Teste de SQL enviado pela VersoTech.
Candidato: Maurício Arthur Quednau da Costa

As queries foram criadas e testadas utilizando a ferramenta onecompiler (https://onecompiler.com/postgresql/) com base nos scripts de inserção de dados enviados.

-- 1 - Consulta de Vendedores: Escreva uma query para listar todos os vendedores ativos, mostrando as colunas id, nome, salario. Ordene o resultado pelo nome em ordem ascendente.

```
select id_vendedor as id, nome, salario from VENDEDORES where
inativo <> TRUE order by nome;
```

Resultado:

```
 id |    nome    | salario
----+------------+---------
  2 | Vendedor B | 4000.00
  4 | Vendedor D | 3800.00
  1 | Vendedor Z | 3000.00
(3 rows)
```

-- 2 - Funcionários com Salário Acima da Média: Escreva uma query para listar os funcionários que possuem um salário acima da média salarial de todos os funcionários. A consulta deve mostrar as colunas id, nome, e salário, ordenadas pelo salário em ordem descendente.

```
select id_vendedor as id, nome, salario
from VENDEDORES
where salario > (select AVG(salario) from VENDEDORES)
order by salario desc;
```

Resultado:

```
 id |    nome    | salario
----+------------+---------
  5 | Vendedor E | 4200.00
  2 | Vendedor B | 4000.00
  4 | Vendedor D | 3800.00
(3 rows)
```

-- 3 - Resumo por cliente: Escreva uma query para listar todos os clientes e o valor total de pedidos já transmitidos. A consulta deve retornar as colunas id, razao_social, total, ordenadas pelo total em ordem descendente.

```
select
    c.id_cliente as id,
    c.razao_social,
    COALESCE(SUM(p.valor_total), 0) as total
```

```
from CLIENTES c
left join pedido p ON c.id_cliente = p.id_cliente
group by c.id_cliente, c.razao_social
order by total desc;
```

Resultado:

```
 id | razao_social | total
----+--------------+--------
  4 | Cliente D    | 530.00
  3 | Cliente C    | 430.00
  2 | Cliente B    | 350.00
  1 | Cliente A    | 250.00
  5 | Cliente E    |      0
(5 rows)
```

-- 4 - Situação por pedido: Escreva uma query que retorne a situação atual de cada pedido da base. A consulta deve retornar as colunas id, valor, data e situacao.

```
select
    id_pedido as id,
    valor_total as valor,
    data_emissao as data,
    CASE
        WHEN data_cancelamento IS NOT NULL THEN 'CANCELADO'
        WHEN data_faturamento IS NOT NULL THEN 'FATURADO'
        ELSE 'PENDENTE'
    END AS situacao
from pedido
order by id_pedido;
```

Resultado:

```
 id | valor  |    data    | situacao
----+--------+------------+-----------
  1 | 120.00 | 2023-07-06 | PENDENTE
  2 | 130.00 | 2023-07-07 | PENDENTE
  3 | 170.00 | 2023-07-08 | FATURADO
  4 | 180.00 | 2023-07-09 | CANCELADO
  5 | 210.00 | 2023-07-10 | PENDENTE
  6 | 220.00 | 2023-07-11 | FATURADO
  7 | 260.00 | 2023-07-12 | CANCELADO
  8 | 270.00 | 2023-07-13 | FATURADO
(8 rows)
```

-- 5 - Produtos mais vendidos: Escreva uma query que retorne o produto mais vendido (em quantidade), incluindo o valor total vendido deste produto, quantidade de pedidos em que ele apareceu e para quantos clientes diferentes ele foi vendido. A consulta deve retornar as colunas id_produto, quantidade_vendida, total_vendido, clientes, pedidos. Caso haja empate em quantidade de vendas, utilizar o total vendido como critério de desempate.

```
select
    i.id_produto,
    SUM(i.quantidade) as quantidade_vendida,
    SUM(i.preco_praticado * i.quantidade) as total_vendido,
    COUNT(DISTINCT i.id_pedido) as pedidos,
    COUNT(DISTINCT p.id_cliente) as clientes
from itens_pedido i
join pedido p ON i.id_pedido = p.id_pedido
group by i.id_produto
order by quantidade_vendida DESC, total_vendido DESC
limit 1;
```

Resultado:

```
 id_produto | quantidade_vendida | total_vendido | pedidos | clientes
------------+--------------------+---------------+---------+----------
          2 |                 12 |        220.00 |       3 |        2
(1 row)
```