banco estimou com o que aconteceu, reconhecer estatísticas velhas, distinguir os algoritmos de junção — laço aninhado, hash, merge —, entender como um ORDER BY com LIMIT é resolvido e seguir um roteiro para achar o passo caro de qualquer plano.

Capítulo disponível na versão completa

CAPÍTULO

31

Visões, funções e gatilhos

Cinco pessoas que calculavam o faturamento de cinco jeitos e passam a ler a mesma visão, um resumo que fica velho até alguém mandar atualizar, a margem que vira uma função, o preço que deixa rastro quando muda e a regra da Dra. Lúcia que não cabia num CHECK.

AO FIM DESTE CAPÍTULO
Guardar consultas no banco como visões e visões materializadas, criar funções em SQL que recebem parâmetros e devolvem valores ou tabelas, e escrever gatilhos que reagem a mudanças nas tabelas para registrar histórico e cobrar regras que olham mais de uma tabela.

Capítulo disponível na versão completa

CAPÍTULO

32

Quem pode ver o quê

Uma planilha de receitas de controlados com o CPF de cada cliente, mandada por e-mail para o financeiro todo; um usuário que pode ler as vendas e não pode apagá-las; uma gerente que só enxerga a própria loja; e a busca por cidade que devolve todos os clientes quando alguém digita uma aspa.

AO FIM DESTE CAPÍTULO
Controlar o acesso aos dados com papéis e privilégios — GRANT e REVOKE, por tabela e por coluna —, restringir linhas com segurança por linha, entender e evitar a injeção de SQL e expor dados sensíveis só pelo que cada função precisa ver.

Capítulo disponível na versão completa

CAPÍTULO

33
Backup

A foto do UPDATE 3104 de 2019 e as três horas de vendas que se perderam para consertá-lo; uma cópia do banco num arquivo; a mesma cópia devolvida num banco de ensaio; os preços de ontem trazidos de volta sem perder as vendas de hoje; e o backup que só vale depois do primeiro ensaio.

AO FIM DESTE CAPÍTULO
Copiar um banco com pg*dump, nos formatos de texto e personalizado, restaurá-lo com psql e pg*restore num banco novo, recuperar uma tabela só sem desfazer o resto, entender o backup físico e a recuperação até um instante, e fazer do ensaio de restauração uma rotina.

Capítulo disponível na versão completa

PARTE 7

Os bancos que você vai encontrar

PostgreSQL, MySQL, SQLite, SQL Server e Oracle — e a reunião de 15 de março.

NESTA PARTE

34 PostgreSQL e MySQL

35 SQLite, SQL Server e Oracle

36 Quinze de março

CAPÍTULO

34

PostgreSQL e MySQL

Uma chave estrangeira que o MySQL aceita e ignora, uma cidade digitada sem acento que ele encontra assim mesmo, um operador || que vira zero, uma divisão de inteiros com quatro casas, um ROLLBACK que não desfaz o CREATE TABLE — e as mesmas consultas da Santa Rita, rodadas nos dois bancos.

AO FIM DESTE CAPÍTULO

Levar a base e as consultas da Santa Rita para o MySQL, conhecer as diferenças que mudam resultados — tipos, chaves, comparação de textos, divisão, concatenação, ordem do NULL, escrita, transações — e escrever SQL que funciona nos dois quando é preciso.

Capítulo disponível na versão completa

CAPÍTULO

35

SQLite, SQL Server e Oracle

Um banco completo dentro de um arquivo, que aceita a loja de metragem 'a combinar' e uma venda numa loja que não existe; uma soma de preços que não é igual a ela mesma; e a mesma consulta da Santa Rita escrita nos dialetos dos dois bancos comerciais que a Bem-Estar e o banco do financiamento usam.

AO FIM DESTE CAPÍTULO

Conhecer o SQLite — um banco sem servidor, num arquivo — e as liberdades que ele toma com tipos e chaves, e reconhecer o que muda no SQL Server e no Oracle: limitar linhas, emendar textos, gerar chaves, datas, upsert e transações, para escrever SQL que atravessa os cinco bancos.

Capítulo disponível na versão completa

CAPÍTULO

36

Quinze de março

O estoque lote por lote, com o que venceu e o que vai vencer; a margem de cada loja, com a conta que dá para refazer; uma consulta que confere a si mesma; e a reunião em que o analista do banco pede para rodar os números ali, na frente de todo mundo.

AO FIM DESTE CAPÍTULO
Montar as três respostas que o banco pediu — o estoque com os vencidos, a margem por loja e a soma que fecha — com as ferramentas do livro todo, guardá-las de um jeito que qualquer um possa reproduzir e escrever as consultas que conferem os próprios números.

Capítulo disponível na versão completa

A base da Santa Rita

AO FIM DESTE CAPÍTULO
Os scripts SQL que criam as tabelas da Drogarias Santa Rita e as preenchem com as vendas de dezembro de 2026 de três lojas, a amostra da Rede Bem-Estar, a remessa em CSV e o esquema traduzido para o MySQL.

Capítulo disponível na versão completa

Referência rápida

AO FIM DESTE CAPÍTULO
A sintaxe do livro em tabelas, para consultar sem reler os capítulos: a ordem de leitura, as partes de uma consulta, junções, agregação, janelas, escrita, estrutura, transações, os comandos do psql, as diferenças entre os cinco bancos e as armadilhas, cada uma com a sua defesa.

Capítulo disponível na versão completa

Onde praticar

AO FIM DESTE CAPÍTULO
As bancadas do livro — PostgreSQL, MySQL e SQLite —, como recomeçar a base, como trabalhar em cópias e como fazer a base crescer para praticar com volume, sem nenhum arquivo além dos que o livro mostra.

Capítulo disponível na versão completa

Respostas

A planilha de 1,2 milhão de linhas

1

```
SELECT nome, cidade, fidelidade FROM clientes;
```

```
            nome           |    cidade     | fidelidade
 ------------------------+---------------+------------
  Maria Aparecida Souza  | Divinópolis   | ouro
  José Carlos Lima       | Divinópolis   | prata
  Juliana Rocha Alves    | Itaúna        | bronze
  Geraldo Nunes          | Divinópolis   |
  Fernanda Dias Costa    | Itaúna        |
  Paulo Henrique Martins | Pará de Minas | bronze
  Raquel Teixeira        | Divinópolis   |
  Sebastião Ferreira     | Itaúna        | prata
 (8 rows)
```

Três clientes — Geraldo, Fernanda e Raquel — aparecem com a coluna `fidelidade` vazia: não estão no programa.

2

```
SELECT id, momento, pagamento FROM vendas;
```

```
  id |        momento          | pagamento
 ----+-------------------------+-----------
    1 | 2026-12-01 08:14:05-03 | pix
    2 | 2026-12-01 10:02:44-03 | dinheiro
    3 | 2026-12-02 15:31:10-03 | débito
    4 | 2026-12-03 09:47:52-03 | crédito
    5 | 2026-12-04 18:20:33-03 | crédito
    6 | 2026-12-07 11:05:19-03 | dinheiro
    7 | 2026-12-09 14:40:01-03 | pix
    8 | 2026-12-11 16:12:47-03 | crédito
    9 | 2026-12-14 09:30:00-03 | pix
   10 | 2026-12-16 19:55:12-03 | débito
   11 | 2026-12-18 10:21:36-03 | pix
   12 | 2026-12-21 13:03:58-03 | crédito
   13 | 2026-12-23 17:48:09-03 | crédito
   14 | 2026-12-28 08:59:41-03 | dinheiro
   15 | 2026-12-30 20:10:27-03 | pix
 (15 rows)
```

Os cupons aparecem em ordem de número e de hora, porque foram gravados nessa ordem. Mas uma tabela não tem ordem própria: o banco devolve as linhas na ordem que for mais conveniente para ele, e essa ordem pode mudar depois que linhas forem alteradas ou apagadas. O `-03` no fim do momento é o fuso horário: três horas a menos que o horário de Greenwich, o de Brasília.

3 ··

```
ERROR:  relation "cliente" does not exist
LINE 1: SELECT nome, cidad FROM cliente;
                               ^
```

O banco começa pelo `FROM`: antes de procurar as colunas, ele precisa saber de que tabela elas são, e `cliente` não existe — a tabela é `clientes`, no plural. Com o nome da tabela certo, aparece o segundo erro:

QUEBRADO sql

```sql
SELECT nome, cidad FROM clientes;
```

```
ERROR:  column "cidad" does not exist
LINE 1: SELECT nome, cidad FROM clientes;
                     ^
HINT:  Perhaps you meant to reference the column
"clientes.cidade".
```

Agora a dica aponta o caminho: `cidade`. Um erro por vez é o normal: o banco para no primeiro que encontra, e o segundo só aparece quando o primeiro sai do caminho.

Uma linguagem de 1974

1 O programa do modelo de rede navega por ponteiros num caminho que ele mesmo define; se a estrutura das ligações muda, o caminho pode deixar de existir ou mudar de significado, e o programa precisa ser reescrito. A consulta SQL não diz caminho nenhum: diz o resultado. Um índice novo só dá ao otimizador mais uma opção de caminho, e a consulta continua igual — no máximo, fica mais rápida.

2 ·· Primeiro, a ordem: um conjunto não tem ordem, então o banco pode devolver as linhas em qualquer ordem, e ela pode mudar de uma execução para outra; quem precisa de uma ordem precisa pedi-la. Segundo, as repetições: num conjunto, duas linhas idênticas não fazem sentido; uma tabela sem nada que distinga uma linha da outra vira um problema na hora de alterar ou apagar uma delas, e é por isso que toda tabela precisa de alguma coisa que identifique cada linha.

Montando a bancada

1 Em `\d produtos`, a coluna *Nullable* traz `not null` em todas as colunas menos três: `fornecedor_id`, `custo` e `tarja`. Um produto pode não ter fornecedor (os serviços, como a aferição de pressão), pode estar sem custo no cadastro e só tem tarja se for medicamento de tarja.

2 ..

NORUN sql

```
\x
SELECT * FROM lojas WHERE id = 3;
```

O registro aparece em lista, com `metragem` sem nada depois da barra: o campo não tem valor. No modo normal ou no modo em lista, o psql mostra um valor ausente do mesmo jeito, como um espaço vazio.

O faturamento que dobrou

1

```
SELECT f.nome, coalesce(c.nome, '(ninguém)') AS chefe
FROM funcionarios f
LEFT JOIN funcionarios c ON f.chefe_id = c.id
WHERE f.loja_id IS NULL
ORDER BY f.id;
```

```
1         nome       |      chefe
2   -----------------+-----------------
3    Cláudia Menezes | (ninguém)
4    Osvaldo Ribeiro | Cláudia Menezes
5    Lúcia Amaral    | Cláudia Menezes
6    Bruna Teixeira  | Osvaldo Ribeiro
7   (4 rows)
```

O `WHERE` é sobre `f`, o lado que não falta, e por isso não desfaz o `LEFT JOIN`.

2 .. Cada item forma par com todos os lotes do seu produto. A losartana tem 5 itens e 2 lotes: 10 pares. A dipirona, 5 e 2: 10. A amoxicilina, 3 e 2: 6. Clonazepam, protetor, sérum, fralda infantil e loratadina têm um lote cada, e dão um par por item: 2 + 3 + 1 + 4 + 2 = 12. A aferição, com 3 itens e nenhum lote, dá 0. Total: 10 + 10 + 6 + 12 = 38 — a contagem que a conferência mostrou. A cardinalidade prevê o tamanho de uma junção antes de rodá-la.

3 ...

```
SELECT f.nome, f.salario, c.nome AS chefe,
       c.salario AS salario_chefe
FROM funcionarios f
JOIN funcionarios c ON f.chefe_id = c.id
WHERE f.salario > c.salario * 0.6
ORDER BY f.id;
```

```
     nome     | salario  |      chefe      | salario_chefe
--------------+----------+-----------------+---------------
 Lúcia Amaral | 13900.00 | Cláudia Menezes |      21800.00
 Ana Mendes   |  4950.00 | Zilda Moreira   |       7200.00
(2 rows)
```

A condição compara duas linhas da mesma tabela, o que só é possível porque a autojunção as pôs lado a lado, numa linha só do resultado. É para isso que ela existe: perguntas sobre a relação entre duas linhas de uma mesma tabela.

INSERT, UPDATE, DELETE

1

```
UPDATE estoque
SET quantidade = quantidade - 2
WHERE loja_id = 1 AND lote_id = 4
RETURNING loja_id, lote_id, old.quantidade AS antes,
          new.quantidade AS depois;
```

```
 loja_id | lote_id | antes | depois
---------+---------+-------+--------
       1 |       4 |    40 |     38
(1 row)

UPDATE 1
```

A chave do estoque é o par `(loja_id, lote_id)`, e o `WHERE` pelas duas colunas garante que uma linha só muda.

2 ..

129,90 vezes 1,1 dá 142,890, com três casas. A coluna é `numeric(10,2)`, e o valor é arredondado ao ser guardado: o `RETURNING` mostra `142.89`, o valor que ficou na tabela, e não o da conta. A resposta do psql é `UPDATE 1`.

3 ...

```
SELECT c.id, c.nome FROM clientes c
WHERE NOT EXISTS (SELECT 1 FROM vendas v
                  WHERE v.cliente_id = c.id);
```

```
 id | nome
----+------
(0 rows)
```

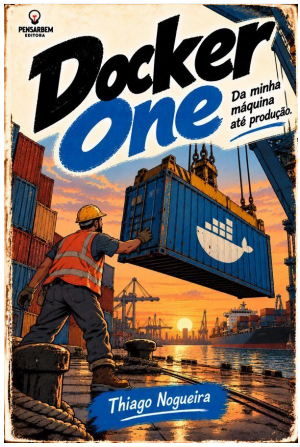
Nenhum: todos os oito clientes compraram em dezembro, e um `DELETE` com o mesmo `WHERE` daria `DELETE 0`. Mas o problema não é este mês. Um cliente sem compra em dezembro pode ter comprado em novembro, e apagar o cadastro apaga o histórico do programa de fidelidade e o consentimento dele. A pergunta “quem não comprou” é um relatório; “apagar quem não comprou” é uma regra de negócio, que precisa de alguém que a assine.

Glossário

Banco de dados relacional Um programa que guarda dados em tabelas com colunas de tipo definido, ligadas umas às outras por valores em comum, e que responde a perguntas escritas em SQL. O **PostgreSQL**, que a Santa Rita usa, é um deles; o MySQL, o SQL Server, o Oracle e o SQLite são outros.

Relação O nome que Codd deu, tirado da matemática dos conjuntos, ao que os bancos de dados chamam de **tabela**: um conjunto de linhas, todas com as mesmas colunas. Por ser um conjunto, as linhas não têm uma ordem própria, e duas linhas idênticas não fazem sentido.

Conheça as demais obras dessa coleção



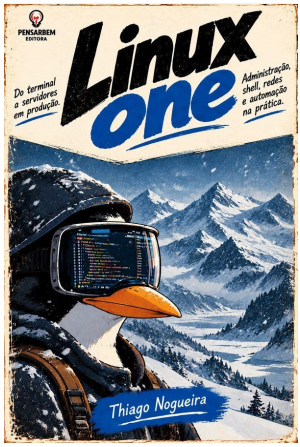
Docker One

Da Minha Máquina à Produção, Sem Mistério



Git One

Versionar, Voltar Atrás e Trabalhar em Equipe Sem Medo



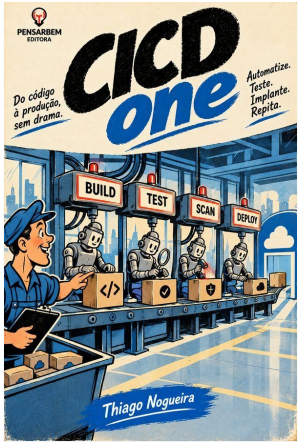
Linux One

Terminal, Shell e as Ferramentas que Todo Dev Usa Todo Dia



Cloud One

Fundamentos de Nuvem, com AWS, Google Cloud, Azure e DigitalOcean



CI/CD One

Integração e Entrega Contínuas, com GitHub Actions, GitLab CI, CircleCI e Jenkins

---

Visite a editora Pensar Bem →

pensarbem.com.br