

Joins (Unões de Tabelas) e Modelagem de Banco de Dados



Tabelas sendo unidas

Conectando dados distribuídos em múltiplas tabelas para
análise eficiente

Objetivos da Aula



Compreender o conceito e a importância dos Joins em bancos de dados relacionais e seu papel fundamental na análise de dados distribuídos em múltiplas tabelas.



Dominar os diferentes tipos de Joins (Inner, Left, Right, Full Outer) e entender quando e como aplicar cada um deles em situações específicas.



Aplicar Joins em cenários complexos como Self Join (união de uma tabela com ela mesma) e Non-Equijoin (união sem igualdade), expandindo as possibilidades de consulta.



Implementar e apresentar modelos conceituais de banco de dados, compreendendo a relação entre o design do banco e as operações de Join necessárias.



Criar recursos de segurança e procedimentos armazenados para garantir a integridade e eficiência das operações de banco de dados em ambientes de produção.



Dominar Joins é essencial para extrair valor de dados distribuídos em múltiplas tabelas

Abertura e Contexto

💡 Por que precisamos de Joins?

No mundo real, os dados raramente estão contidos em uma única tabela. Para evitar redundância e garantir a integridade dos dados, utilizamos o processo de **normalização**, que distribui as informações em múltiplas tabelas relacionadas.

Normalização de Dados

É o processo de organizar dados em um banco de dados, incluindo a criação de tabelas e estabelecimento de relações entre elas de acordo com regras projetadas para proteger os dados e tornar o banco de dados mais flexível.

Embora a normalização seja essencial para o design eficiente de bancos de dados, ela cria um desafio: **os dados relacionados ficam espalhados em várias tabelas**. Para análise e relatórios, precisamos frequentemente reunir esses dados fragmentados.

É aqui que entram os **Joins** - operações que permitem "costurar" os dados distribuídos em múltiplas tabelas, reconstruindo as relações entre eles para análise e apresentação.

Dados Normalizados vs. Dados Unidos



Joins permitem reconstruir as relações entre dados distribuídos em múltiplas tabelas

O que é um Join?

União Interna (Inner Join)

🔗 Conceito

O **Inner Join** é o tipo mais comum de join, que retorna apenas os registros que possuem **correspondência em ambas as tabelas**. É como uma interseção entre dois conjuntos, onde apenas os elementos comuns são incluídos no resultado.

Quando usar:

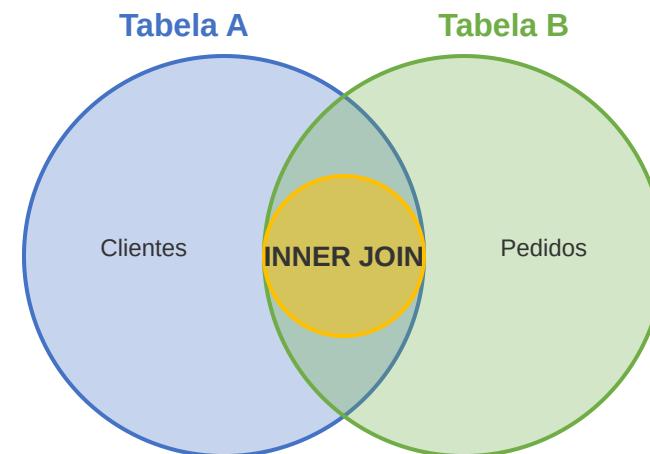
- Quando você precisa apenas dos dados que têm correspondência nas duas tabelas
- Quando deseja eliminar registros incompletos ou sem relação
- Para garantir integridade referencial nos resultados

</> Sintaxe SQL

```
-- Sintaxe básica do INNER JOIN
SELECT TabelaA.Coluna1, TabelaA.Coluna2, TabelaB.Coluna1
FROM TabelaA
INNER JOIN TabelaB
ON TabelaA.ChavePrimaria = TabelaB.ChaveEstrangeira;
```

```
-- Exemplo prático com Clientes e Pedidos
SELECT C.Nome, C.Email, P.PedidoID, P.DataPedido, P.ValorTotal
FROM Clientes C
INNER JOIN Pedidos P
ON C.ClienteID = P.ClienteID
ORDER BY P.DataPedido;
```

Diagrama de Venn



O *Inner Join* retorna apenas a interseção entre as tabelas A e B

📋 Resultado do Inner Join

Nome	Email	PedidoID	DataPedido	ValorTotal
Maria Silva	maria@email.com	101	2023-01-15	R\$ 150,00
João Santos	joao@email.com	102	2023-01-20	R\$ 85,50
Maria Silva	maria@email.com	103	2023-02-10	R\$ 220,75
Ana Oliveira	ana@email.com	104	2023-02-15	R\$ 65,00

Observe que Carlos e Lucia não aparecem no resultado pois não fizeram pedidos

Produto Cartesiano e Apelidos

⚠ Produto Cartesiano

O **Produto Cartesiano** ocorre quando você faz um Join sem especificar a condição de junção (cláusula ON). Ele combina **cada linha** da primeira tabela com **cada linha** da segunda tabela.

❗ Cuidado!

O resultado de um Produto Cartesiano pode ser catastrófico em tabelas grandes. Se a Tabela A tem 1.000 linhas e a Tabela B tem 1.000 linhas, o resultado terá 1.000.000 de linhas!

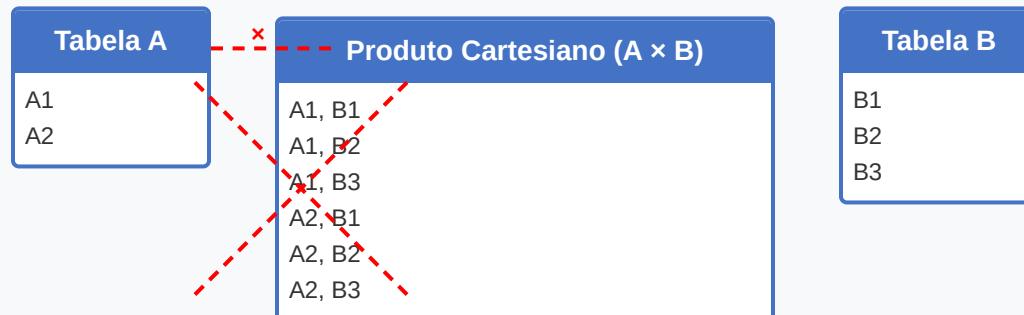
Sintaxe do Produto Cartesiano

-- Forma explícita (CROSS JOIN)

```
SELECT * FROM TabelaA CROSS JOIN TabelaB;
```

-- Forma implícita (sem condição de junção)

```
SELECT * FROM TabelaA, TabelaB;
```



🔗 Apelidos em Tabelas (Aliases)

Apelidos (ou Aliases) são nomes temporários atribuídos às tabelas ou colunas em uma consulta SQL. Eles são essenciais para:

Simplificar consultas com nomes de tabelas longos

Evitar ambiguidade quando colunas têm o mesmo nome

Tornar obrigatório em Self Joins (tabela se junta a si mesma)

Sintaxe de Apelidos

-- Forma com a palavra-chave AS

```
SELECT C.Nome, P.DataPedido  
FROM Clientes AS C  
INNER JOIN Pedidos AS P  
ON C.ClienteID = P.ClienteID;
```

C.Nome	P.PedidoID	P.DataPedido
Maria Silva	101	2023-01-15
João Santos	102	2023-01-20
Maria Silva	103	2023-02-10

Usando apelidos (C para Clientes e P para Pedidos) torna a consulta mais legível

Uniões Externas (Outer Joins)

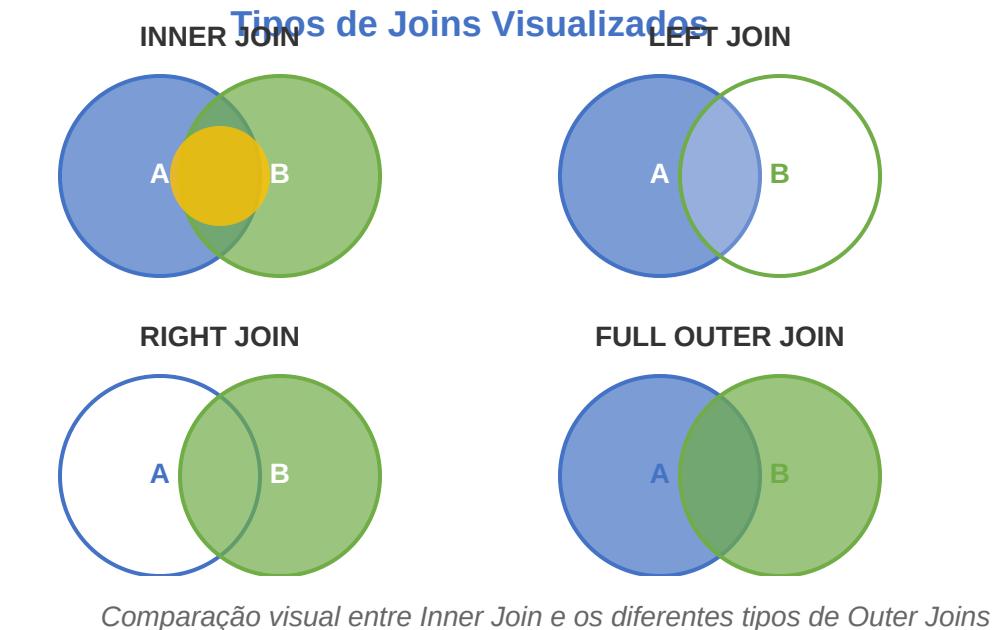
💡 Por que precisamos de Outer Joins?

O **Inner Join** é útil, mas tem uma limitação importante: ele **descarta todos os registros que não possuem correspondência** na outra tabela. Em muitos cenários de análise, precisamos manter registros mesmo quando não há correspondência.

Limitação do Inner Join

Ao usar Inner Join entre Clientes e Pedidos, clientes que nunca fizeram pedidos são completamente excluídos do resultado. Da mesma forma, pedidos sem cliente associado também são excluídos.

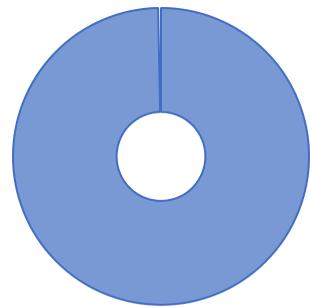
Tipo de Join	Descrição	Uso Típico
LEFT JOIN	Mantém todos os registros da tabela da esquerda	Listar todos os clientes, mesmo sem pedidos
RIGHT JOIN	Mantém todos os registros da tabela da direita	Listar todos os pedidos, mesmo sem cliente
FULL OUTER JOIN	Mantém todos os registros de ambas as tabelas	Análise completa de dados



Left Join e Right Join

→ Left Join

Left Join retorna todos os registros da tabela à esquerda (primeira tabela) e os registros correspondentes da tabela à direita. Se não houver correspondência, os campos da tabela à direita serão preenchidos com **NULL**.



■ Tabela A (todos) ■ Tabela B (apenas correspondências)

-- Sintaxe do **LEFT JOIN**

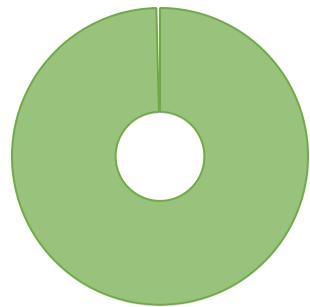
```
SELECT C.Nome, C.Email, P.PedidoID  
FROM Clientes C  
LEFT JOIN Pedidos P ON C.ClienteID = P.ClienteID;
```

Nome	Email	PedidoID
Ana Oliveira	ana@email.com	104
Carlos Pereira	carlos@email.com	NULL
João Santos	joao@email.com	102

Observe que Carlos aparece no resultado mesmo sem pedidos

← Right Join

Right Join retorna todos os registros da tabela à direita (segunda tabela) e os registros correspondentes da tabela à esquerda. Se não houver correspondência, os campos da tabela à esquerda serão preenchidos com **NULL**.



■ Tabela A (apenas correspondências) ■ Tabela B (todos)

-- Sintaxe do **RIGHT JOIN**

```
SELECT C.Nome, P.PedidoID, P.ValorTotal  
FROM Clientes C  
RIGHT JOIN Pedidos P ON C.ClienteID = P.ClienteID;
```

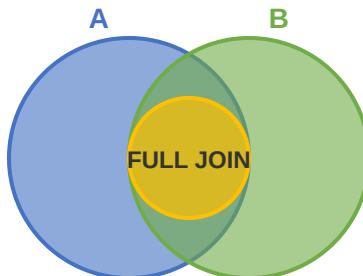
Nome	PedidoID	ValorTotal
Maria Silva	101	150.00
João Santos	102	85.50
NULL	106	45.75

Observe que o pedido 106 aparece mesmo sem cliente associado

Full Outer Join e Joins Múltiplos

Full Outer Join

Full Outer Join retorna todos os registros de ambas as tabelas. Quando não há correspondência, os campos da tabela sem correspondência são preenchidos com **NULL**.



-- Sintaxe do FULL OUTER JOIN (PostgreSQL)

```
SELECT C.Nome, P.PedidoID  
FROM Clientes C  
FULL OUTER JOIN Pedidos P ON C.ClienteID = P.ClienteID;
```

Nome	PedidoID
Maria Silva	101
Carlos Pereira	NULL
NULL	106

Full Outer Join mostra todos os clientes e todos os pedidos

Joins Múltiplos

É possível unir mais de duas tabelas em uma única consulta, criando uma nova cláusula JOIN e ON para cada tabela adicional. A ordem é sequencial.



Resultado do Join Múltiplo

```
Cliente: Maria Silva  
Pedido: 101 (15/01/23)
```

-- Exemplo de múltiplos joins

```
SELECT C.Nome AS Cliente,  
P.PedidoID,  
PR.Nome AS Produto  
FROM Clientes C  
INNER JOIN Pedidos P ON C.ClienteID = P.ClienteID  
INNER JOIN ItensPedido IP ON P.PedidoID = IP.PedidoID  
INNER JOIN Produtos PR ON IP.ProdutoID = PR.ProdutoID;
```

Regra importante: Para cada tabela adicional, adicione uma nova cláusula JOIN e especifique a condição de junção com ON. A ordem dos joins afeta o resultado.

Tópicos Avançados - Non-Equijoin e Self Join

≠ Non-Equijoin (União sem Igualdade)

Non-Equijoin é um tipo de join que não usa o operador de igualdade (=) na condição de junção. Em vez disso, utiliza outros operadores de comparação como >, <, BETWEEN, IN etc.

Exemplo: Faixas Salariais

Unir funcionários com suas faixas salariais, onde o salário do funcionário está entre os valores mínimo e máximo da faixa.

-- Non-Equijoin usando BETWEEN

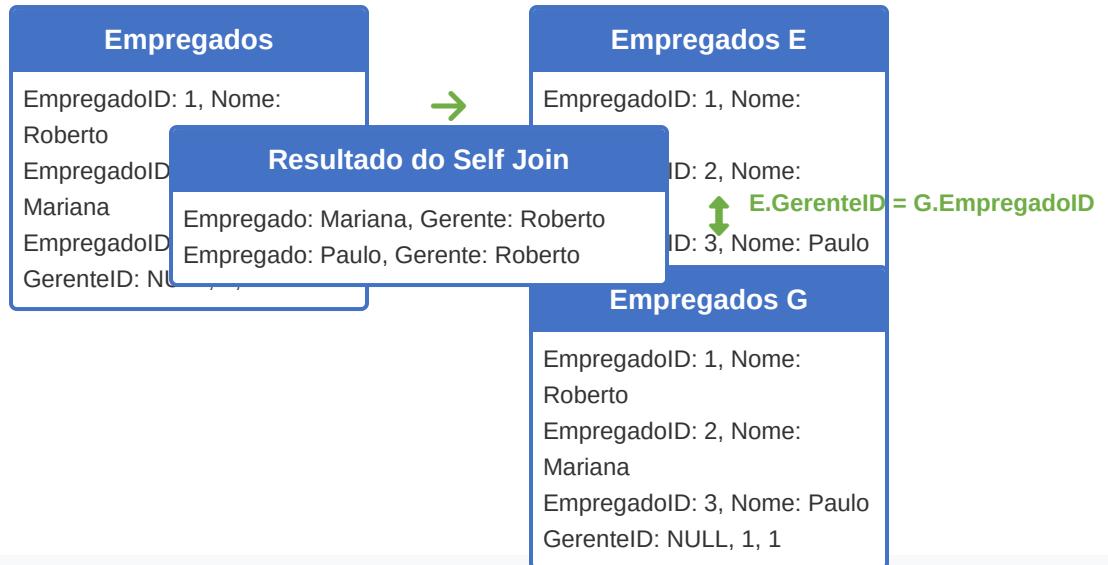
```
SELECT F.Nome, F.Salario, FS.Classificacao
FROM Funcionarios F
JOIN FaixasSalario FS
ON F.Salario BETWEEN FS.FaixaMin AND FS.FaixaMax
ORDER BY F.Salario;
```

Nome	Salario	Classificacao
Elisa Ferreira	1.800,00	Iniciante
Ana Silva	3.500,00	Júnior
Bruno Santos	5.200,00	Pleno
Carla Oliveira	7.800,00	Sênior

Quando usar: Non-Equijoins são úteis para classificação, categorização, faixas de valores, e qualquer situação onde a relação entre tabelas não é uma simples igualdade.

⟳ Self Join (União com a Própria Tabela)

Self Join ocorre quando uma tabela é unida a si mesma. Isso é necessário quando uma tabela tem uma chave estrangeira que referencia sua própria chave primária, criando relações hierárquicas ou recursivas.



-- Self Join para mostrar empregados e seus gerentes

```
SELECT E.Nome AS Empregado,
       G.Nome AS Gerente
  FROM Empregados E
 LEFT JOIN Empregados G
    ON E.GerenteID = G.EmpregadoID
 ORDER BY G.Nome, E.Nome;
```

Casos de Uso Comuns para Self Join

- Estruturas hierárquicas (gerentes e subordinados)
- Árvores genealógicas (pais e filhos)
- Categorias e subcategorias
- Rotas de viagem (origem e destino)
- Comparação de registros na mesma tabela

Importante: Em Self Joins, o uso de apelidos (aliases) é **obrigatório** para diferenciar as duas "cópias" da mesma tabela na consulta.

Atividade Prática - Diferentes Tipos de Joins

Modelo de Dados

Tabela: Clientes

ClienteID	Nome	Email
1	Maria Silva	maria@email.com
2	João Santos	joao@email.com
3	Ana Oliveira	ana@email.com
4	Carlos Pereira	carlos@email.com

Tabela: Pedidos

PedidoID	ClienteID	DataPedido	ValorTotal
101	1	2023-01-15	150.00
102	2	2023-01-20	85.50
103	1	2023-02-10	220.75
106	NULL	2023-03-10	45.75

Passo a Passo

- 1 Analise as tabelas e identifique as colunas em comum para o JOIN.
- 2 Determine o tipo de JOIN necessário com base no requisito.
- 3 Escreva a consulta SQL com SELECT, FROM, JOIN e ON.
- 4 Teste e ajuste adicionando filtros, ordenação ou agrupamento.

Tarefas

1 Inner Join

Escreva uma consulta SQL que mostre todos os pedidos e os clientes que os fizeram.

```
-- Solução Tarefa 1: Inner Join
SELECT C.Nome, P.PedidoID, P.DataPedido
FROM Clientes C
INNER JOIN Pedidos P ON C.ClienteID = P.ClienteID
ORDER BY P.DataPedido;
```

2 Left Join

Escreva uma consulta SQL que mostre todos os clientes, incluindo aqueles que nunca fizeram pedidos.

```
-- Solução Tarefa 2: Left Join
SELECT C.Nome, COUNT(P.PedidoID) AS TotalPedidos
FROM Clientes C
LEFT JOIN Pedidos P ON C.ClienteID = P.ClienteID
GROUP BY C.ClienteID, C.Nome;
```

3 Right Join

Escreva uma consulta SQL que mostre todos os pedidos, incluindo aqueles que não têm cliente associado.

```
-- Solução Tarefa 3: Right Join
SELECT P.PedidoID, P.DataPedido, C.Nome AS NomeCliente
FROM Clientes C
RIGHT JOIN Pedidos P ON C.ClienteID = P.ClienteID
ORDER BY P.DataPedido;
```

Atividade de Entrega Final e Apresentação

Requisitos do Projeto Final

Modelo Conceitual

- ✓ Diagrama de Entidade-Relacionamento (ER) completo do banco de dados
- ✓ Entidades com atributos e tipos de dados claramente definidos
- ✓ Relacionamentos com cardinalidade (1:1, 1:N, N:N) especificados
- ✓ Chaves primárias e estrangeiras identificadas

Recursos de Segurança

-- Criação de usuários com permissões específicas

```
CREATE USER 'analista_dados'@'localhost' IDENTIFIED BY  
'senha_segura123';
```

-- Atribuição de permissões granulares

```
GRANT SELECT ON empresa.* TO 'analista_dados'@'localhost';  
GRANT SELECT, INSERT, UPDATE ON empresa.relatorios TO  
'analista_dados'@'localhost';
```

-- Revogação de permissões sensíveis

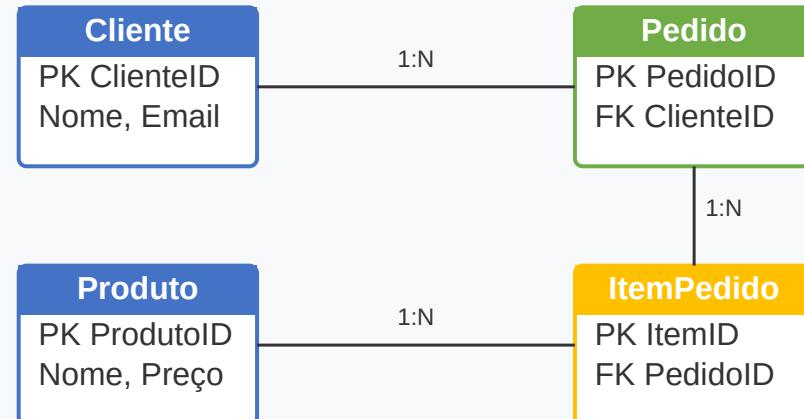
```
REVOKE DROP, TRUNCATE ON empresa.* FROM  
'analista_dados'@'localhost';
```

Procedimentos Armazenados

-- Exemplo de Stored Procedure

```
DELIMITER //  
CREATE PROCEDURE InserirNovoCliente(  
    IN p_Nome VARCHAR(100),  
    IN p_Email VARCHAR(100),  
    OUT p_ClienteID INT  
)  
BEGIN  
    -- Validação de dados  
    IF p_Email NOT LIKE '%@%.%' THEN  
        SIGNAL SQLSTATE '45000'  
        SET MESSAGE_TEXT = 'Formato de email inválido';  
    END IF;  
    -- Inserção e retorno do ID  
    INSERT INTO Clientes (Nome, Email)  
    VALUES (p_Nome, p_Email);  
    SET p_ClienteID = LAST_INSERT_ID();  
END //  
DELIMITER ;
```

Instruções para Apresentação



Exemplo de diagrama ER simplificado para o projeto final

Roteiro de Apresentação

- 1 **Introdução do Projeto:** Apresente o contexto e os objetivos do banco de dados desenvolvido, explicando o problema que ele resolve.
- 2 **Modelo Conceitual:** Explique o diagrama ER, destacando as entidades principais, seus atributos e os relacionamentos entre elas.
- 3 **Implementação de Joins:** Demonstre exemplos de consultas com diferentes tipos de joins (Inner, Left, Right, Full Outer), explicando quando cada um é mais apropriado.
- 4 **Recursos Avançados:** Apresente os joins complexos implementados (Self Join, Non-Equijoin), os recursos de segurança e os procedimentos armazenados.
- 5 **Demonstração Prática:** Execute algumas consultas em tempo real para mostrar o funcionamento do banco de dados e os resultados obtidos.

Critérios de Avaliação

- ✓ Completude do modelo conceitual e adequação à proposta do projeto
- ✓ Implementação correta dos diferentes tipos de joins e sua aplicação adequada
- ✓ Qualidade dos recursos de segurança e dos procedimentos armazenados
- ✓ Clareza e organização da apresentação e capacidade de responder a perguntas

Encerramento e Próximos Passos

✓ Conceitos-Chave Abordados

- ⌚ **Joins como ponte entre tabelas normalizadas**, permitindo a reconstrução de relações entre dados distribuídos para análise eficiente.
- 🔗 **Inner Join** para obter apenas registros com correspondência em ambas as tabelas, ideal para garantir integridade referencial.
- ☒ **Outer Joins** (Left, Right e Full) para manter registros sem correspondência, essenciais para análises completas e identificação de anomalias.
- ☛ **Joins múltiplos** para conectar três ou mais tabelas em uma única consulta, permitindo análises complexas e relatórios abrangentes.
- 🕒 **Joins avançados** como Self Join e Non-Equijoin para resolver problemas específicos como hierarquias e faixas de valores.

📘 Recursos Adicionais

Para aprofundamento

- 🌐 **W3Schools SQL Tutorial** - Tutoriais interativos sobre SQL e Joins com exemplos práticos.
- 📚 **SQL Cookbook** (O'Reilly) - Receitas práticas para resolver problemas comuns com SQL, incluindo técnicas avançadas de Join.
- 💻 **SQLZoo** - Ambiente interativo para praticar diferentes tipos de Joins com feedback imediato.

"Joins são para bancos de dados o que pontes são para cidades separadas por rios: essenciais para conectar informações que, de outra forma, permaneciam isoladas e com valor limitado."

— Princípios de Design de Bancos de Dados Relacionais

↗ Impacto dos Joins na Análise de Dados

Comparação da complexidade de consultas com e sem o uso de Joins

↗ Próximos Passos

- 1 **Otimização de consultas com Joins** - Aprender a usar índices, analisar planos de execução e melhorar o desempenho de consultas complexas.
- 2 **Common Table Expressions (CTEs)** - Explorar consultas recursivas e temporárias para resolver problemas hierárquicos complexos.
- 3 **Window Functions** - Avançar para funções analíticas que operam em conjuntos de linhas relacionadas a linha atual.
- 4 **Implementação do projeto final** - Aplicar os conceitos de Joins no modelo conceitual desenvolvido, criando consultas complexas para análise de dados.

Ferramentas recomendadas

- ✳️ **DBeaver** - Cliente SQL universal com suporte a diversos SGBDs e ferramentas visuais para design de consultas.
- ✳️ **dbdiagram.io** - Ferramenta online para criar diagramas ER e visualizar relações entre tabelas.