```sql
-- Consultas avançadas

-- 1. Liste os clientes que possuem mais de 2 ordens de serviço
--  abertas.
SELECT c.nome AS Cliente, COUNT(os.id_os) AS
Total_Ordens_Abertas
FROM Clientes c
JOIN Ordens_de_Servico os ON c.id_cliente = os.id_cliente
WHERE os.status = 'Aberta'
GROUP BY c.id_cliente, c.nome
HAVING COUNT(os.id_os) > 2;

-- 2. Liste os funcionários com a quantidade de ordens de servi
-- ço em que participaram, exibindo apenas os que participaram
-- de mais de 3 ordens.
SELECT f.nome AS Funcionario, COUNT(fos.id_os) AS
Total_Ordens_Participadas
FROM Funcionarios f
JOIN Funcionarios_OS fos ON f.id_funcionario =
fos.id_funcionario
GROUP BY f.id_funcionario, f.nome
HAVING COUNT(fos.id_os) > 3;

-- 3. Mostre os veículos que receberam mais de R$ 1000,00 em se
-- rviços, considerando o somatório de valor_cobrado.
SELECT v.placa AS Placa_Veiculo, v.modelo AS
Modelo_Veiculo, SUM(sos.valor_cobrado * sos.quantidade) AS
Total_Servicos
FROM Veiculos v
JOIN Ordens_de_Servico os ON v.id_veiculo = os.id_veiculo
JOIN Servicos_OS sos ON os.id_os = sos.id_os
GROUP BY v.id_veiculo, v.placa, v.modelo
HAVING SUM(sos.valor_cobrado * sos.quantidade) > 1000.00;

-- 4. Liste os produtos com estoque abaixo da média geral de to
-- dos os produtos.
SELECT nome AS Produto, quantidade_estoque AS Estoque_Atual
FROM Produtos
WHERE quantidade_estoque < (SELECT AVG(quantidade_estoque)
FROM Produtos);

-- 5. Crie uma consulta que use uma tabela derivada para calcul
-- ar o total gasto por ordem de serviço (produtos + serviços)
-- e filtre as que ultrapassam R$ 500.
SELECT
    id_os,
    Total_Gasto
```

```sql
FROM (
    SELECT
        os.id_os,
        COALESCE(SUM(pos.quantidade * pos.valor_unitario),
0) + COALESCE(SUM(sos.quantidade * sos.valor_cobrado), 0)
AS Total_Gasto
    FROM Ordens_de_Servico os
    LEFT JOIN Produtos_OS pos ON os.id_os = pos.id_os
    LEFT JOIN Servicos_OS sos ON os.id_os = sos.id_os
    GROUP BY os.id_os
) AS GastoPorOS
WHERE Total_Gasto > 500.00;

-- -
-- Functions

-- 6. Crie uma function que receba o id_os e retorne o total da
--  ordem de serviço, somando serviços e produtos.
DELIMITER //
CREATE FUNCTION CalcularTotalOS(p_id_os INT) RETURNS
DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE v_total_servicos DECIMAL(10,2);
    DECLARE v_total_produtos DECIMAL(10,2);

    SELECT COALESCE(SUM(sos.valor_cobrado *
sos.quantidade), 0)
    INTO v_total_servicos
    FROM Servicos_OS sos
    WHERE sos.id_os = p_id_os;

    SELECT COALESCE(SUM(pos.valor_unitario *
pos.quantidade), 0)
    INTO v_total_produtos
    FROM Produtos_OS pos
    WHERE pos.id_os = p_id_os;

    RETURN v_total_servicos + v_total_produtos;
END //
DELIMITER ;

-- Exemplo de uso da function CalcularTotalOS
-- SELECT CalcularTotalOS(1); -- Substitua 1 pelo id_os desejad
-- o

-- 7. Crie uma function que receba o id_funcionario e retorne a
--  quantidade de ordens em que ele participou.
```

```sql
DELIMITER //
CREATE FUNCTION ContarOrdensFuncionario(p_id_funcionario
INT) RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE v_quantidade_ordens INT;

    SELECT COUNT(fos.id_os)
    INTO v_quantidade_ordens
    FROM Funcionarios_OS fos
    WHERE fos.id_funcionario = p_id_funcionario;

    RETURN v_quantidade_ordens;
END //
DELIMITER ;

-- Exemplo de uso da function ContarOrdensFuncionario
-- SELECT ContarOrdensFuncionario(1); -- Substitua 1 pelo id_fu
-- ncionario desejado

-- -
-- Stored Procedures

-- 8. Crie uma stored procedure que, dado um id_cliente, exiba
-- todas as ordens de serviço dele com total gasto e status.
DELIMITER //
CREATE PROCEDURE ExibirOrdensCliente(IN p_id_cliente INT)
BEGIN
    SELECT
        os.id_os,
        os.data_entrada,
        os.data_saida,
        os.status,
        CalcularTotalOS(os.id_os) AS Total_Gasto
    FROM Ordens_de_Servico os
    WHERE os.id_cliente = p_id_cliente;
END //
DELIMITER ;

-- Exemplo de uso da stored procedure ExibirOrdensCliente
-- CALL ExibirOrdensCliente(1); -- Substitua 1 pelo id_cliente
-- desejado

-- 9. Crie uma stored procedure para finalizar uma ordem de ser
-- viço, definindo a data_saida atual e alterando o status para
--  'Concluída'.
DELIMITER //
CREATE PROCEDURE FinalizarOrdemServico(IN p_id_os INT)
```

```
BEGIN
    UPDATE Ordens_de_Servico
    SET data_saida = CURRENT_TIMESTAMP,
        status = 'Concluída'
    WHERE id_os = p_id_os;
END //
DELIMITER ;

-- Exemplo de uso da stored procedure FinalizarOrdemServico
-- CALL FinalizarOrdemServico(1); -- Substitua 1 pelo id_os des
-- ejado

-- -
-- Views

-- 10. Crie uma view que exiba o resumo das ordens de serviço:
-- id, cliente, valor total (serviços + produtos), e status.
CREATE VIEW vw_ResumoOrdensServico AS
SELECT
    os.id_os,
    c.nome AS Cliente,
    CalcularTotalOS(os.id_os) AS Valor_Total,
    os.status
FROM Ordens_de_Servico os
JOIN Clientes c ON os.id_cliente = c.id_cliente;

-- Exemplo de uso da view vw_ResumoOrdensServico
-- SELECT * FROM vw_ResumoOrdensServico;

-- 11. Crie uma view com os produtos mais utilizados em ordens
-- de serviço (com contagem e total em reais usados).
CREATE VIEW vw_ProdutosMaisUtilizados AS
SELECT
    p.nome AS Produto,
    SUM(pos.quantidade) AS Quantidade_Total_Utilizada,
    SUM(pos.quantidade * pos.valor_unitario) AS
Total_Em_Reais
FROM Produtos p
JOIN Produtos_OS pos ON p.id_produto = pos.id_produto
GROUP BY p.nome
ORDER BY Quantidade_Total_Utilizada DESC;

-- Exemplo de uso da view vw_ProdutosMaisUtilizados
-- SELECT * FROM vw_ProdutosMaisUtilizados;

-- -
-- Triggers
```

```sql
-- 12. Crie uma trigger que, ao inserir um item em Produtos_OS,
--  desconte automaticamente o estoque do produto correspondent
-- e.
DELIMITER //
CREATE TRIGGER trg_DescontarEstoqueProdutoOS
AFTER INSERT ON Produtos_OS
FOR EACH ROW
BEGIN
    UPDATE Produtos
    SET quantidade_estoque = quantidade_estoque -
NEW.quantidade
    WHERE id_produto = NEW.id_produto;
END //
DELIMITER ;

-- 13. Crie uma trigger que, ao inserir um pagamento com status
--  'Recebido', altere o status da ordem de serviço corresponde
-- nte para 'Paga'.
DELIMITER //
CREATE TRIGGER trg_AtualizarStatusOSPaga
AFTER INSERT ON Pagamentos
FOR EACH ROW
BEGIN
    IF NEW.status = 'Recebido' THEN
        UPDATE Ordens_de_Servico
        SET status = 'Paga'
        WHERE id_os = NEW.id_os;
    END IF;
END //
DELIMITER ;

-- -
-- Extras (índices e performance)

-- 14. Compare o tempo de execução de uma consulta que busca or
-- dens de serviço por cliente com e sem índice no campo id_cli
-- ente.

-- Para testar, primeiro remova o índice se existir (se for a p
-- rimeira vez que você está executando, pode ignorar esta linh
-- a):
-- ALTER TABLE Ordens_de_Servico DROP INDEX idx_id_cliente_os;

-- Consulta SEM índice (execute e anote o tempo ou use EXPLAIN)
-- :
SELECT * FROM Ordens_de_Servico WHERE id_cliente = 1; --
Substitua 1 por um id_cliente existente
EXPLAIN SELECT * FROM Ordens_de_Servico WHERE id_cliente =
```

```sql
1;

-- Adicionar o índice para comparar:
CREATE INDEX idx_id_cliente_os ON Ordens_de_Servico
(id_cliente);

-- Consulta COM índice (execute novamente e compare o tempo):
SELECT * FROM Ordens_de_Servico WHERE id_cliente = 1; --
Substitua 1 por um id_cliente existente
EXPLAIN SELECT * FROM Ordens_de_Servico WHERE id_cliente =
1;

-- 15. Faça uma consulta que junte várias tabelas (cliente, veí
-- culo, OS, pagamento) e otimize-a utilizando índices adequado
-- s.

-- Adicionar índices para otimização em campos de JOIN e WHERE
-- comuns:
CREATE INDEX idx_id_veiculo_os ON Ordens_de_Servico
(id_veiculo);
CREATE INDEX idx_id_os_pagamentos ON Pagamentos (id_os);
-- (Os índices de FOREIGN KEY já são criados automaticamente pe
-- lo MySQL/MariaDB, mas aqui estamos sendo explícitos para dem
-- onstração)

SELECT
    c.nome AS Cliente,
    v.placa AS Placa_Veiculo,
    os.id_os AS Ordem_Servico_ID,
    os.status AS Status_OS,
    p.valor_total AS Valor_Pagamento,
    p.status AS Status_Pagamento
FROM Clientes c
JOIN Veiculos v ON c.id_cliente = v.id_cliente
JOIN Ordens_de_Servico os ON v.id_veiculo = os.id_veiculo
LEFT JOIN Pagamentos p ON os.id_os = p.id_os
WHERE c.data_cadastro >= '2023-01-01' AND os.status =
'Concluída';

EXPLAIN
SELECT
    c.nome AS Cliente,
    v.placa AS Placa_Veiculo,
    os.id_os AS Ordem_Servico_ID,
    os.status AS Status_OS,
    p.valor_total AS Valor_Pagamento,
    p.status AS Status_Pagamento
FROM Clientes c
```

```sql
JOIN Veiculos v ON c.id_cliente = v.id_cliente
JOIN Ordens_de_Servico os ON v.id_veiculo = os.id_veiculo
LEFT JOIN Pagamentos p ON os.id_os = p.id_os
WHERE c.data_cadastro >= '2023-01-01' AND os.status =
'Concluída';


-- 16. Desenvolva uma consulta pesada (ex: ordens por mês, tota
-- l gasto, cliente, status) e otimize seu desempenho.

-- Consulta Pesada (sem otimização explícita além das FKs e índ
-- ices criados acima):
SELECT
    YEAR(os.data_entrada) AS Ano,
    MONTH(os.data_entrada) AS Mes,
    c.nome AS Cliente,
    os.status AS Status_OS,
    COUNT(os.id_os) AS Quantidade_Ordens,
    SUM(CalcularTotalOS(os.id_os)) AS
Total_Gasto_Por_Cliente_Mes_Status
FROM Ordens_de_Servico os
JOIN Clientes c ON os.id_cliente = c.id_cliente
GROUP BY Ano, Mes, c.nome, os.status
ORDER BY Ano, Mes, Total_Gasto_Por_Cliente_Mes_Status DESC;

EXPLAIN
SELECT
    YEAR(os.data_entrada) AS Ano,
    MONTH(os.data_entrada) AS Mes,
    c.nome AS Cliente,
    os.status AS Status_OS,
    COUNT(os.id_os) AS Quantidade_Ordens,
    SUM(CalcularTotalOS(os.id_os)) AS
Total_Gasto_Por_Cliente_Mes_Status
FROM Ordens_de_Servico os
JOIN Clientes c ON os.id_cliente = c.id_cliente
GROUP BY Ano, Mes, c.nome, os.status
ORDER BY Ano, Mes, Total_Gasto_Por_Cliente_Mes_Status DESC;

-- Otimização: Adicionar índices em campos frequentemente usado
-- s em WHERE, GROUP BY e ORDER BY, e considerar pré-calculo se
--  a view for muito pesada ou usada frequentemente.
-- Neste caso, o `CalcularTotalOS` dentro do `SUM` pode ser um
-- gargalo se houver muitos registros. Uma abordagem mais otimi
-- zada seria calcular esses totais em uma view materializada o
-- u em um campo persistido.

-- Adicionar índice composto para GROUP BY e ORDER BY
```

```sql
CREATE INDEX idx_os_data_cliente_status ON
Ordens_de_Servico (data_entrada, id_cliente, status);

-- Consulta otimizada (sem JOINs aninhados, usando a função par
-- a total, mas com índices para as colunas de agrupamento):
SELECT
    YEAR(os.data_entrada) AS Ano,
    MONTH(os.data_entrada) AS Mes,
    c.nome AS Cliente,
    os.status AS Status_OS,
    COUNT(os.id_os) AS Quantidade_Ordens,
    SUM(CalcularTotalOS(os.id_os)) AS
Total_Gasto_Por_Cliente_Mes_Status
FROM Ordens_de_Servico os
JOIN Clientes c ON os.id_cliente = c.id_cliente
GROUP BY Ano, Mes, c.nome, os.status
ORDER BY Ano, Mes, Total_Gasto_Por_Cliente_Mes_Status DESC;

EXPLAIN
SELECT
    YEAR(os.data_entrada) AS Ano,
    MONTH(os.data_entrada) AS Mes,
    c.nome AS Cliente,
    os.status AS Status_OS,
    COUNT(os.id_os) AS Quantidade_Ordens,
    SUM(CalcularTotalOS(os.id_os)) AS
Total_Gasto_Por_Cliente_Mes_Status
FROM Ordens_de_Servico os
JOIN Clientes c ON os.id_cliente = c.id_cliente
GROUP BY Ano, Mes, c.nome, os.status
ORDER BY Ano, Mes, Total_Gasto_Por_Cliente_Mes_Status DESC;
```