

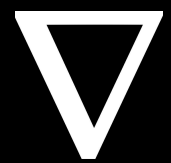


PostgreSQL



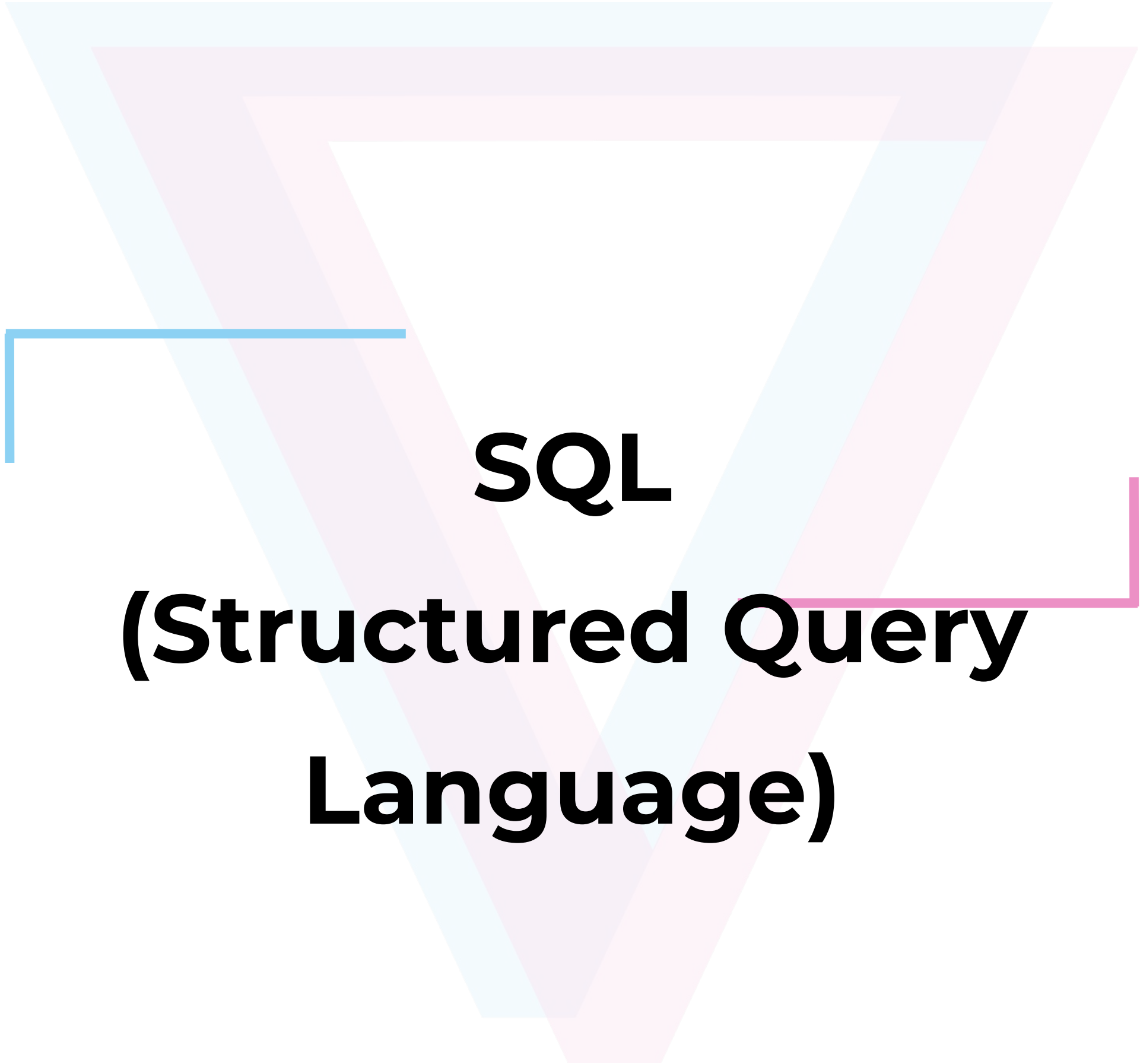
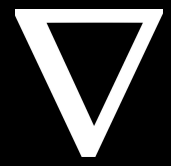
Presença

- 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



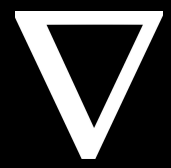
Presença



The background features a large, light blue downward-pointing triangle and a slightly offset, semi-transparent pink downward-pointing triangle. A blue line starts from the left edge, extends horizontally, and then turns 90 degrees downward. A pink line starts from the right edge, extends horizontally, and then turns 90 degrees downward. The text is centered between these two lines.

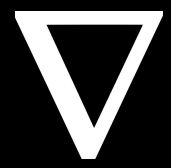
SQL

(Structured Query Language)



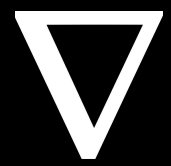
Aula 2 - Aprofundamento em SQL

- Revisão SQL Básico (Aula 1)
 - SELECT, FROM, WHERE, LIMIT
- GROUP BY
- HAVING
- ORDER BY
- JOIN
- SUBQUERY



Revisão SQL Básico

- Comandos principais: SELECT, FROM, WHERE, LIMIT
 - SELECT: colunas a visualizar
 - FROM: tabela de origem
 - WHERE: filtra registros
 - LIMIT: restringe número de linhas



Exemplo SELECT e FROM

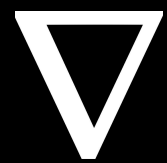
- Sintaxe básica:

```
SELECT coluna1, coluna2, ...  
FROM nome_da_tabela;
```

```
SELECT Nome, Nota  
FROM Alunos;
```

Tabela ALUNOS

ID	Nome	Nota
1	Ana Costa	85
2	Bruno Lima	92
3	Carla Souza	78



Exemplo WHERE

- Sintaxe:

```
SELECT coluna1, coluna2  
FROM tabela  
WHERE condição;
```

```
SELECT Nome, Nota  
FROM Alunos  
WHERE Nota > 80;
```

Nome	Nota
Ana Costa	85
Bruno Lima	92



Exemplo LIMIT

- Restringe número de linhas

```
SELECT Nome, Nota  
FROM Alunos  
LIMIT 2;
```

Nome	Nota
Ana Costa	85
Bruno Lima	92

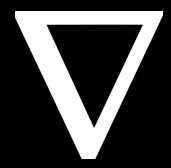


Exercícios de Fixação

1. Selecione apenas os nomes de todos os alunos.
2. Liste os alunos que tiraram nota menor que 80.
3. Retorne apenas um aluno qualquer (use LIMIT).
4. Mostre apenas os IDs e as notas de todos os alunos.

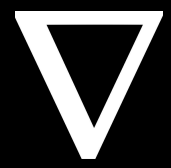
Dica:

Sempre pense em SELECT como a pergunta "O que quero ver?" e em FROM como "De onde vou buscar?". Depois, utilize WHERE para "filtrar o que não quero" e LIMIT para "reduzir a quantidade de linhas".



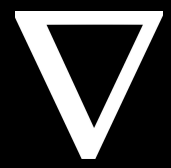
GROUP BY

- Agrupa registros com valores iguais
- Usado com funções de agregação
- COUNT, SUM, AVG, MIN, MAX



GROUP BY

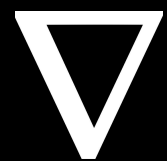
- **COUNT:** conta o número de registros em cada grupo.
- **SUM:** soma os valores numéricos de cada grupo.
- **AVG:** calcula a média dos valores de cada grupo.
- **MIN:** retorna o menor valor em cada grupo.
- **MAX:** retorna o maior valor em cada grupo.



GROUP BY

Tabela de PEDIDOS

ID_Pedido	Cliente	Categoria	Valor
101	Alice	Eletrônicos	3500.00
102	Bob	Papelaria	150.00
103	Alice	Eletrônicos	1200.00
104	Carla	Papelaria	75.00
105	Bob	Eletrônicos	2000.00
106	Alice	Papelaria	300.00



GROUP BY

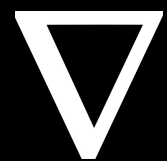
Exemplo 1: Total de pedidos por cliente.

```
SELECT Cliente, COUNT(*) AS Total_Pedidos  
FROM Pedidos  
GROUP BY Cliente;
```

Cliente	Total <i>pedidos</i>
Alice	3
Bob	2
Carla	1

Tabela de PEDIDOS

ID_Pedido	Cliente	Categoria	Valor
101	Alice	Eletrônicos	3500.00
102	Bob	Papelaria	150.00
103	Alice	Eletrônicos	1200.00
104	Carla	Papelaria	75.00
105	Bob	Eletrônicos	2000.00
106	Alice	Papelaria	300.00



GROUP BY

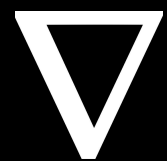
Exemplo 2: Valor total gasto em cada categoria.

```
SELECT Categoria, SUM(Valor) AS Total_Gasto  
FROM Pedidos  
GROUP BY Categoria;
```

Categoria	Total _{Gasto}
Eletrônicos	6700.00
Papelaria	525.00

Tabela de PEDIDOS

ID_Pedido	Cliente	Categoria	Valor
101	Alice	Eletrônicos	3500.00
102	Bob	Papelaria	150.00
103	Alice	Eletrônicos	1200.00
104	Carla	Papelaria	75.00
105	Bob	Eletrônicos	2000.00
106	Alice	Papelaria	300.00



GROUP BY

Exemplo 3: Maior e menor valor de pedido por cliente.

```
SELECT Cliente, MIN(Valor) AS Menor_Pedido, MAX(Valor) AS Maior_Pedido  
FROM Pedidos  
GROUP BY Cliente;
```

Tabela de PEDIDOS

Cliente	Menor <i>pedido</i>	Maior <i>pedido</i>
Alice	300.00	3500.00
Bob	150.00	2000.00
Carla	75.00	75.00

ID_Pedido	Cliente	Categoria	Valor
101	Alice	Eletrônicos	3500.00
102	Bob	Papelaria	150.00
103	Alice	Eletrônicos	1200.00
104	Carla	Papelaria	75.00
105	Bob	Eletrônicos	2000.00
106	Alice	Papelaria	300.00

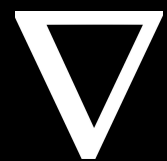


Exercícios de Fixação

1. Conte quantos pedidos cada cliente realizou.
2. Calcule o valor médio dos pedidos em cada categoria.
3. Mostre o cliente que possui o maior valor total de pedidos somados.
4. Liste, para cada categoria, o menor e o maior pedido registrado.

Dica:

Sempre que utilizar funções de agregação (COUNT, SUM, AVG, etc.), lembre-se de combinar com GROUP BY para obter resultados organizados por grupo.



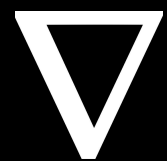
HAVING

- WHERE filtra antes do agrupamento
- HAVING filtra após o agrupamento

Tabela de NOTAS

ID_Aluno	Disciplina	Nota
1	Matemática	85
1	História	90
2	Matemática	70
2	História	75
3	Matemática	95
3	História	88

- ❑ Se quisermos selecionar apenas os alunos que têm nota maior que 80, usamos WHERE.
- ❑ Se quisermos selecionar apenas as turmas cuja média das notas é maior que 80, precisamos de HAVING.



HAVING

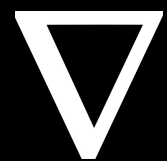
Exemplo 1: Média de notas por disciplina.

```
SELECT Disciplina, AVG(Nota) AS Media  
FROM Notas  
GROUP BY Disciplina;
```

Disciplina	Média
Matemática	83.3
História	84.3

Tabela de NOTAS

ID_Aluno	Disciplina	Nota
1	Matemática	85
1	História	90
2	Matemática	70
2	História	75
3	Matemática	95
3	História	88



HAVING

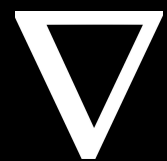
Exemplo 2: Selecionar apenas disciplinas cuja média de notas seja maior que 84.

```
SELECT Disciplina, AVG(Nota) AS Media  
FROM Notas  
GROUP BY Disciplina  
HAVING AVG(Nota) > 84;
```

Disciplina	Média
História	84.3

Tabela de NOTAS

ID_Aluno	Disciplina	Nota
1	Matemática	85
1	História	90
2	Matemática	70
2	História	75
3	Matemática	95
3	História	88



HAVING

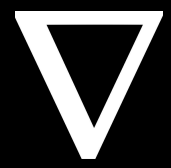
Exemplo 3: Contar quantos alunos existem em cada disciplina e mostrar apenas disciplinas com mais de 2 registros.

```
SELECT Disciplina, COUNT(*) AS Total_Alunos  
FROM Notas  
GROUP BY Disciplina  
HAVING COUNT(*) > 2;
```

Disciplina	Total _{Alunos}
Matemática	3
História	3

Tabela de NOTAS

ID_Aluno	Disciplina	Nota
1	Matemática	85
1	História	90
2	Matemática	70
2	História	75
3	Matemática	95
3	História	88



Exercícios de Fixação

1. Liste a nota média de cada disciplina.
2. Mostre apenas as disciplinas cuja média seja maior que 85.
3. Conte quantos registros cada disciplina possui e filtre apenas disciplinas com pelo menos 3 alunos.
4. Qual disciplina tem a menor nota máxima registrada?

Dica:

Lembre-se:

- Use WHERE para filtrar linhas individuais.
- Use HAVING para filtrar grupos agregados.



ORDER BY

- Ordena resultados (ASC ou DESC)
- Pode usar múltiplas colunas

- Sintaxe:

```
SELECT coluna1, coluna2, ...
```

```
FROM tabela
```

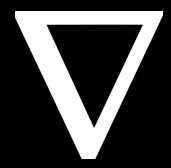
```
ORDER BY coluna1 [ASC|DESC], coluna2 [ASC|DESC];
```



ORDER BY

- Considere a seguinte tabela:

ID	Nome	Nota
1	Ana Costa	85
2	Bruno Lima	92
3	Carla Souza	78
4	Diego Alves	92
5	Fernanda Silva	70

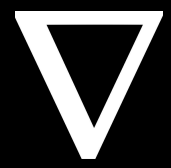


ORDER BY

Exemplo 1: Ordenar os alunos pelo nome em ordem alfabética.

```
SELECT id, nome, nota  
FROM alunos  
ORDER BY nome ASC;
```

ID	Nome	Nota
1	Ana Costa	85
2	Bruno Lima	92
3	Carla Souza	78
4	Diego Alves	92
5	Fernanda Silva	70



ORDER BY

Exemplo 2: Ordenar os alunos da maior para a menor nota.

```
SELECT id, nome, nota  
FROM alunos  
ORDER BY nota DESC;
```

ID	Nome	Nota
2	Bruno Lima	92
4	Diego Alves	92
1	Ana Costa	85
3	Carla Souza	78
5	Fernanda Silva	70



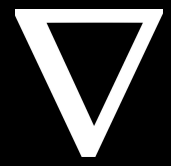
ORDER BY

Exemplo 3: Ordenar usando múltiplas colunas.

Primeiro pela nota (decrescente) e, em caso de empate, pelo nome (crescente).

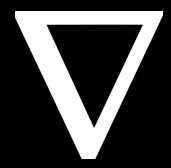
```
SELECT id, nome, nota  
FROM alunos  
ORDER BY nota DESC, nome ASC;
```

ID	Nome	Nota
2	Bruno Lima	92
4	Diego Alves	92
1	Ana Costa	85
3	Carla Souza	78
5	Fernanda Silva	70



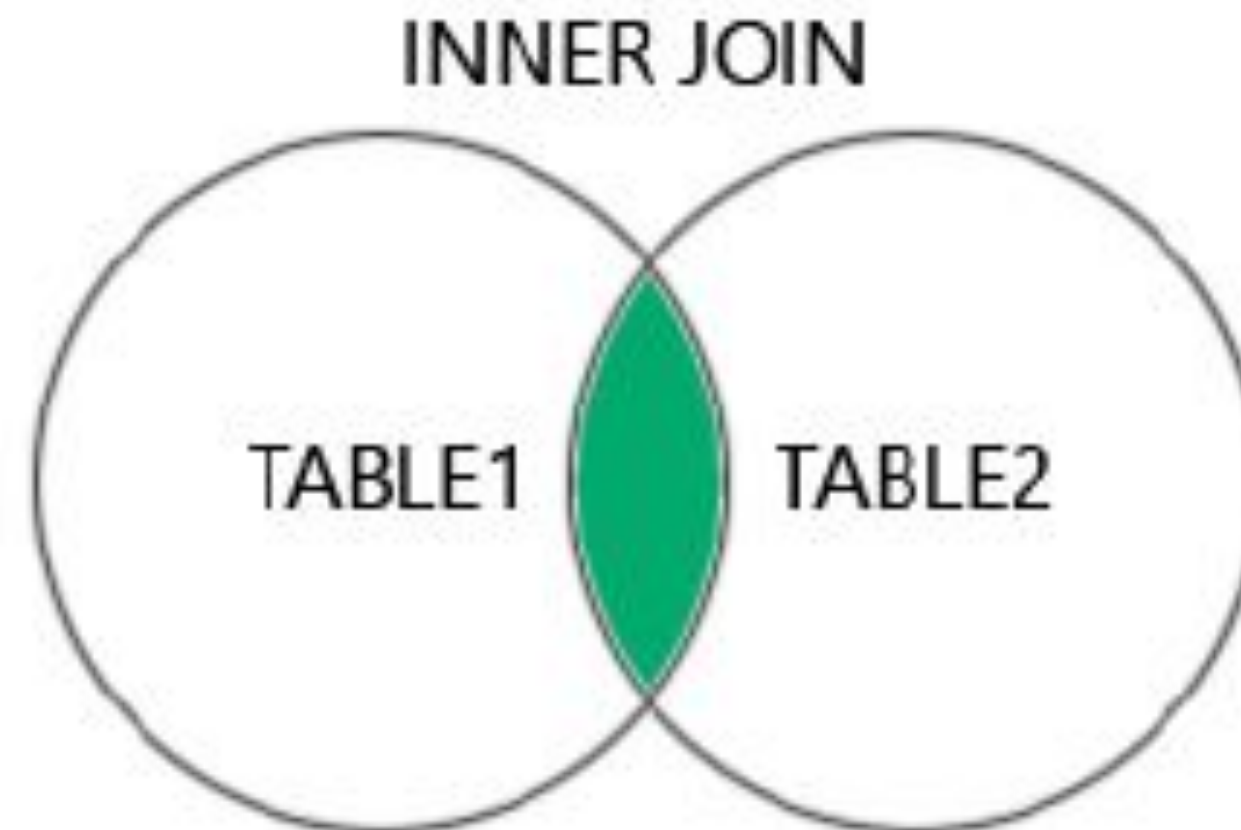
JOIN

- Combina dados de diferentes tabelas
- Tipos: INNER JOIN, LEFT JOIN, RIGHT JOIN
- Útil para relacionar clientes, pedidos, produtos



JOIN

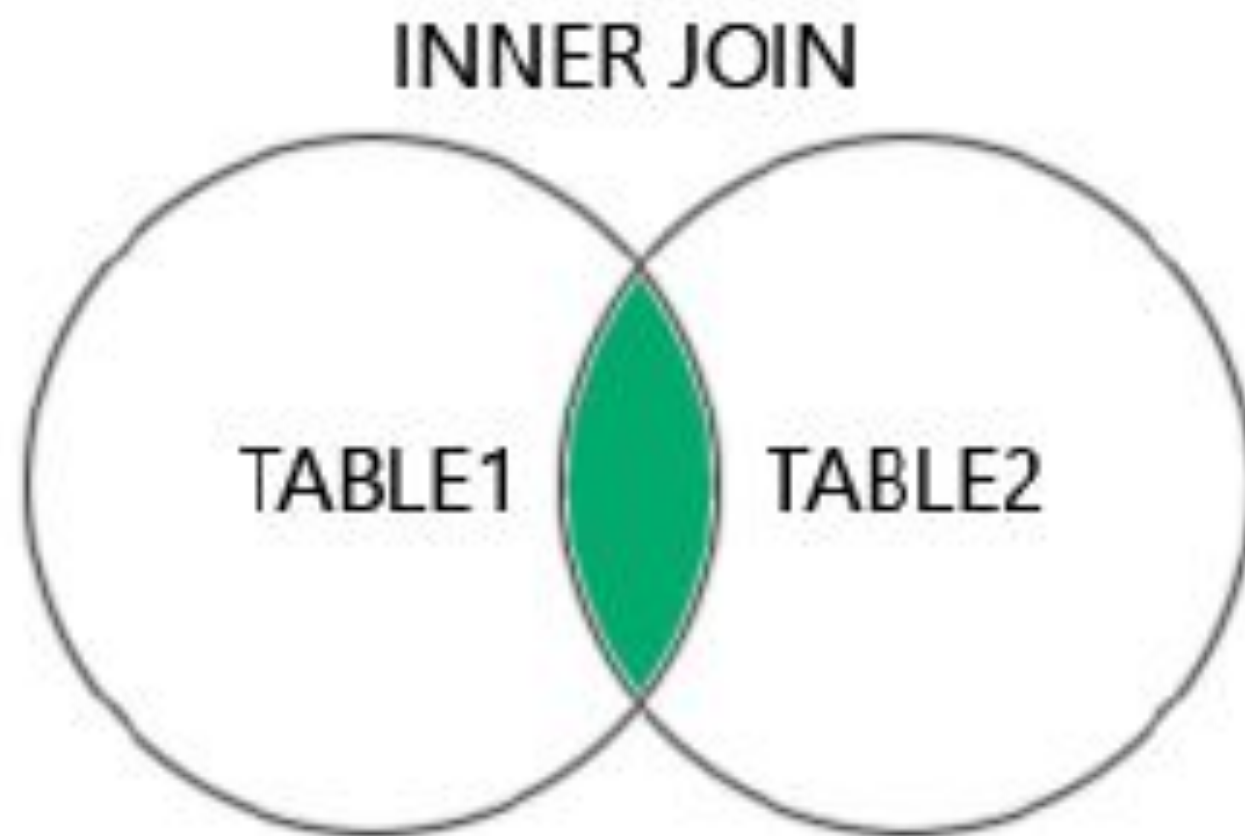
- **Inner Join (Junção Interna):** Retorna apenas os registros que possuem uma correspondência em ambas as tabelas. O INNER JOIN corresponde aos registros da intersecção dos conjuntos, como na figura abaixo.





JOIN

- **Inner Join (Junção Interna):** Retorna apenas os registros que possuem uma correspondência em ambas as tabelas. O INNER JOIN corresponde aos registros da intersecção dos conjuntos, como na figura abaixo.

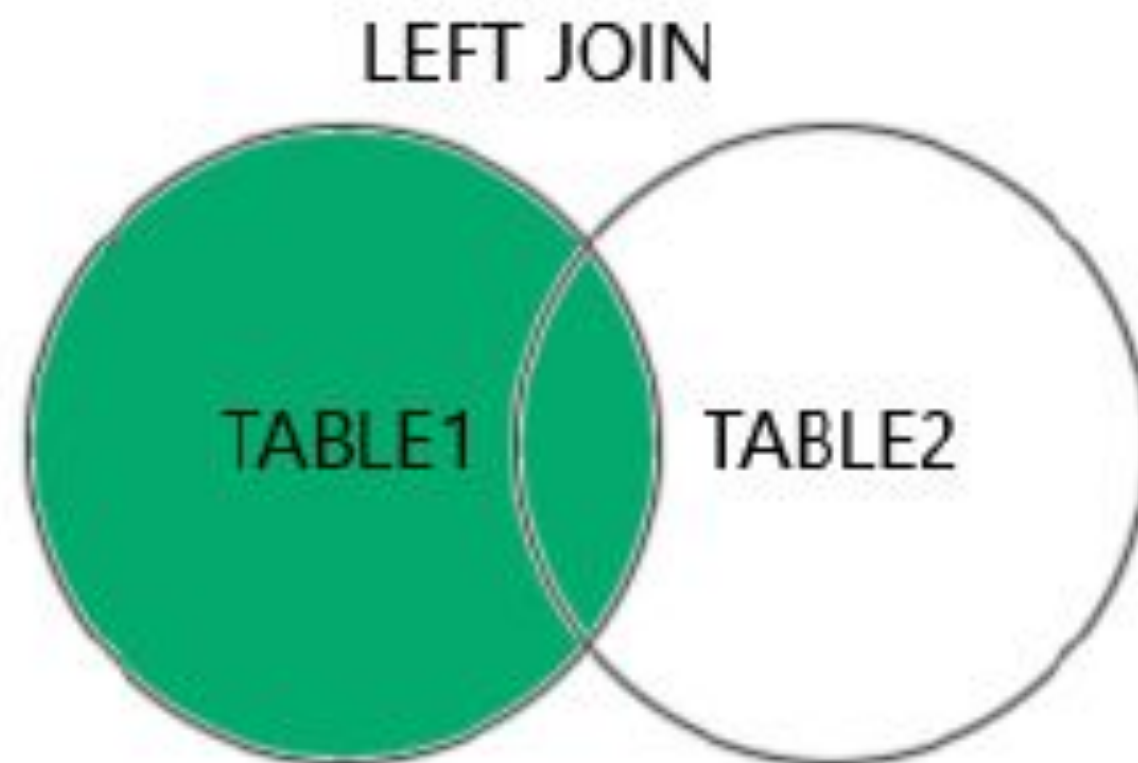


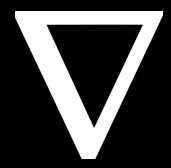
```
SELECT <Lista de Atributos>  
FROM Tabela1  
INNER JOIN Tabela2 ON Tabela1.Atributo = Tabela2.Atributo;
```



JOIN

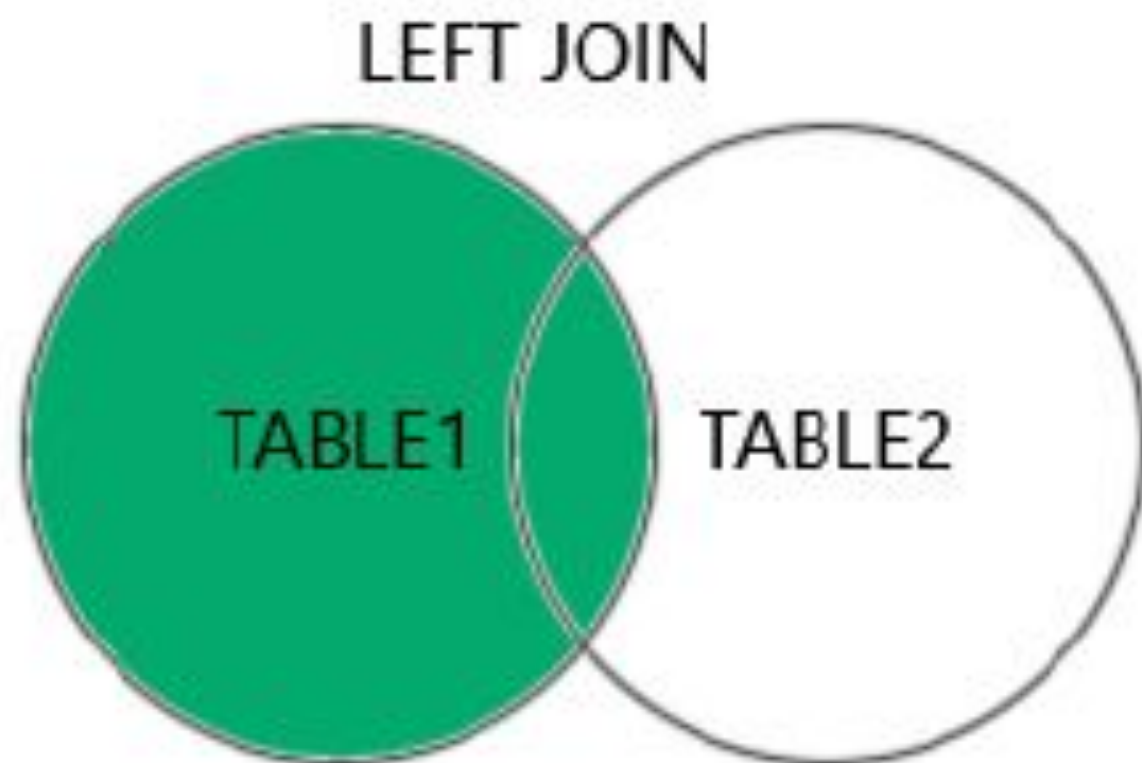
- **Left Join:** Retorna todos os registros da tabela à esquerda (tabela 1) e os registros correspondentes da tabela à direita. Se não houver correspondência na tabela da direita para um registro da esquerda, os campos da direita virão como NULL.



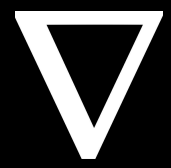


JOIN

- **Left Join:** Retorna todos os registros da tabela à esquerda (tabela 1) e os registros correspondentes da tabela à direita. Se não houver correspondência na tabela da direita para um registro da esquerda, os campos da direita virão como NULL.

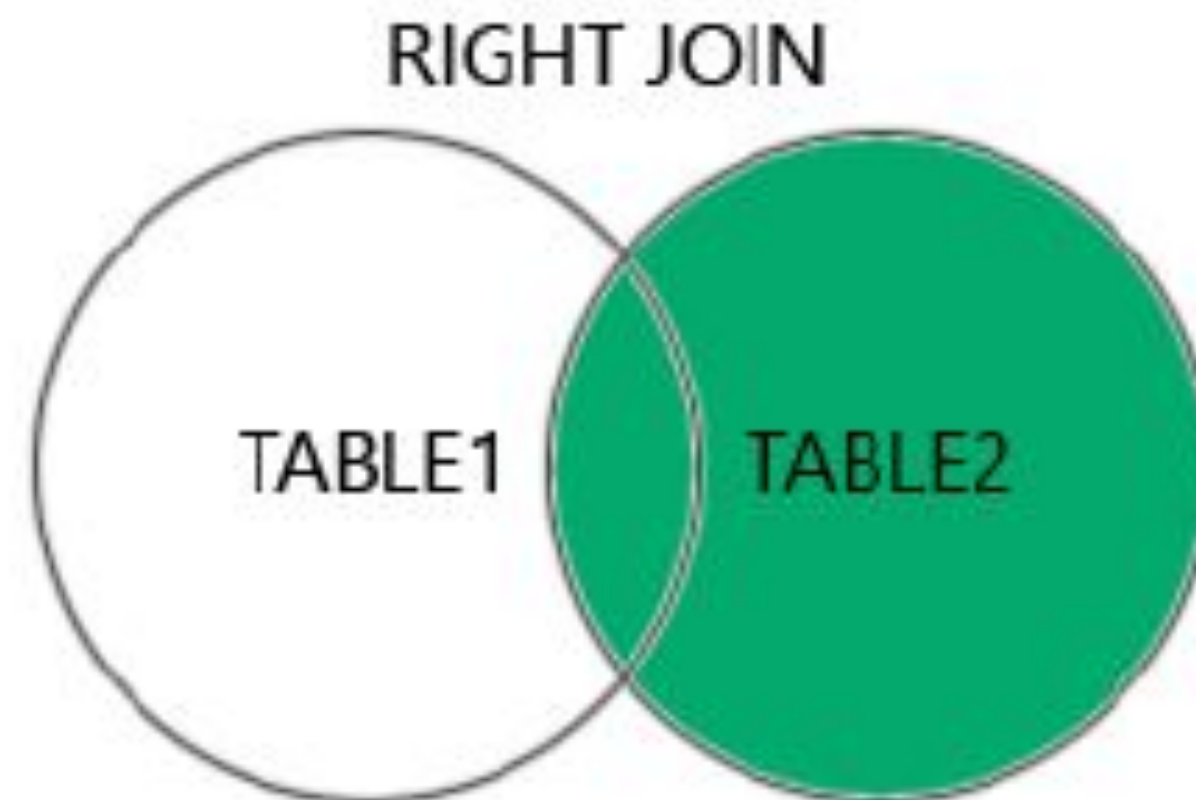


```
SELECT <Lista de Atributos>  
FROM Tabela1  
LEFT JOIN Tabela2 ON Tabela1.Atributo = Tabela2.Atributo;
```



JOIN

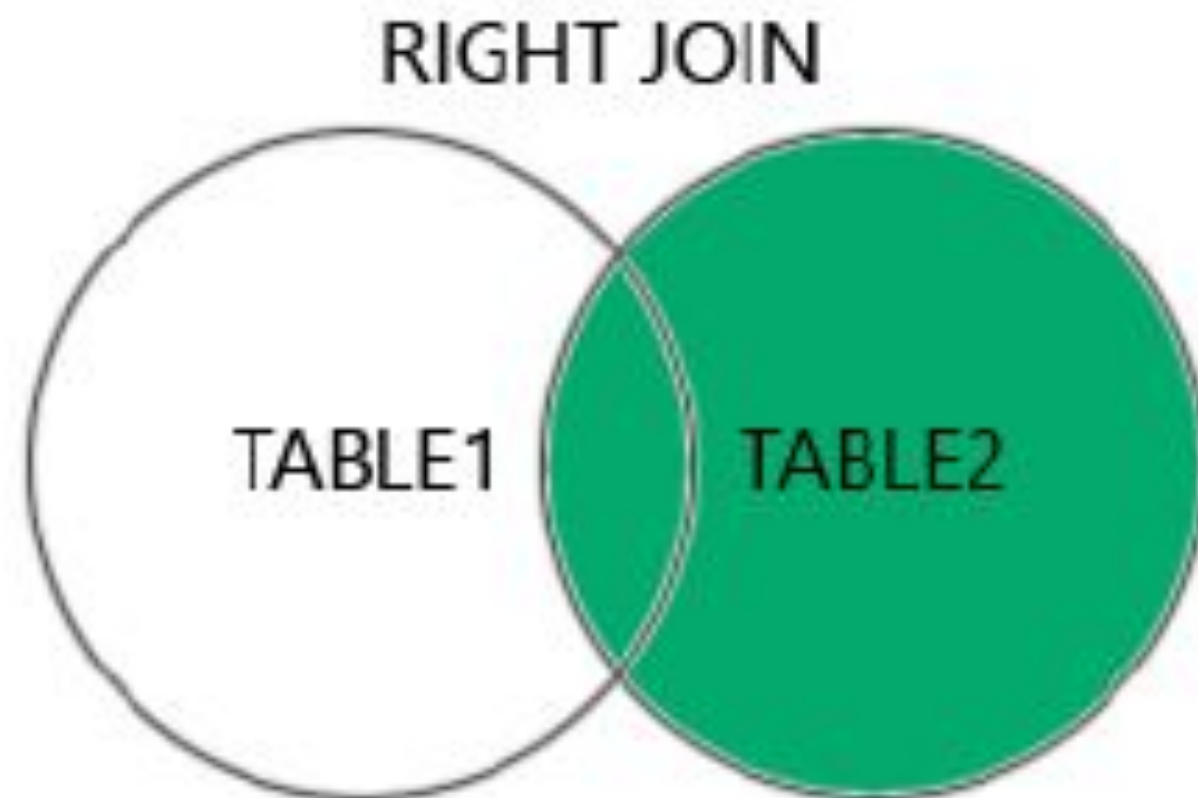
- **Right Join:** É o inverso do Left Join. Retorna os registros da tabela 2 e os registros correspondentes da tabela à esquerda. Em caso de não correspondência, registros da tabela à esquerda virão como NULL.



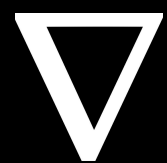


JOIN

- **Right Join:** É o inverso do Left Join. Retorna os registros da tabela 2 e os registros correspondentes da tabela à esquerda. Em caso de não correspondência, registros da tabela à esquerda virão como NULL.



```
SELECT <Lista de Atributos>  
FROM Tabela1  
RIGHT JOIN Tabela2 ON Tabela1.Atributo = Tabela2.Atributo;
```



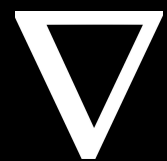
JOIN

- Suponha que temos as relações Clientes e Pedidos

ID_Cliente	Nome	Cidade
1	Alice Silva	São Paulo
2	Bob Souza	São Carlos
3	Maria Oliveira	Rio de Janeiro
4	Gabriel Santos	Minas Gerais

ID_Pedido	ID_Cliente	Produto	Valor
101	1	Notebook	3500.00
102	3	Mouse	80.00
103	2	Teclado	150.00
104	2	Monitor	1100.00

Queremos criar uma consulta que mostre o nome de cada cliente e seu respectivo produto. Para isso, observe que a relação Produtos possui como chave estrangeira o id do cliente. Logo, podemos utilizar isso para realizar o JOIN.



JOIN

SELECT

Clientes.Nome,
Pedidos.Produto,
Pedidos.Valor

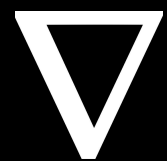
FROM

Clientes

INNER JOIN

Pedidos ON Clientes.ID_Cliente = Pedidos.ID_Cliente;

Nome	Produto	Valor
Alice Silva	Notebook	3500.00
Maria Oliveira	Mouse	80.00
Bob Souza	Teclado	150.00
Bob Souza	Monitor	1100.00



JOIN

Em situações em que precisamos escrever inúmeros atributos (em SELECT, WHERE, JOIN, etc.) de uma relação, ao invés de utilizar a estrutura Nome_Relação.Atributo, podemos utilizar o denominado aliases (apelido).

SELECT

C.Nome,

Pedidos.Produto,

Pedidos.Valor

FROM

Clientes AS C

INNER JOIN

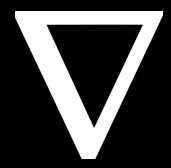
Pedidos AS P ON C.ID_Cliente = P.ID_Cliente;

Nome	Produto	Valor
Alice Silva	Notebook	3500.00
Maria Oliveira	Mouse	80.00
Bob Souza	Teclado	150.00
Bob Souza	Monitor	1100.00



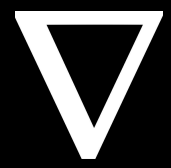
SUBQUERY

- Consulta dentro de outra consulta
- Usos: WHERE, SELECT, FROM
- Vantagens: legibilidade, modularidade
- Desvantagens: performance inferior a JOIN em alguns casos



SUBQUERY

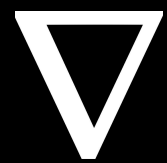
- **Filtragem de dados (cláusula WHERE):** Para filtrar registros da consulta principal com base no resultado da subquery.
- **Cálculos intermediários (cláusula SELECT):** Para retornar um único valor que será usado como uma nova coluna na sua consulta principal.
- **Tabelas temporárias (cláusula FROM):** Para criar uma tabela virtual "em tempo real" que será usada na consulta principal.



Subquery na cláusula WHERE

Vamos supor que você tem uma tabela de Funcionarios e uma tabela de Departamentos. Você quer encontrar o nome de todos os funcionários que trabalham no departamento chamado 'Vendas'.

```
SELECT
    Nome
FROM
    Funcionarios
WHERE
    ID_Departamento = (
    SELECT
        ID_Departamento
    FROM
        Departamentos
    WHERE
        Nome_Departamento = 'Vendas'
    );
```

Subquery na cláusula WHERE

Você também pode usar operadores como IN para lidar com múltiplos resultados da subquery. Por exemplo, para encontrar os funcionários que trabalham em 'Vendas' ou 'Marketing', a subquery retornaria mais de um ID_Departamento.

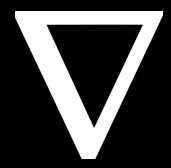
```
SELECT
    Nome
FROM
    Funcionarios
WHERE
    ID_Departamento IN (
    SELECT
        ID_Departamento
    FROM
        Departamentos
    WHERE
        Nome_Departamento IN ('Vendas', 'Marketing')
    );
```



Subquery na cláusula **SELECT**

Neste caso, a subquery atua como uma coluna, retornando um único valor para cada linha da consulta principal. Vamos supor que você tem uma tabela de Clientes e uma tabela de Pedidos. Você quer listar o nome de cada cliente e, ao lado, a quantidade total de pedidos que cada um fez.

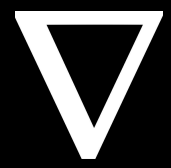
```
SELECT
    Nome_Cliente,
    (
        SELECT
            COUNT(ID_Pedido)
        FROM
            Pedidos
        WHERE
            Pedidos.ID_Cliente = Clientes.ID_Cliente
    ) AS Total_Pedidos
FROM
    Clientes;
```



Subquery na cláusula FROM

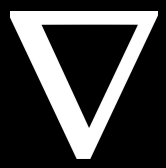
A subquery retorna um conjunto de resultados que é tratado como uma nova tabela virtual temporária na cláusula FROM. Essa "tabela" pode então ser usada na consulta principal.

Imagine que você quer encontrar o nome dos clientes que fizeram mais de dois pedidos. Para isso, primeiro você precisa agrupar os clientes por ID e contar seus pedidos. A subquery faria isso, e a consulta principal usaria o resultado.



Subquery na cláusula FROM

```
SELECT
    C.Nome
FROM
    Clientes AS C
INNER JOIN
    (
        SELECT
            ID_Cliente,
            COUNT(*) AS Total_Pedidos
        FROM
            Pedidos
        GROUP BY
            ID_Cliente
        HAVING
            COUNT(*) > 2
    ) AS Clientes_Com_Mais_De_2_Pedidos
ON
    C.ID_Cliente = Clientes_Com_Mais_De_2_Pedidos.ID_Cliente;
```

Vantagens e Desvantagens das Subqueries

VANTAGENS

Legibilidade: facilita a leitura de consultas complexas, dividindo-as em partes lógicas.

Flexibilidade: permitem operações que seriam difíceis ou impossíveis com outros comandos.

Modularidade: possibilitam pensar no problema em etapas, onde cada subquery resolve uma parte do problema maior.

DESVANTAGENS

Performance: podem ser menos eficientes que JOINS em certos casos, dependendo do otimizador do SGBD.

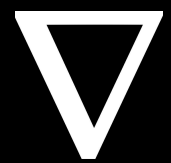
Complexidade: subqueries muito aninhadas tornam o código difícil de depurar e manter.

Retorno: subqueries em SELECT e em WHERE (sem IN) só podem retornar um único valor.



Vantagens e Desvantagens das Subqueries

Em geral, prefira **JOIN** para unir tabelas quando a operação for simplesmente combinar dados — é a abordagem mais performática na maioria dos casos. Use **subqueries** quando precisar de filtros dinâmicos, cálculos intermediários ou quando a lógica do problema se encaixar melhor em "consulta dentro de consulta".



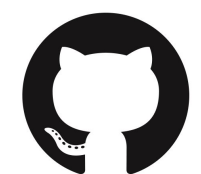
data@icmc.usp.br



@data.icmc



/c/DataICMC



/icmc-data



data.icmc.usp.br

|| obrigado!