Aula 4

1. Introdução à Análise de Séries Temporais

Uma **série temporal** é uma sequência de observações coletadas ao longo do tempo, geralmente em intervalos regulares, como dias, horas ou minutos. A ordem dos dados é essencial, pois o valor de um momento pode depender de valores anteriores.

- **Importância:** permite identificar tendências, padrões sazonais e realizar previsões.
- **Exemplos:** vendas diárias, logs de acesso, leituras de sensores, preços de ações, temperatura.

Exemplo de tabela:

Data/Hora (UTC)	Vendas	Temperatura (°C)
2025-09-01 08:00:00	15	22.5
2025-09-01 09:00:00	18	23.0
2025-09-01 10:00:00	20	24.1
2025-09-01 11:00:00	25	25.0

Tabela 1: Exemplo de série temporal. Todas as datas estão em UTC, que é o horário universal coordenado, referência global sem alteração por fusos ou horário de verão.

2. Tipos de Dados de Data e Hora no SQL

No SQL existem diferentes tipos para armazenar informações temporais:

- `DATE`: armazena apenas a data (ano, mês, dia).
- `TIME`: armazena apenas a hora (hora, minuto, segundo).
- `DATETIME` / `TIMESTAMP`: armazenam data e hora completas.

Boas práticas:

- Sempre use `TIMESTAMP` se precisar de registro completo de data e hora.
- Armazene datas no formato **UTC (Coordinated Universal Time)**, que é o padrão de tempo internacional usado como referência global, sem sofrer alterações por fusos horários ou horário de verão.

3. Funções Essenciais de Data e Hora

Extração de componentes de datas

Ano, Mês, Dia, Hora, Minuto, Segundo:

```
SELECT EXTRACT(YEAR FROM datahora) AS ano,
       EXTRACT(MONTH FROM datahora) AS mes,
       EXTRACT(DAY FROM datahora) AS dia
FROM vendas;

SELECT DATE_PART('hour', datahora) AS hora,
       SUM(vendas) AS total_vendas
FROM vendas
GROUP BY hora
ORDER BY hora;
```

Obtenção da data e hora atuais

```
SELECT NOW() AS datahora_atual;
SELECT CURRENT_DATE AS data_atual;
SELECT CURRENT_TIMESTAMP AS timestamp_atual;
```

Formatação de datas e timestamps

```
SELECT TO_CHAR(datahora, 'YYYY-MM-DD HH24:MI:SS') AS data_formatada
FROM vendas;
```

Conversão de strings para tipos de data

```
SELECT TO_DATE('2025-09-26','YYYY-MM-DD') AS data_convertida;
SELECT CAST('2025-09-26' AS DATE) AS data_convertida;
```

4. Manipulação e Aritmética com Datas – Cesar

- Adição e subtração de intervalos de tempo (`INTERVAL`).
- Cálculo da diferença entre duas datas (subtração direta ou uso de `AGE`).
- Truncamento de datas para o início do período: `DATE_TRUNC`.

Na análise de séries temporais no PostgreSQL, frequentemente precisamos manipular datas: prever prazos adicionando intervalos, calcular o tempo decorrido entre eventos ou organizar registros em períodos bem definidos.

Suponha a seguinte tabela de entregas:

Adição de intervalos: podemos calcular a entrega prevista (5 dias após a venda) usando:

id_pedido	data_venda	data_entrega
1	2025-01-05 10:23:54	2025-01-10 15:00:00
2	2025-01-15 14:12:31	2025-01-20 09:30:00
3	2025-02-08 11:30:00	2025-02-18 16:45:00
4	2025-02-25 16:20:15	2025-02-28 10:00:00

```
SELECT
    id_pedido,
    data_venda,
    data_venda + INTERVAL '5 days' AS entrega_prevista
FROM entregas;
```

Diferença entre datas: para calcular quantos dias cada entrega levou:

```
SELECT
    id_pedido,
    EXTRACT(DAY FROM data_entrega - data_venda) AS dias_para_entrega
FROM entregas;
```

Truncamento de datas: para agrupar pedidos por mês:

```
SELECT
    DATE_TRUNC('month', data_venda) AS mes,
    COUNT(*) AS pedidos
FROM entregas
GROUP BY mes
ORDER BY mes;
```

Essas operações são fundamentais para ajustar prazos, avaliar eficiência logística e organizar relatórios em intervalos regulares.

5. Lidando com Fusos Horários (Time Zones) – Cesar

- A importância da padronização de fusos horários (e.g., UTC).
- Funções para conversão entre diferentes fusos horários (`AT TIME ZONE`).

Em sistemas distribuídos, registros podem vir de diferentes regiões do mundo. Para evitar inconsistências, a prática recomendada é armazenar todas as datas no fuso horário UTC. A partir disso, podemos converter para o horário local sempre que necessário.

Suponha a tabela de acessos a um sistema global:

Se quisermos visualizar os horários no fuso de São Paulo:

id_acesso	data_acesso (UTC)
1	2025-01-05 15:00:00+00
2	2025-01-05 22:30:00+00
3	2025-01-06 01:15:00+00
4	2025-01-06 10:45:00+00

```
SELECT
    id_acesso,
    data_acesso AT TIME ZONE 'UTC' AT TIME ZONE 'America/Sao_Paulo'
        AS horario_local
FROM acessos;
```

O resultado será:

id_acesso	horario_local (São Paulo)
1	2025-01-05 12:00:00
2	2025-01-05 19:30:00
3	2025-01-05 22:15:00
4	2025-01-06 07:45:00

Assim, conseguimos adaptar os relatórios para diferentes regiões sem comprometer a integridade da base original.

6. Agrupando Dados Temporais

Nas seções anteriores, abordamos conceitos iniciais para manipulação de tipos de dados temporais. A partir desta seção, começaremos a tratar de questões relacionadas à análise e entendimento de séries temporais, como verificar tendências e padrões num intervalo de tempo.

Uma das principais formas de se analisar dados temporais é realizando agrupamentos de acordro com o espaço de tempo: dias, semanas, meses, etc. Isso permite verificar mudanças de variáveis dentro de um intervalo regular e preciso de tempo, favorecendo a verificação de padrões e tendências. Para tanto, podemos utilizar o comando GROUP BY juntamente ao comando de truncamento de datas para gerar períodos de tempo. Para exemplificar, suponha que temos uma relação Vendas, a qual tem os seguintes atributos: id_venda, id_produto (INTEGER), data_venda (TIMESTAMP) e valor (NUMERIC). Para o ano de 2025, suponha que a empresa obteve os dados da tabela abaixo.

id_venda	id_produto	data_venda	valor
1	101	2025-01-05 10:23:54	150.00
2	102	2025-01-15 14:12:31	75.50
3	103	2025-01-15 19:45:10	299.90
4	101	2025-01-28 08:55:00	150.00
5	201	2025-02-08 11:30:00	45.00
6	202	2025-02-18 21:05:49	199.99
7	102	2025-02-25 16:20:15	75.50
8	301	2025-03-02 12:00:00	89.90
9	101	2025-03-10 13:45:00	149.90
10	302	2025-03-20 09:10:25	550.00
11	201	2025-03-20 18:25:33	45.00
12	103	2025-03-29 22:15:00	299.90
13	401	2025-04-15 10:05:12	1200.00
14	402	2025-04-15 15:30:00	3300.00
15	101	2025-04-16 09:00:45	150.00

Se quisermos obter informações acerca das vendas ocorridas mês a mês no primeiro quadrimestre do ano de 2025, poderiamos fazer:

```
SELECT
    DATE_TRUNC('month', data_venda) AS mes,
    COUNT(id_venda) AS numero_de_vendas,
    SUM(valor) AS receita_total,
    AVG(valor) AS ticket_medio
FROM vendas
WHERE data_venda >= '2025-01-01' AND data_venda < '2025-05-01'
GROUP BY mes
ORDER BY mes;
```

O que gera a tabela abaixo.

mes	numero_de_vendas	receita_total	ticket_medio
2025-01-01 00:00:00	4	675.40	168.85
2025-02-01 00:00:00	3	320.49	106.83
2025-03-01 00:00:00	5	1134.70	226.94
2025-04-01 00:00:00	8	14405.89	1800.74

Observe que as datas de venda serão arredondadas para o primeiro dia de cada mês. Ao realizar o agrupamento, datas como "2025-03-25 14:30:10"e "2025-03-07 09:25:45"serão tratadas como um único grupo ("2025-03-01 00:00:00"). Então, para cada grupo, calcularemos a soma e a média dos valores arrecadados de vendas. Com isso, podemos realizar relatórios temporais dos dados, o que facilita o entendimento e a análise dos dados.

7. Análise de Tendências Simples

A partir da técnica de agrupamento mostrada na seção anterior, podemos determinar características relacionadas aos dados. Para isso, é necessário observar um aspecto importante: a tendência dos dados. A tendência de um conjunto de dados é a direção de evolução das informações (crescimento, estagnação e decrescimento, por exemplo) ao longo de um período de tempo. É com base nela que podemos buscar informações acerca de comportamentos dos dados.

Na seção anterior, utilizamos um exemplo baseado nas vendas de uma empresa. Ao executar o código mostrado em uma base de dados, poderíamos obter dados como: tendência geral dos dados (de aumento e de queda) e melhores e piores meses. Com isso, insights e análises adicionais poderiam ser feitas com maior facilidade sobre os dados.

Outro exemplo poderia ser de um site que mantém uma relação com os dias e horários que um usuário acessou a plataforma, conforme segue.

id_acesso	id_usuario	data_acesso
1	123	2025-03-20 09:05:11
2	456	2025-03-20 09:15:21
3	789	2025-03-20 10:30:00
4	123	2025-03-20 11:12:45
5	101	2025-03-20 14:00:19
6	456	2025-03-20 15:22:30
7	123	2025-03-20 20:05:00
8	456	2025-03-21 08:30:00
9	789	2025-03-21 10:45:10
10	222	2025-03-21 11:00:00
11	123	2025-03-21 13:20:15
12	456	2025-03-21 17:50:00
13	789	2025-03-22 11:05:00
14	333	2025-03-22 15:25:30
15	123	2025-03-22 18:00:00
16	456	2025-03-23 16:40:10
17	101	2025-03-23 20:55:00
18	123	2025-03-24 09:00:00
19	456	2025-03-24 09:10:00
20	789	2025-03-24 10:15:20
21	222	2025-03-24 11:30:00
22	101	2025-03-24 14:20:50
23	123	2025-03-24 15:00:00
24	456	2025-03-24 16:45:10
25	789	2025-03-25 08:55:00
26	123	2025-03-25 09:30:00
27	456	2025-03-25 10:00:00
28	333	2025-03-25 12:10:15
29	101	2025-03-25 14:40:00
30	222	2025-03-25 15:30:45

Suponha que queremos verificar o dia com a maior quantidade de acessos. Para isso, podemos fazer:

```
SELECT
    EXTRACT(DOW FROM data_acesso) AS dia_da_semana,
    TO_CHAR(data_acesso, 'Day') AS nome_do_dia,
    COUNT(DISTINCT id_usuario) AS usuarios_ativos
FROM acessos_site
GROUP BY dia_da_semana, nome_do_dia
ORDER BY dia_da_semana;
```

O comando EXTRACT(DOW ...) extrai o dia da semana, sendo domingo correspondente a 0 e sábado correspondente a 6. A tabela resultante da consulta será:

dia_da_semana	nome_do_dia	usuarios_ativos
0	Sunday	2
1	Monday	5
2	Tuesday	6
4	Thursday	4
5	Friday	4
6	Saturday	3

Logo, podemos observar que os dias de semana são aqueles mais utilizados pelos usuários ao acessar o site. Isso pode ser usado, posteriormente, para desenvolver campanhas e promoções, indicando em quais dias publicar os eventos.

8. Geração de Séries Temporais Completas

Nas seções anteriores, observamos o uso de GROUP BY para agregar dados temporais e analisar tendências. Contudo, essa técnica, embora eficiente, possui uma limitação crítica que pode invalidar nossas conclusões: ela tem problemas com dados ausentes.

A consulta GROUP BY só pode relatar os períodos em que ocorreram eventos. Na relação de acessos ao site, se um dia não tivesse nenhum registro (em caso de manutenção da plataforma, por exemplo), este não estaria presente na tabela resultante. Dentre as consequências disso, podemos citar:

- Visualizações Enganosas: Em um gráfico de linhas, a ausência do dia com "zero acessos"faria com que a linha conectasse diretamente o dia anterior ao posterior, escondendo a queda e distorcendo a percepção da continuidade e da volatilidade dos dados.
- Cálculos Incorretos: Métricas como "média móvel de 7 dias"se tornariam imprecisas, pois seriam calculadas sobre menos de 7 pontos de dados.

Para que nossas análises, gráficos e futuros modelos de previsão sejam confiáveis, precisamos de uma série temporal contínua. Portanto, o próximo passo é aprender a técnica para preencher essas lacunas, garantindo que cada período de tempo em nosso intervalo de análise esteja devidamente representado.

A estratégia para reparar dados ausentes é criar uma nova tabela com todas as datas possíveis de acordo com o tipo de período utilizado (dia, semana, mês, etc.). A partir disso, podemos realizar um LEFT JOIN entre as relações e, para datas inexistentes, substituir os valores nulos por valores padrões (conforme visto na aula 3). Para isso, vamos usar a função GENERATE_SERIES(Inicio, Fim, Intervalo), onde delimitamos datas de início e fim de um período e a forma de acréscimo (intervalo) do período, como dias, semanas, meses e anos. Observe o exemplo abaixo.

```
SELECT generate_series(
    '2025-04-15'::date,
    '2025-04-19'::date,
    '1 day'::interval
) AS dia;
```

Ao executar algo semelhante ao código acima, teríamos uma relação com a que segue.

dia
2025-04-15 00:00:00
2025-04-16 00:00:00
2025-04-17 00:00:00
2025-04-18 00:00:00
2025-04-19 00:00:00

Se a relação base não possui, por exemplo, o dia 16, então, ao realizar um LEFT JOIN, o registro correspondente ao dia 16 terá valores nulos.

Suponha, dada a situação anterior, que a tabela de acessos ao site não possui dados do dia 25/03, como ilustrado abaixo:

id_acesso	id_usuario	data_acesso
1	123	2025-03-20 09:05:11
2	456	2025-03-20 09:15:21
3	789	2025-03-20 10:30:00
4	123	2025-03-20 11:12:45
5	101	2025-03-20 14:00:19
6	456	2025-03-20 15:22:30
7	123	2025-03-20 20:05:00
8	456	2025-03-21 08:30:00
9	789	2025-03-21 10:45:10
10	222	2025-03-21 11:00:00
11	123	2025-03-21 13:20:15
12	456	2025-03-21 17:50:00
13	789	2025-03-22 11:05:00
14	333	2025-03-22 15:25:30
15	123	2025-03-22 18:00:00
16	456	2025-03-23 16:40:10
17	101	2025-03-23 20:55:00
18	123	2025-03-24 09:00:00
19	456	2025-03-24 09:10:00
20	789	2025-03-24 10:15:20
21	222	2025-03-24 11:30:00
22	101	2025-03-24 14:20:50
23	123	2025-03-24 15:00:00
24	456	2025-03-24 16:45:10

Logo, podemos visualizar as tendências por dia a partir do seguinte código:

```
SELECT
    c.dia_completo,
    COALESCE(COUNT(*), 0) AS acessos_diarios
FROM (
    SELECT generate_series(
        '2025-03-20'::date,
        '2025-03-25'::date,
        '1 day'::interval
    )::date AS dia_completo
) c
LEFT JOIN acessos_site a ON DATE_TRUNC('day', a.data_acesso) = c.dia_completo
GROUP BY c.dia_completo
ORDER BY c.dia_completo;
```

O que retornará a seguinte tabela:

dia_completo	acessos_diarios
2025-03-20 00:00:00	7
2025-03-21 00:00:00	5
2025-03-22 00:00:00	3
2025-03-23 00:00:00	2
2025-03-24 00:00:00	7
2025-03-25 00:00:00	0

9. Janelas Móveis (Rolling Time Windows) – Demba

Em séries temporais, não basta observar apenas os valores absolutos de cada período. Muitas vezes, é necessário suavizar as variações de curto prazo ou identificar tendências ocultas. Para isso, utilizamos as **funções de janela móveis** (`Window Functions`) que permitem calcular médias, somas e contagens sobre um intervalo de linhas definido em torno da linha atual.

Essas técnicas são fundamentais em análises financeiras, monitoramento de acessos e acompanhamento de métricas de desempenho.

—

9.1 Média Móvel (Moving Average)

A média móvel calcula a média dos últimos n períodos, suavizando oscilações momentâneas. Isso ajuda a responder perguntas como: *Qual foi a média de receita nos últimos 3 meses?*

Exemplo: Suponha a tabela `VendasMensais`:

mes	receita
2025-01-01	10000
2025-02-01	12000
2025-03-01	15000
2025-04-01	13000
2025-05-01	17000

Consulta SQL para calcular a média móvel de 3 meses:

```
SELECT
    mes,
    receita,
    ROUND(
        AVG(receita) OVER (
            ORDER BY mes
            ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
        ), 2
    ) AS media_movel_3m
FROM VendasMensais;
```

Resultado:

mes	receita	media_movel_3m
2025-01-01	10000	10000.00
2025-02-01	12000	11000.00
2025-03-01	15000	12333.33
2025-04-01	13000	13333.33
2025-05-01	17000	15000.00

—

9.2 Soma e Contagem Móveis

Além de médias, também é possível calcular somas ou contagens dentro da janela definida.
Exemplo: Soma acumulada dos últimos 3 meses:

```
SELECT
    mes,
    receita,
    SUM(receita) OVER (
        ORDER BY mes
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ) AS soma_movel_3m
FROM VendasMensais;
```

Resultado:

mes	receita	soma_movel_3m
2025-01-01	10000	10000
2025-02-01	12000	22000
2025-03-01	15000	37000
2025-04-01	13000	40000
2025-05-01	17000	45000

—

9.3 Janelas Deslizantes vs. Acumuladas

Existem dois tipos principais de análise com funções de janela:

- **Janela deslizante (Rolling Window):** considera apenas os últimos n registros (ex.: últimos 3 meses).
- **Janela acumulada (Cumulative Window):** considera todos os registros desde o início até a linha atual.

Exemplo de soma acumulada:

```
SELECT
    mes,
    receita,
    SUM(receita) OVER (ORDER BY mes) AS soma_acumulada
FROM VendasMensais;
```

—

Resumo: As janelas móveis são ferramentas poderosas para suavizar flutuações em séries temporais, identificar tendências e analisar padrões de comportamento. O uso de funções como `AVG`, `SUM` e `COUNT` com cláusulas `OVER` permite criar métricas derivadas sem subconsultas adicionais, tornando o código mais limpo e eficiente.

10. Comparação com Períodos Anteriores – Araujo

- Uso das funções `LAG` e `LEAD` para acessar dados de períodos anteriores ou posteriores.
- Cálculo do crescimento percentual período a período (mês a mês, ano a ano).
- Análise de sazonalidade comparando o mesmo período em anos diferentes.

10. Comparação com Períodos Anteriores – Araujo

Em análises temporais, muitas vezes não basta saber o valor absoluto de uma métrica (como receita, acessos ou número de vendas). É essencial entender como esses valores evoluem ao longo do tempo. Para isso, utilizamos técnicas de comparação com períodos anteriores, que ajudam a responder perguntas como:

- A receita deste mês cresceu ou caiu em relação ao mês passado?
- Como está o desempenho deste ano em relação ao ano anterior?
- Existe sazonalidade (mesmo mês em anos diferentes mostra comportamento parecido)?

No PostgreSQL, podemos utilizar funções de janela como `LAG` e `LEAD` para acessar valores de períodos anteriores e posteriores sem precisar de `JOIN` adicionais.

—

10.1 Funções `LAG` e `LEAD`

- `LAG(coluna)`: acessa o valor da linha anterior dentro da partição/ordenação definida.
- `LEAD(coluna)`: acessa o valor da linha seguinte.

Exemplo: Suponha a tabela `VendasMensais`:

mes	receita
2025-01-01	10000
2025-02-01	12000
2025-03-01	15000
2025-04-01	13000

Podemos comparar cada mês com o mês anterior:

```
SELECT
    mes,
    receita,
    LAG(receita) OVER (ORDER BY mes) AS receita_anterior
FROM VendasMensais;
```

Resultado:

mes	receita	receita_anterior
2025-01-01	10000	NULL
2025-02-01	12000	10000
2025-03-01	15000	12000
2025-04-01	13000	15000

—

10.2 Crescimento Percentual Período a Período

A partir da função `LAG`, podemos calcular a variação percentual mês a mês:

```
SELECT
    mes,
    receita,
    LAG(receita) OVER (ORDER BY mes) AS receita_anterior,
    ROUND(
        (receita - LAG(receita) OVER (ORDER BY mes))
        / LAG(receita) OVER (ORDER BY mes) * 100, 2
    ) AS crescimento_percentual
FROM VendasMensais;
```

Resultado esperado:

mes	receita	receita_anterior	crescimento%
2025-01-01	10000	NULL	NULL
2025-02-01	12000	10000	20.00
2025-03-01	15000	12000	25.00
2025-04-01	13000	15000	-13.33

Esse tipo de análise é fundamental para entender tendências de curto prazo.

—

10.3 Comparação Anual (Sazonalidade)

Além da comparação mês a mês, podemos verificar o comportamento em anos diferentes, útil para detectar **sazonalidade**.

Exemplo: Suponha a tabela `ReceitaMensal` com dois anos de dados:

mes	receita
2024-01-01	9000
2024-02-01	11000
2025-01-01	10000
2025-02-01	12000

Consulta para comparar meses iguais em anos diferentes:

```
SELECT
    EXTRACT(YEAR FROM mes) AS ano,
    EXTRACT(MONTH FROM mes) AS mes_num,
    receita,
    LAG(receita) OVER (PARTITION BY EXTRACT(MONTH FROM mes)
                       ORDER BY mes) AS receita_ano_anterior
FROM ReceitaMensal;
```

Resultado esperado:

ano	mes	receita	receita_ano_anterior
2024	1	9000	NULL
2025	1	10000	9000
2024	2	11000	NULL
2025	2	12000	11000

Assim, conseguimos comparar diretamente o desempenho de janeiro de 2025 com janeiro de 2024, fevereiro com fevereiro, e assim por diante.

—

11. Análise de Contribuição e Proporção

Em uma análise de dados, ver os números totais é apenas o primeiro passo. É muito importante entender a contribuição de cada componente para o resultado geral. É preciso saber quais partes são mais responsáveis por um resultado e como a importância de cada uma muda ao longo do tempo.

Para fazer isso, calculamos o percentual que cada parte representa do total em cada período. As Funções de Janela do SQL com a cláusula PARTITION BY são a ferramenta ideal para essa tarefa.

Suponha a tabela abaixo

mes	categoria	receita_categoria
2025-01-01	Eletrônicos	5000
2025-01-01	Livros	3000
2025-01-01	Vestuário	2000
2025-02-01	Eletrônicos	6000
2025-02-01	Livros	2500
2025-02-01	Vestuário	3500

Quando usamos uma função de janela móvel, podemos passar a cláusula PARTITION BY dentro o OVER (), o cálculo da função de janela é feito para cada partição. Temos um exemplo de um código abaixo, que pega a contribuição percentual de uma categoria na receita em cada mês.

```
SELECT
    mes,
    categoria,
    receita_categoria,
    ROUND(
      (receita_categoria * 100.0 / SUM(receita_categoria) OVER (PARTITION BY mes))
    , 2) AS contribuicao_percentual
FROM
    VendasPorCategoria
ORDER BY
    mes, categoria;
```

Se a nossa nova coluna com os percentuais fosse apenas:

```
    ROUND(
        (receita_categoria * 100.0 / SUM(receita_categoria))
    , 2) AS contribuicao_percentual
```

Então as linhas seriam apenas o percentual de receita ao longo de todos os tempos. Por exemplo, o total de todas as 6 receitas é 22000, enquanto os livros dão 5500 da receita, logo, o percentual seria 25%:

mes	categoria	receita_categoria	contribuicao_percentual
2025-01-01	Eletrônicos	5000	50
2025-01-01	Livros	3000	25
2025-01-01	Vestuário	2000	25
2025-02-01	Eletrônicos	6000	50
2025-02-01	Livros	2500	25
2025-02-01	Vestuário	3500	25

Todavia, como pedimos para particionar, o cálculo será feito considerando apenas as linhas que possuem o mês. Por exemplo, no primeiro mês, o total da receita é 10000, enquanto o livro tem uma receita de 3000 (ou seja, 30% de contribuição para a receita):

mes	categoria	receita_categoria	contribuicao_percentual
2025-01-01	Eletrônicos	5000	50
2025-01-01	Livros	3000	30
2025-01-01	Vestuário	2000	20
2025-02-01	Eletrônicos	6000	50
2025-02-01	Livros	2500	21
2025-02-01	Vestuário	3500	29

12. Indexação para Comparação de Crescimento

Comparar o crescimento de diferentes séries temporais, como produtos com datas de lançamento diferentes, pode ser enganoso se usarmos apenas seus valores absolutos. Uma forma de fazer uma comparação mais correta é verificar o crescimento percentual relativo a um ponto de partida.

Por exemplo, na tabela abaixo com dois produtos A, B:

mes	id_produto	receita
2025-01-01	A	1000
2025-02-01	A	1200
2025-03-01	A	1500
2025-02-01	B	500
2025-03-01	B	750
2025-04-01	B	1100

Podemos criar uma nova coluna com esse crescimento percentual em relação ao primeiro mês que o produto apareceu. Então de janeiro a março, A cresceu 150%, enquanto de fevereiro a abril, o produto B cresceu 220%.

Vejamos o código de SQL abaixo:

```
WITH VendasComValorInicial AS (
    SELECT
        mes,
        id_produto,
        receita,
     FIRST_VALUE(receita) OVER (PARTITION BY id_produto ORDER BY mes) AS receita_inicial
    FROM
        VendasDeProdutos
)
SELECT
    mes,
    id_produto,
    receita,
    ROUND(
        (receita * 100.0 / receita_inicial)
```

```
    , 2) AS indice_crescimento
FROM
    VendasComValorInicial
ORDER BY
    id_produto, mes;
```

O primeiro trecho define essa nova tabela que vamos operar, que simplesmente cria uma coluna nova com a receita inicial daquele produto (isso é feito pegando a primeira linha na partição de cada produto, ordenado pelo mês).

Então é feito um cálculo da variação e coloca nessa coluna chamada indice_crescimento. O resultado é:

mes	categoria	receita_categoria	indice_crescimento
2025-01-01	A	1000	100.00
2025-02-01	A	1200	120.00
2025-03-01	A	1500	150.00
2025-02-01	B	500	100.00
2025-03-01	B	750	150.00
2025-04-01	B	1100	220.00