ANOTAÇÕES DOS ESTUDOS DE BANCO DE DADOS (MYSQL)

FASE 4: Consultas Avançadas

Sumário

1. Agrupamento e Filtragem

O agrupamento e a filtragem são conceitos fundamentais em SQL para realizar análises em tabelas, permitindo agrupar dados semelhantes e aplicar condições em diferentes níveis.

1.1. Agrupamento com GROUP BY

O GROUP BY é usado para agrupar linhas que têm valores iguais em uma coisa ou mais colunas, permitindo a aplicação de funções de agregação (como *SUM, COUNT, AVG, MAX, MIN*) **em cada grupo.**

Obs.: No caso do GROUP BY, todas as colunas no SELECT que não são usadas em uma função de agregação devem ser parte da cláusula GROUP BY. Ou seja, quando você agrupa por uma coluna, todas as outras colunas no SELECT também precisam ser ou agregadas com uma função, ou precisam ser incluídas no GROUP BY.

- **Exemplo prático**

Tabela: vendas

id	produto	quantidade	valor_unitario	categoria
1	Celular Samsung	2	1500.00	Eletrônicos
2	Notebook Dell	1	3500.00	Eletrônicos
3	Camisa Polo	3	80.00	Vestuário
4	Geladeira Brastemp	1	2400.00	Eletrodomésticos
5	Camisa Polo	2	80.00	Vestuário

- Faça uma consulta que agrupe as vendas por categoria e calcule o total de quantidade vendida.

```
SELECT categoria, SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY categoria;
```

- **Resultado:**

categoria	total_vendido
Eletrônicos	3
Vestuário	5
Eletrodomésticos	1

1.2. Filtragem com HAVING

O HAVING é usado para filtrar grupos criados pelo GROUP BY com base em condições aplicadas aos resultados das funções de agregação.

- **Diferença entre WHERE e HAVING:**
  - **WHERE:** Filtra os dados antes de serem agrupados.
  - **HAVING:** Filtra os dados após serem agrupados.

- **Exemplo prático:**
  - Filtrar apenas as categorias com total de vendas maior que 3:

```
SELECT categoria, SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY categoria
HAVING total_vendido > 3;
```

  - **Resultado:**

categoria	total_vendido
Vestuário	5

1.3. Usando GROUP BY com múltiplas colunas

É possível agrupar dados com base em mais de uma coluna.

- **Exemplo:**

```sql
SELECT produto, categoria, SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY produto, categoria;
```

- **Resultado:**

produto	categoria	total_vendido
Celular Samsung	Eletrônicos	2
Notebook Dell	Eletrônicos	1
Camisa Polo	Vestuário	5
Geladeira Brastemp	Eletrodomésticos	1

1.4. Filtragem avançada com WHERE e HAVING juntos

Você pode combinar WHERE e HAVING na mesma consulta para aplicar filtros em diferentes momentos.

- **Exemplo:**
  1. Considere apenas vendas de produtos da categoria “Eletrônicos”.
  2. Entre esses, agrupe por produto e mostre apenas os produtos com total de vendas maior que 1.

```sql
SELECT produto, SUM(quantidade) AS total_vendido
FROM vendas
WHERE categoria = 'Eletrônicos'
GROUP BY produto
HAVING total_vendido > 1;
```

- **Resultado:**

produto	total_vendido
Celular Samsung	2

1.5. Ordenando os resultados de agrupamentos

O ORDER BY pode ser usado junto com GROUP BY para organizar resultados.

- **Exemplo:**
  - Ordenar as categorias por total de vendas em ordem decrescente:

```
SELECT categoria, SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY categoria
ORDER BY total_vendido DESC;
```

- **Resultado:**

categoria	total_vendido
Vestuário	5
Eletrônicos	3
Eletrodomésticos	1

1.6. Agrupando todos os dados com ROLLUP (opcional)

O ROLLUP permite gerar totais cumulativos ao final de um agrupamento.

- **Exemplo:**
  - Calcular o total de vendas por categoria e o total geral:

```
SELECT categoria, SUM(quantidade) AS total_vendido
FROM vendas
GROUP BY categoria WITH ROLLUP;
```

- **Resultado:**

categoria	total_vendido
Eletrônicos	3
Vestuário	5
Eletrodomésticos	1
NULL	9

NULL indica o total geral.

2. Exercício prático 01 (Agrupamento e Filtragem)

Você vai trabalhar com o seguinte cenário: uma loja online que gerencia dados sobre clientes, produtos, pedidos e categorias. O objetivo é realizar agrupamentos, aplicar filtragens e utilizar funções de agregação para responder a questões importantes do negócio.

1. Criação das Tabelas

Crie as seguintes tabelas usando os comandos DDL:

Tabela: clientes

Coluna	Tipo de Dados	Restrição
id	INT	Chave primária, Auto Increment
nome	VARCHAR(100)	Não nulo
cidade	VARCHAR(100)	Não nulo

Tabela: produtos

Coluna	Tipo de Dados	Restrição
id	INT	Chave primária, Auto Increment
nome	VARCHAR(100)	Não nulo
preco	DECIMAL(10,2)	Não nulo
id_categoria	INT	Não nulo

Tabela: categorias

Coluna	Tipo de Dados	Restrição
id	INT	Chave primária, Auto Increment
nome	VARCHAR(100)	Não nulo

Tabela: pedidos

Coluna	Tipo de Dados	Restrição
id	INT	Chave primária, Auto Increment
id_cliente	INT	Não nulo, Chave estrangeira (clientes.id)
id_produto	INT	Não nulo, Chave estrangeira (produtos.id)
quantidade	INT	Não nulo
data_pedido	DATE	Não nulo

2. Inserção de Dados

Insira os seguintes registros nas tabelas:

Clientes

id	nome	cidade
1	João Silva	São Paulo
2	Maria Oliveira	Belo Horizonte
3	Pedro Santos	Rio de Janeiro
4	Ana Costa	Salvador

Categorias

id	nome
1	Eletrônicos
2	Vestuário
3	Eletrodomésticos

Produtos

id	nome	preco	id_categoria
1	Celular Samsung	1500.00	1
2	Notebook Dell	3500.00	1
3	Camisa Polo	80.00	2
4	Geladeira	2400.00	3

Pedidos

id	id_cliente	id_produto	quantidade	data_pedido
1	1	1	2	2025-01-15
2	2	3	3	2025-01-16
3	3	2	1	2025-01-17
4	4	4	1	2025-01-18
5	1	3	2	2025-01-19

3. Consultas Práticas

Resolva as seguintes questões utilizando **agrupamento** e **filtragem**:

Parte 1: Análise de Produtos

1. Liste todas as categorias e o total de produtos disponíveis em cada uma.
2. Calcule o preço médio dos produtos em cada categoria.
3. Encontre as categorias onde o preço médio dos produtos é maior que 1000.

Parte 2: Análise de Pedidos

4. Liste o total de produtos vendidos agrupados por cliente (use o nome do cliente).
5. Calcule a quantidade total de produtos vendidos agrupados por categoria.
6. Mostre apenas os clientes que compraram mais de 2 produtos no total.
7. Encontre o total de vendas (quantidade * preço) para cada cliente e ordene do maior para o menor.

Parte 3: Filtragens Avançadas

8. Liste os pedidos realizados entre as datas 2025-01-16 e 2025-01-18.
9. Mostre os clientes que compraram produtos da categoria "Eletrônicos".
10. Exiba as categorias de produtos que **não foram vendidos**.

4. Desafios Avançados

1. Calcule a receita total gerada por cada categoria de produto.
2. Encontre os produtos mais vendidos (com maior quantidade total) e os menos vendidos.
3. Liste os clientes que realizaram pedidos em mais de uma data diferente.

3. JOINs (Junções)

Os JOINs (junções) são usados em SQL para combinar dados de duas ou mais tabelas baseadas em colunas relacionadas.

Esse recurso é essencial quando os dados de um banco relacional estão distribuídos em diferentes tabelas e precisamos cruzar essas informações.

Para os exemplos abaixo com os JOINs, utilizaremos as seguintes tabelas:

Tabela clientes

id_cliente	nome	cidade
1	João Silva	São Paulo
2	Maria Lima	Rio de Janeiro
3	Pedro Costa	Belo Horizonte
4	Ana Souza	Curitiba

Tabela pedidos

id_pedido	id_cliente	produto	valor
101	1	Notebook	3500.00
102	2	Celular	2000.00
103	1	Mouse	150.00
104	3	Teclado	300.00

3.1. INNER JOIN

- Retorna apenas os registros que têm correspondência nas duas tabelas.
- É o tipo de JOIN mais comum.

- **Exemplo: Faça uma consulta que retorne os clientes que fizeram pedidos.**

```sql
SELECT clientes.nome, pedidos.produto, pedidos.valor
FROM clientes
INNER JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

- **Resultado:**

nome	produto	valor
João Silva	Notebook	3500.00
João Silva	Mouse	150.00
Maria Lima	Celular	2000.00
Pedro Costa	Teclado	300.00

OBS.: O cliente *Ana Souza* não aparece porque não tem pedidos cadastrados. O pedido 105 também não aparece porque o cliente 5 não existe na tabela *clientes*.

3.2. LEFT JOIN

- Retorna todas as linhas da tabela à esquerda e apenas os registros correspondentes da tabela à direita.
- Para linhas sem correspondência, as colunas da tabela à direita retornam **NULL**.

- **Exemplo: Faça uma consulta e retorne todos os clientes, incluindo aqueles sem pedidos.**

```sql
SELECT clientes.nome, pedidos.produto, pedidos.valor
FROM clientes
LEFT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

- **Resultado:**

nome	produto	valor
João Silva	Notebook	3500.00
João Silva	Mouse	150.00
Maria Lima	Celular	2000.00
Pedro Costa	Teclado	300.00
Ana Souza	NULL	NULL

3.3. RIGHT JOIN

- Retorna todas as linhas da tabela à direita e apenas as correspondências da tabela à esquerda.
- Para linhas sem correspondência, as colunas da tabela à esquerda retornam **NULL**.

- **Exemplo: Faça uma consulta que retorne todos os pedidos, incluindo aqueles sem clientes correspondentes.**

```
SELECT clientes.nome, pedidos.produto, pedidos.valor
FROM clientes
RIGHT JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

- **Resultado:**

nome	produto	valor
João Silva	Notebook	3500.00
João Silva	Mouse	150.00
Maria Lima	Celular	2000.00
Pedro Costa	Teclado	300.00
NULL	Monitor	1200.00

3.4. FULL JOIN (ou *FULL OUTER JOIN*)

- Retorna todas as linhas de ambas as tabelas, combinando correspondências;
- Linhas sem correspondência em uma das tabelas exibem NULL.

- **Exemplo: Faça uma consulta que ela retorne todos os clientes e todos os pedidos.**

```
SELECT clientes.nome, pedidos.produto, pedidos.valor
FROM clientes
FULL JOIN pedidos ON clientes.id_cliente = pedidos.id_cliente;
```

- **Resultado:**

nome	produto	valor
João Silva	Notebook	3500.00
João Silva	Mouse	150.00
Maria Lima	Celular	2000.00
Pedro Costa	Teclado	300.00
Ana Souza	NULL	NULL
NULL	Monitor	1200.00

3.5. CROSS JOIN

- Retorna o produto cartesiano das tabelas, ou seja, combina todas as linhas da tabela 1 com todas as linhas da tabela 2;
- Não usa uma condição **ON**.

- **Exemplo:**

```
SELECT clientes.nome, pedidos.produto
FROM clientes
CROSS JOIN pedidos;
```

Se houver *4 clientes e 5 pedidos*, o resultado terá 4 x 5 = 20 linhas.

3.6. Como saber qual tabela vem primeiro

A tabela da esquerda e a tabela da direita em um JOIN são definidas pela ordem em que você menciona na consulta SQL.

A primeira tabela mencionada na cláusula FROM é a tabela da esquerda;
A segunda tabela mencionada após o JOIN é a tabela da direita.

Vejamos a regra geral:

```
SELECT *
FROM tabela_esquerda
JOIN tabela_direita ON condição;
```

Caso ainda haja dúvidas sobre qual tabela colocar à esquerda ou à direita, tenha em mente o seguinte:

a. Pergunte-se: *Qual tabela deve sempre aparecer no resultado?*
b. Escolha *LEFT JOIN ou RIGHT JOIN* de acordo com qual tabela você quer priorizar.

3.7. Combinando JOINs com condições adicionais

Você pode adicionar cláusulas como WHERE, GROUP BY ou HAVING para tornar a consulta mais específica.

- **Exemplo:**
  - Listar os nomes dos clientes que fizeram compras acima de R$ 400:

```
SELECT clientes.nome, vendas.valor
FROM clientes
INNER JOIN vendas
ON clientes.id_cliente = vendas.id_cliente
WHERE vendas.valor > 400;
```

3.8. JOINs com Alias

Alias tornam suas consultas mais legíveis e fáceis de interpretar;

- **Exemplo:**

```sql
SELECT c.nome, v.valor
FROM clientes AS c
INNER JOIN vendas AS v
ON c.id_cliente = v.id_cliente;
```

4. Exercício prático 02 (JOINs)

1. Cenário

Você trabalha com um sistema de vendas e precisa organizar dados de **clientes**, **produtos** e **pedidos**.

Crie as tabelas abaixo e insira os dados fornecidos.

2. Estrutura das Tabelas

Tabela: clientes

id_cliente	nome	email
1	João Silva	joao@email.com
2	Maria Souza	maria@email.com
3	Pedro Lima	pedro@email.com
4	Ana Costa	ana@email.com

Tabela: produtos

id_produto	nome_produto	preco
1	Laptop Dell	5000.00
2	Smartphone Samsung	2000.00
3	Fone de Ouvido	300.00
4	Monitor LG	1500.00

Tabela: pedidos

id_pedido	id_cliente	id_produto	quantidade	data_pedido
1	1	1	1	2025-01-15
2	2	2	2	2025-01-16
3	3	4	1	2025-01-17
4	NULL	3	3	2025-01-18

3. Objetivo

Com base nos dados acima, resolva as consultas abaixo utilizando diferentes tipos de JOINs.

4. Consultas

1. **INNER JOIN:**
    - Liste os nomes dos clientes, os produtos comprados e a quantidade adquirida.

2. **LEFT JOIN:**
    - Liste todos os clientes e, caso tenham realizado pedidos, mostre os produtos comprados. Inclua os clientes que não fizeram pedidos.

3. **RIGHT JOIN:**
    - Liste todos os produtos e, caso tenham sido comprados, exiba o nome do cliente que os adquiriu.

4. **FULL JOIN (Simule com LEFT JOIN e UNION):**
    - Liste todos os clientes e todos os produtos, mesmo que não tenham relação entre si.

5. **Usando JOINs com condições:**
    - Liste os nomes dos clientes que compraram produtos cujo preço seja maior que R$ 2.000, mostrando também o nome do produto e o valor total da compra (quantidade * preço).

6. **CROSS JOIN:**
    - Combine todos os clientes com todos os produtos (produto cartesiano). Mostre o nome do cliente e o nome do produto.

5. Desafio Extra

Crie uma consulta que mostre o **valor total de vendas por cliente**. Para clientes que não realizaram pedidos, mostre o valor total como 0.00.

5. Subconsultas

Subconsultas, também conhecidas como subqueries, são consultas SQL aninhadas dentro de outra consulta.

Elas são usadas para retornar resultados que serão utilizados pela consulta externa, ajudando a resolver problemas complexos de forma organizada e modular.

- **Estrutura básica:**
  - Uma subconsulta é definida entre parênteses e pode ser usada em diversas partes de uma consulta principal, como na cláusula SELECT, WHERE, ou até mesmo em FROM.

```
SELECT coluna1, coluna2
FROM tabela_principal
WHERE coluna1 = (SELECT coluna
                 FROM outra_tabela
                 WHERE condição);
```

5.1. Tipos de Subconsultas

- **Subconsulta de linha única**
  - Retorna apenas um valor (uma única linha e coluna);
  - Usada com operadores de comparação, como =, <, >, <=, >=, ou <>.

  - Exemplo: *Qual é o produto mais caro da tabela produtos?*

```
SELECT nome_produto, preco
FROM produtos
WHERE preco = (SELECT MAX(preco) FROM produtos);
```

- **Subconsulta de múltiplas linhas:**
  - Retorna múltiplos valores (uma única coluna, mas várias linhas);
  - Usada com operadores como IN, ANY ou ALL.

  - **Exemplo:** *Quais clientes fizeram pedidos em janeiro de 2025?*

```sql
SELECT nome
FROM clientes
WHERE id_cliente IN (SELECT id_cliente
                     FROM pedidos
                     WHERE data_pedido BETWEEN '2025-01-01' AND '2025-01-31');
```

- **Subconsulta de múltiplas colunas:**
  - Retorna várias colunas (uma ou mais linhas);
  - Usada geralmente em cláusulas como FROM ou EXISTS.

  - **Exemplo:** *Liste todos os clientes com seus pedidos.*

```sql
SELECT nome, data_pedido, quantidade
FROM clientes
JOIN (SELECT id_cliente, data_pedido, quantidade
      FROM pedidos) AS pedidos_detalhados
ON clientes.id_cliente = pedidos_detalhados.id_cliente;
```

5.2. Uso em diferentes cláusulas

- **Subconsulta em <u>SELECT</u>:**
  - Permite calcular ou processar valores para cada linha.

  - **Exemplo:** *Mostre o nome de cada cliente e o número de pedidos que ele fez:*

```sql
SELECT nome,
       (SELECT COUNT(*)
        FROM pedidos
        WHERE pedidos.id_cliente = clientes.id_cliente) AS total_pedidos
FROM clientes;
```

- **Subconsulta em <u>WHERE</u>:**
  - Filtra os resultados com base nos valores retornados pela subconsulta.

  - **Exemplo:** *Lista os produtos cujo preço é maior que a média dos preços:*

```sql
SELECT nome_produto, preco
FROM produtos
WHERE preco > (SELECT AVG(preco) FROM produtos);
```

- **Subconsulta em FROM:**
  - Cria uma tabela derivada que pode ser usada na consulta principal.
  - **Exemplo:** *Mostre a média de preços dos produtos em categorias diferentes.*

```sql
SELECT categoria, AVG(preco) AS preco_medio
FROM (SELECT categoria, preco FROM produtos) AS tabela_derivada
GROUP BY categoria;
```

- **Subconsulta com EXISTS:**
  - Verifica se uma subconsulta retorna resultados (TRUE ou FALSE).
  - **Exemplo:** *Liste os clientes que já fizeram pedidos.*

```sql
SELECT nome
FROM clientes
WHERE EXISTS (SELECT 1
              FROM pedidos
              WHERE clientes.id_cliente = pedidos.id_cliente);
```

- **Subconsulta com ANY e ALL:**
  - **ANY:** retorna verdadeiro se qualquer valor na subconsulta satisfizer a condição;
  - **ALL:** retorna verdadeiro se todos os valores na subconsulta satisfizerem a condição.
  - **Exemplo com ANY:** *Lista os produtos que custam mais do que qualquer produto na categoria "Eletrônicos".*

```sql
SELECT nome_produto
FROM produtos
WHERE preco > ANY (SELECT preco
                   FROM produtos
                   WHERE categoria = 'Eletrônicos');
```

- **Exemplo com ALL:** *Liste os produtos que são mais caros do que todos os produtos na categoria "Móveis":*

```sql
SELECT nome_produto
FROM produtos
WHERE preco > ALL (SELECT preco
                   FROM produtos
                   WHERE categoria = 'Móveis');
```

5.3. Boas práticas com subconsultas

1. Leia a lógica da consulta de dentro para fora:
   - Sempre entenda o que a subconsulta está retornando antes de combinar com a consulta externa.

2. Use subconsultas apenas quando necessário:
   - Evite subconsultas desnecessárias, pois podem impactar o desempenho.

3. Otimize o uso de índices:
   - Certifique-se de que as colunas usadas na subconsulta estão indexadas, especialmente em grandes bases de dados.

4. Considere substituir subconsultas por JOINs:
   - Algumas subconsultas podem ser reescritas como JOINs, que são geralmente mais eficientes.

6. Exercício prático 03 (Subconsultas)

Neste exercício, você terá que usar subconsultas para resolver diferentes questões sobre um sistema de gerenciamento de estoque e vendas.

1. Cenário

Imagine que você gerencia um banco de dados para uma loja que contém informações sobre produtos, clientes e pedidos. As tabelas abaixo foram criadas (as mesmas do exercício 02):

Tabela: clientes

id_cliente	nome	email
1	João Silva	joao@email.com
2	Maria Souza	maria@email.com
3	Pedro Lima	pedro@email.com
4	Ana Costa	ana@email.com

Tabela: produtos

id_produto	nome_produto	preco	estoque
1	Laptop Dell	5000.00	10
2	Smartphone Samsung	2000.00	15
3	Fone de Ouvido	300.00	50
4	Monitor LG	1500.00	8

Tabela: pedidos

id_pedido	id_cliente	id_produto	quantidade	data_pedido
1	1	1	1	2025-01-15
2	2	2	2	2025-01-16
3	3	4	1	2025-01-17

4	NULL	3	3	2025-01-18

2. Objetivo

Resolva as consultas abaixo utilizando **subconsultas**:

Consultas

1. **Valor Total do Pedido**
    - Liste o id_pedido, o id_cliente e o valor total de cada pedido. Use uma subconsulta para calcular o valor total com base no preço do produto e na quantidade.

2. **Estoque Abaixo da Média**
    - Liste os produtos cujo estoque está abaixo da média de todos os produtos.

3. **Clientes Sem Pedidos**
    - Liste os nomes e e-mails dos clientes que ainda não realizaram pedidos.

4. **Produto Mais Caro**
    - Mostre o nome e o preço do produto mais caro da loja.

5. **Clientes que Compraram Produtos Específicos**
    - Liste os nomes dos clientes que compraram qualquer produto com preço acima de R$ 3.000.

6. **Produtos com Pedido em Janeiro de 2025**
    - Liste os nomes dos produtos que foram pedidos no mês de janeiro de 2025.

7. **Pedidos com Estoque Insuficiente**

- Verifique se existe algum pedido em que a quantidade solicitada seja maior do que o estoque do produto.

8. **Quantidade de Pedidos por Cliente**
    - Liste o nome dos clientes e o número de pedidos que cada um realizou. Inclua os clientes que não realizaram pedidos (mostre como 0).

3. Desafio Extra

9. **Valor Total das Vendas por Cliente**
    - Liste o nome de cada cliente e o valor total de todas as compras que ele realizou. Inclua clientes que não realizaram compras (mostre o valor como 0.00).

4. Dicas

- Use SELECT, IN, EXISTS, ANY, ou ALL conforme necessário.
- Para calcular agregações, utilize funções como AVG(), SUM(), MAX(), etc.
- Para tratar valores ausentes (clientes sem pedidos), utilize LEFT JOIN junto com subconsultas.