

PostgreSQL

Aula 03

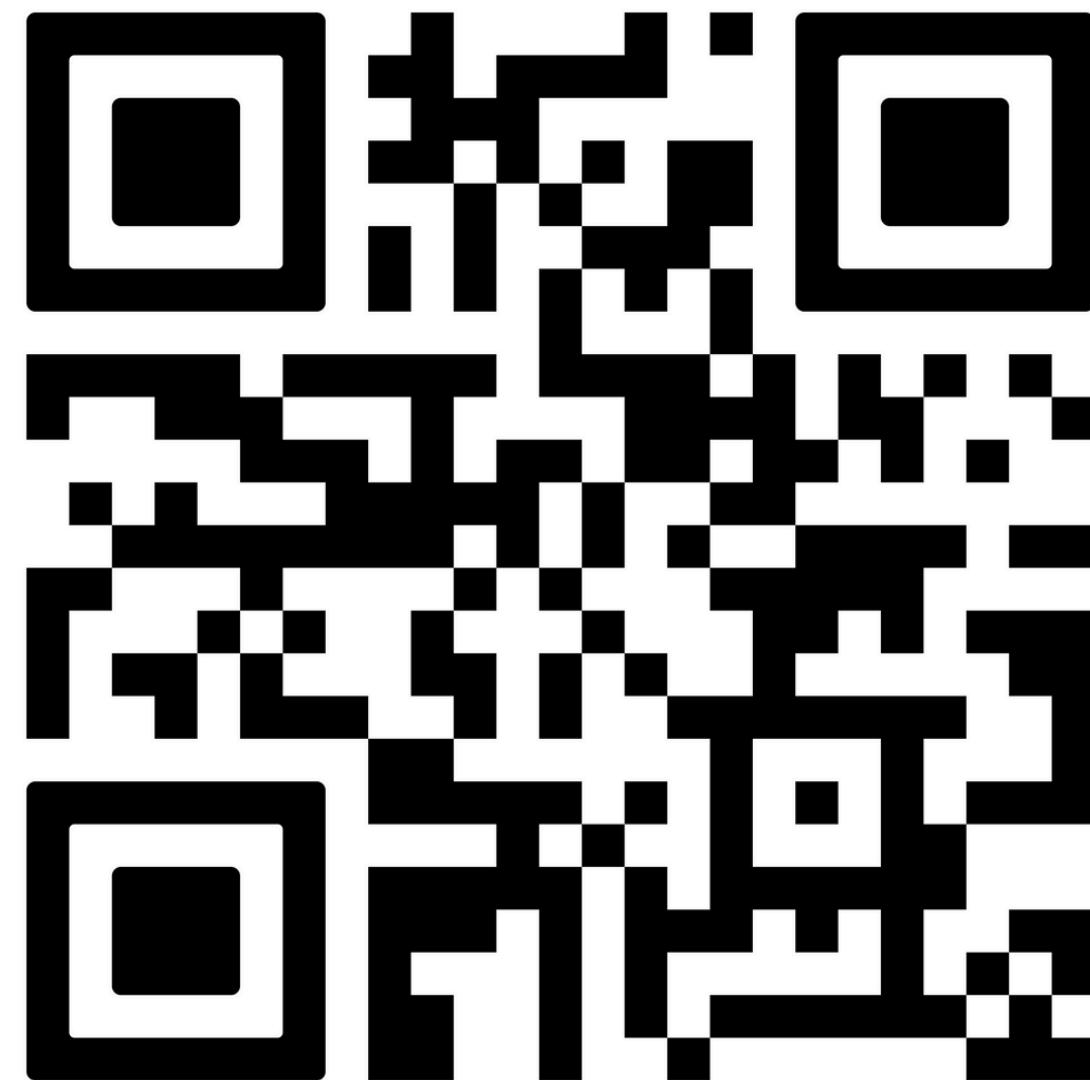


Presença

- Linktree: Presente na bio do nosso instagram
- Presença ficará disponível até 1 hora antes da próxima aula
- É necessário 70% de presença para obter o certificado



Presença



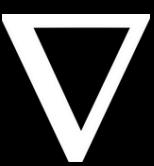
Preparação e Limpeza de Dados

Tópicos

1. Perfil dos Dados
2. Tratamento de Dados
3. Conversão e Padronização de Tipos
4. Binning
5. Modelagem



Perfilamento de Dados



Perfilamento de dados

- No mundo real, os dados raramente são perfeitos; podem ser incompletos, inconsistentes ou incorretos.
- O perfilamento é o primeiro passo crucial no processo de limpeza e tratamento de dados, permitindo entender a estrutura, o conteúdo e a qualidade da informação.

Dimensão e a Variedade dos Dados

Contagem Total de Registros: COUNT(*)

- Obtém o "tamanho" da tabela, ou seja, quantas linhas ela possui.

Contagem de Valores Distintos: COUNT(DISTINCT coluna)

- Obtém a quantidade de valores únicos numa coluna (por exemplo, de quantos estados diferentes meus clientes vêm).

```
SELECT COUNT(*) FROM clientes;
```

```
SELECT COUNT(DISTINCT estado) FROM clientes;
```

Análise de Frequência e Distribuição

Agrupamento: GROUP BY

- Agrupa linhas que têm os mesmos valores em colunas especificadas.
- Normalmente é combinada com COUNT() e outros comandos (como AVG, MAX, etc).
- Antes o COUNT retornaria um único número com o total de linhas, agora vai retornar um número para cada grupo.

```
SELECT estado, COUNT(*) AS total_clientes  
FROM clientes  
GROUP BY estado;
```

Detecção de Outliers

Comandos como: MIN(), MAX() e AVG()

- MIN() retorna o menor valor da coluna.
- MAX() retorna o maior valor da coluna.
- AVG() retorna a média dos valores da coluna.

Em conjunto, podemos verificar valores que fogem do esperado (os outliers).

```
SELECT  
    MIN(idade) AS idade_minima,  
    MAX(idade) AS idade_maxima,  
    AVG(idade) AS idade_media  
FROM clientes;
```

Encontrando Duplicatas

É possível usar o COUNT junto do GROUP BY para detectar valores duplicados. Por exemplo, se um e-mail for único por cliente, então ao agrupar por email, cada grupo deveria ter só um item. Logo, basta detectar quando COUNT() for maior que 1:

```
SELECT email, COUNT(*)
FROM clientes
GROUP BY email
HAVING COUNT(*) > 1;
```

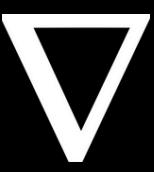
Identificação de Valores Nulos

COUNT(coluna) conta apenas valores não-nulos de uma coluna, então para quantificar os valores nulos, basta comparar o COUNT(*) (número de linhas totais) vs COUNT(coluna):

```
SELECT COUNT(*) - COUNT(telefone)
AS total_nulos_telefone
FROM clientes;
```



Tratamento de Nulos e Valores Inválidos



Valores Nulos e Inválidos

- Dependendo de como foi realizada a extração dos dados, podemos ter problemas de dados incompletos e inconsistentes (“**Garbage In, Garbage Out**”)
- Razões são inúmeras
 - Falha humana e/ou de Sistema
 - Integração de sistemas com diferentes padrões
- Gera Análises Incorretas (Cálculos, agrupamentos, etc.)

Tratamento de Nulos

- Valores nulos são valores com ausência de informação
- Podemos tratar valores nulos de 2 formas:
 - **Substituição** do NULL por outro valor
 - **Eliminação** dos registros nulos
- A depender do atributo faltante e da análise a ser feita, escolhemos entre uma opção ou outra

▽

Tratamento de Nulos: Substituição

- Comando **CASE**
 - if else

```
1   CASE
2       WHEN atributo IS NULL THEN valor
3       ELSE atributo
4   END
```

Tratamento de Nulos: Substituição

- Tabela de Pedidos

id	nome	preco_padrao	preco_desconto
1	Notebook	4500.00	4250.00
2	Mouse	150.00	NULL
3	Teclado	250.00	NULL
4	Monitor	1200.00	1100.00

Tratamento de Nulos: Substituição

- CASE na tabela Pedidos

```
1  SELECT
2      nome AS Nome,
3      CASE
4          WHEN preco_desconto IS NULL THEN preco_padrao
5          ELSE preco_desconto
6      END AS Preco_final
7  FROM
8      Produtos;
```

Tratamento de Nulos: Substituição

- CASE na tabela Pedidos

```
1  SELECT
2      nome AS Nome,
3      CASE
4          WHEN preco_desconto IS NULL THEN preco_padrao
5          ELSE preco_desconto
6      END AS Preco_final
7  FROM
8      Produtos;
```

Nome	Preco_final
Notebook	4250.00
Mouse	150.00
Teclado	250.00
Monitor	1100.00

Tratamento de Nulos: Substituição

- Comando **COALESCE**
 - Retorna o primeiro argumento não-nulo
 - Padrão ANSI SQL

1

COALESCE(Valor1, Valor2, ..., ValorN)

Tratamento de Nulos: Substituição

- COALESCE na tabela Pedidos

```
1  SELECT
2      nome,
3      COALESCE(preco_desconto, preco_padrao) AS preco_final
4  FROM
5      produtos;
```

Tratamento de Nulos: Substituição

- Comando **NULLIF**
 - Valores diferentes: retorna o primeiro
 - Valores iguais: NULL

1

NULLIF(Valor1, Valor2)

- Mais limitado do que COALESCE

Tratamento de Nulos: Eliminação

- Útil em casos em que não é possível substituir um dado nulo e este é fundamental para a análise
- Ex: sistema de venda de produtos com uma relação de Pedidos com usuários com ID faltando

```
1  SELECT *
2  FROM Pedidos
3  WHERE id_usuario IS NOT NULL;
```

Padronização de Valores Faltantes

- Campos sem informação podem ser representados por outros valores além de NULL
 - “ ”, “Desconhecido”, “Não informado”, etc.
- Para evitar problemas de análise, podemos utilizar um nome padrão para valores nulos

Padronização de Valores Faltantes

- Em uma relação de usuários que se cadastraram em um sistema, alguns não informaram seu estado civil

Id_Usuario	Nome	Idade	Estado_Civil
1	Gabriela	28	'Solteira'
2	Fernando	34	"
3	Letícia	45	'Casada'
4	Roberto	52	'N/A'
5	Sandra	21	"

Padronização de Valores Faltantes

```
1  SELECT
2      id_usuario,
3      CASE
4          WHEN estado_civil IN ('', 'N/A') THEN 'Nao Informado'
5          ELSE estado_civil
6      END
7  FROM Cadastros;
```

Valores Fora do Intervalo

- Valores fora do intervalo esperado para um atributo
- Descontos acima de 100% ou uma idade acima de 150 anos para um usuário cadastrado
- Correção com CASE

```
1 CASE
2   WHEN atributo (COMPARACAO) LIMITE1 THEN Valor1
3   WHEN atributo (COMPARACAO) LIMITE2 THEN Valor2
4   ...
5   ELSE atributo
6 END
```

Valores Fora do Intervalo

- Banco de dados com notas de alunos com valores maiores do que 10 ou menores do que 0

```
1  SELECT
2      id_aluno AS ID_Aluno,
3      CASE
4          WHEN nota > 10 THEN 10.0
5          WHEN nota < 0 THEN 0.0
6          ELSE nota
7      END AS Nota_final
8  FROM Alunos;
```

Normalização de Categorias

- Um mesmo valor de um atributo é representado de diferentes formas (agrupamentos diferentes)
- “São Paulo”, “sp”, “SP”, etc

```
1 CASE
2   WHEN cidade IN ("São Paulo", "SP", "sp", "são paulo") THEN "São Paulo"
3   ELSE cidade
4 END AS Cidade
```

Normalização de Categorias

- Um mesmo valor de um atributo é representado de diferentes formas
- “São Paulo”, “sp”, “SP”, etc

```
1 CASE
2   WHEN cidade IN ("São Paulo", "SP", "sp", "são paulo") THEN "São Paulo"
3   ELSE cidade
4 END AS Cidade
```

- Uso de LOWER(), UPPER()

```
1 CASE
2   WHEN LOWER(cidade) IN ("são paulo", "sp") THEN "São Paulo"
3   ELSE cidade
4 END AS Cidade
```



Conversão e Padronização de Tipos



Uso de CAST e CONVERT

As funções CAST() e CONVERT() são as principais ferramentas para alterar explicitamente o tipo de uma coluna.

Isso é necessário quando se precisa realizar operações matemáticas em colunas que foram importadas como strings, ou para formatar datas e horas.

CAST(expressão AS tipo):

- sintaxe CAST é padrão em SQL e geralmente é a mais recomendada por ser mais portável entre diferentes bancos de dados.

CONVERT(tipo, expressão):

Exemplo Prático

```
1 ✓ SELECT
2   '150.75' AS PrecoTexto,
3   CAST('150.75' AS DECIMAL(10, 2)) AS PrecoNumerico,
4   CAST('2023-01-15' AS DATE) AS DataConvertida
5   FROM MinhaTabela;
```

No exemplo acima, a coluna PrecoTexto é convertida para um tipo numérico que pode ser usado em cálculos. Da mesma forma, a string '2023-01-15' é convertida em um tipo de dado DATE

Normalização de Strings

A padronização de strings é essencial para evitar duplicatas causadas por diferenças de caixa (maiúsculas/minúsculas) ou espaços extras. Uma cidade como "são paulo", "São Paulo" e "SÃO PAULO" deve ser tratada como um único valor.

LOWER():

Converte a string para minúsculas.

UPPER():

Converte a string para maiúsculas.

TRIM():

Remove espaços em branco do início e do fim da string.

Exemplo Prático

```
1 ✓ SELECT
2 TRIM(UPPER(Cidade)) AS CidadePadronizada
3 FROM
4 Clientes;
```

Esse comando garante que todos os registros da cidade de São Paulo, por exemplo, serão tratados como ‘SÃO PAULO’, facilitando a contagem de registros e agrupamentos.

Conversão de Datas e Textos

Dados de data e hora, quando armazenados como texto, precisam ser convertidos para tipos nativos como DATE ou TIMESTAMP para que o PostgreSQL possa realizar cálculos e ordenações corretamente. O PostgreSQL é particularmente rigoroso com os formatos de data que aceita em conversões diretas.

O Caso Padrão: Conversão Direta com CAST

Se o texto já estiver no formato padrão ISO 8601 ('YYYY-MM-DD'), a conversão é trivial. Pode-se usar a função CAST ou, de forma mais idiomática em PostgreSQL, o operador de atalho ::.

-- Ambas as linhas funcionam perfeitamente para o formato padrão

```
SELECT CAST('2023-01-15' AS DATE);  
SELECT '2023-01-15'::DATE;
```

O Caso Realista: Usando TO_DATE para Formatos Específicos

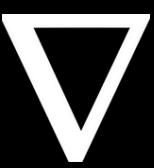
Na prática, os dados de texto raramente estão em um formato ideal. Para lidar com formatos variados e não padronizados, como 'DD/MM/YYYY' ou 'Janeiro 15, 2023', a função TO_DATE é a ferramenta correta e segura, pois permite que você especifique exatamente como a string deve ser lida.

Exemplo Prático:

Vamos calcular há quantos dias um evento ocorreu, sabendo que sua data está em uma coluna de texto no formato 'DD/MM/YYYY'

```
1 ✓ SELECT
2 DataEventoTexto,
3 TO_DATE(DataEventoTexto, 'DD/MM/YYYY') AS DataEventoFormatada,
4 (CURRENT_DATE - TO_DATE(DataEventoTexto, 'DD/MM/YYYY')) AS DiasDesdeOEvento
5 FROM Eventos;
```

Binning





Binning

- O processo de binning em SQL consiste em transformar variáveis contínuas em categorias ou faixas.
- Pode ser feito manualmente (definindo intervalos específicos) ou automaticamente (utilizando funções ou quantis).
- Essa técnica é útil para simplificar análises, criar relatórios e preparar dados para modelos preditivos

Binning Manual

Podemos criar faixas de valores
explicitamente utilizando a cláusula CASE
... WHEN

Binning Manual

ID_Cliente	Idade
1	17
2	22
3	35
4	47
5	63

```
1 v  SELECT
2          customer_id,
3  CASE
4      WHEN age < 18 THEN 'Menor de idade'
5      WHEN age BETWEEN 18 AND 29 THEN 'Jovem'
6      WHEN age BETWEEN 30 AND 59 THEN 'Adulto'
7      ELSE 'Idoso'
8  END AS faixa_etaria
9  FROM clientes;
```

Binning Manual

ID_Cliente	Faixa_Etária
1	Menor de idade
2	Jovem
3	Adulto
4	Adulto
5	Idoso

Binning Automático

Também podemos usar funções como *FLOOR*,
ROUND ou quantis via funções de janela

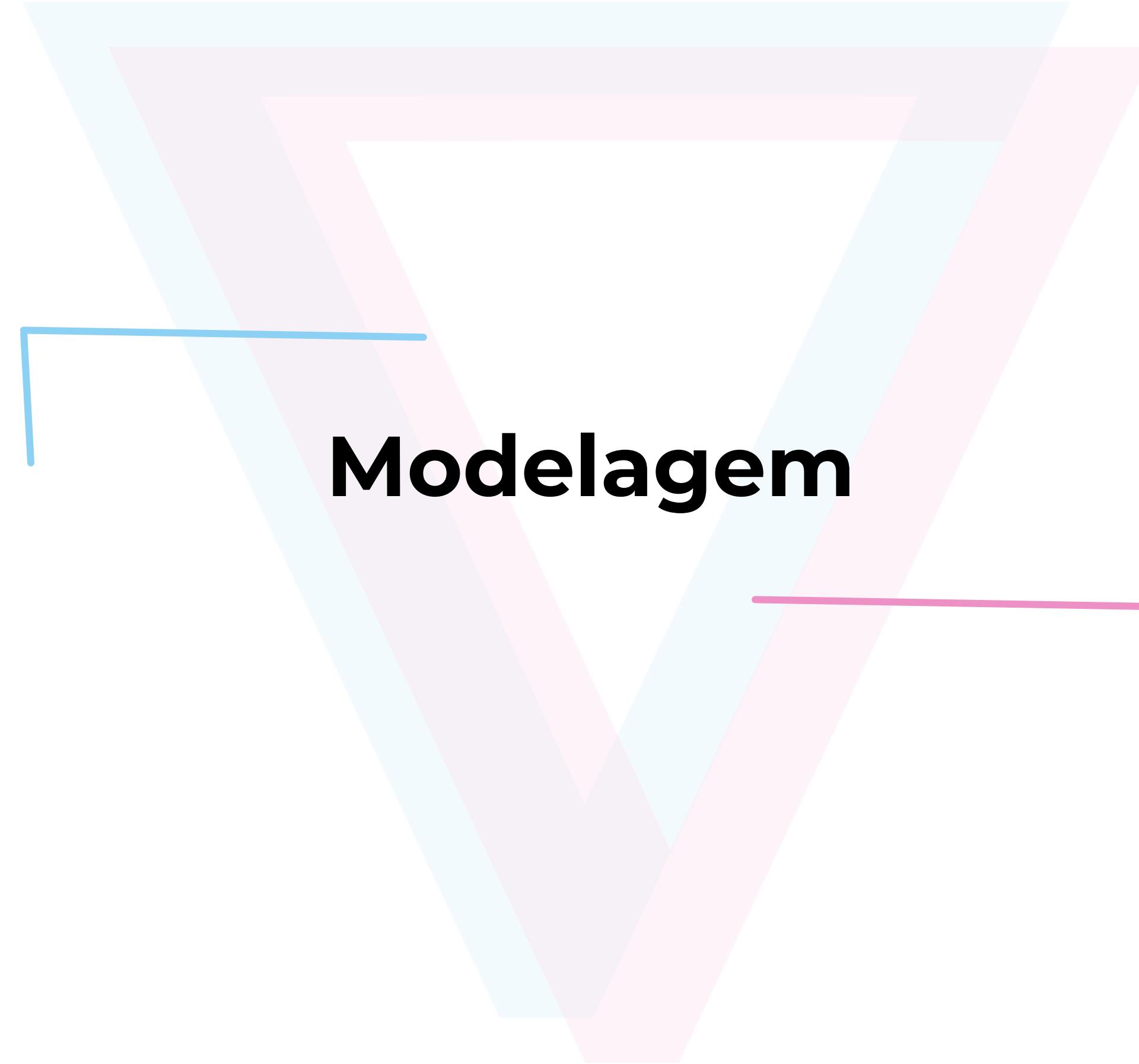
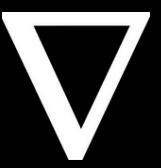
Binning Automático

ID_Produto	Preço
201	12.50
202	23.40
203	48.90
204	52.10
205	97.80

```
1 ✓ SELECT  
2     product_id,  
3     price,  
4     FLOOR(price / 20) * 20 AS faixa_preco  
5 FROM produtos;
```

Binning Automático

ID_Produto	Preço	Faixa_Preço
201	12.50	0
202	23.40	20
203	48.90	40
204	52.10	40
205	97.80	80



Modelagem



Modelagem e reformatação de dados

- Organizar informações para facilitar análise
- Envolve transformação da estrutura das tabelas
- Possibilidade de criar tabelas derivadas para análises específicas



Pivot (linhas → colunas)

- Converte registros de linhas em colunas
- Útil para visualização cruzada de medidas agregadas

Exemplo aplicado em vendas por mês

Produto	Mês	Vendas
A	Jan	100
A	Fev	120
B	Jan	200
B	Fev	180

```
SELECT  
    Produto,  
    SUM(CASE WHEN Mes = 'Jan' THEN Vendas ELSE 0 END) AS Jan,  
    SUM(CASE WHEN Mes = 'Fev' THEN Vendas ELSE 0 END) AS Fev  
FROM Vendas  
GROUP BY Produto;
```

- Resultado esperado

Produto	Jan	Fev
A	100	120
B	200	180



Unpivot (colunas → linhas)

- Processo inverso do pivot
- Converte colunas em linhas
- Útil para análises no formato “long”

Exemplo aplicado em vendas por mês

Produto	Jan	Fev
A	100	120
B	200	180

```
SELECT  
    Produto,  
    Mes,  
    Vendas  
FROM  
    (SELECT Produto, Jan, Fev FROM VendasPivot) vp  
UNPIVOT  
    (Vendas FOR Mes IN (Jan, Fev)) AS unpvt;
```

- Resultado esperado

Produto	Mês	Vendas
A	Jan	100
A	Fev	120
B	Jan	200
B	Fev	180

Tabelas Derivadas (Subqueries e CTEs)

- Criam conjuntos temporários de dados para reutilização
- Duas abordagens:
 - Subqueries na cláusula FROM: cria uma “tabela temporária” dentro de uma consulta
 - CTEs (Common Table Expressions): permitem nomear a tabela derivada e usá-la em múltiplas etapas da consulta, aumentando legibilidade.

Exemplo de SUBQUERY

```
SELECT Produto, MediaVendas  
FROM (  
    SELECT Produto, AVG(Vendas) AS MediaVendas  
    FROM Vendas  
    GROUP BY Produto  
) AS VendasMedia  
WHERE MediaVendas > 150;
```

- Resultado esperado

Produto	MediaVendas
B	190

Exemplo de CTEs

```
WITH VendasMedia AS (
    SELECT Produto, AVG(Vendas) AS MediaVendas
    FROM Vendas
    GROUP BY Produto
)
SELECT *
FROM VendasMedia
WHERE MediaVendas > 150;
```

- Resultado esperado

Produto	MediaVendas
B	190

Boas práticas

- Usar GROUP BY com funções de agregação em pivots
- Preferir CTEs em consultas complexas
- Evitar pivots muito largos: manter formato “long”
- Documentar transformações de dados (subqueries/CTEs)



data@icmc.usp.br



[@data.icmc](https://www.instagram.com/@data.icmc)



[/c/DataICMC](https://www.youtube.com/c/DataICMC)



[/icmc-data](https://github.com/icmc-data)



data.icmc.usp.br

|| obrigado!