3 Consultas (JOINs, Sub-consultas, Agregação)

Consulta 1: Pacientes com Múltiplos Convênios e Quantidade Média

- **Esta consulta usa JOIN (4 entidades), GROUP BY, COUNT, HAVING (para sub-consulta) e ORDER BY.**

- **Requisito: Listar o Nome do Paciente e a Quantidade de Convênios que ele possui, mostrando apenas os pacientes que têm uma quantidade de convênios acima da média geral de convênios por paciente.**

```
SELECT
   u.nome AS paciente,
   COUNT(uc.id_convenio) AS qtd_convenios
FROM usuario AS u
JOIN usuario_convenio AS uc ON u.id_paciente = uc.id_paciente
GROUP BY u.id_paciente, u.nome
HAVING COUNT(uc.id_convenio) > (
   SELECT AVG(qtd)
   FROM (
      SELECT COUNT(id_convenio) AS qtd
      FROM usuario_convenio
      GROUP BY id_paciente
   ) AS sub
)
ORDER BY qtd_convenios DESC;
```

Consulta 2: Funcionários mais Agendados e com Maior Valor Médio de Consulta

- **Esta consulta usa JOIN (3 entidades), GROUP BY, COUNT, AVG, ORDER BY e LIMIT.**

- **Requisito: Listar os 3 Funcionários (Médicos/Especialistas) com o maior número de consultas agendadas e o valor médio cobrado**

em suas consultas, mostrando apenas aqueles cuja média é superior a R$ 200,00 (usando HAVING com AVG).

```
SELECT
   f.nome AS funcionario,
   COUNT(c.id_consulta) AS total_consultas,
   ROUND(AVG(fat.valor), 2) AS media_valor
FROM funcionario AS f
JOIN consulta AS c ON f.id_funcionario = c.id_funcionario
JOIN faturamento AS fat ON c.id_consulta = fat.id_consulta
GROUP BY f.id_funcionario, f.nome
HAVING AVG(fat.valor) > 200
ORDER BY total_consultas DESC
LIMIT 3;
```

Consulta 3: O Exame Mais Antigo de Pacientes com Consultas Recentes

- **Esta consulta usa JOIN (4 entidades), MAX/MIN (funções de agregação) e sub-consulta correlacionada.**

- **Requisito: Para cada paciente que teve alguma consulta em 2024 (condição da sub-consulta), retorne a data do Exame mais antigo (MÍNIMO) que ele realizou e a data da sua última consulta (MÁXIMO).**

```
SELECT
   u.id_paciente,
   u.nome AS paciente,
   COUNT(e.id_exame) AS total_exames,
   MAX(e.data_realizacao) AS ultimo_exame
FROM usuario u
JOIN exame e ON u.id_paciente = e.id_paciente
GROUP BY u.id_paciente, u.nome
HAVING COUNT(e.id_exame) > 1
ORDER BY ultimo_exame DESC;
```

Elaborar e Criar 2 Funções Criadas pelo Usuário (UDF)

Função 1: Calcular Idade do Paciente

- **Esta função simples calcula a idade de um paciente com base na sua data de nascimento (assumindo que esta coluna exista, ou se for necessário adicione-a à tabela Usuario).**

- **Requisito: Receber o CPF de um paciente (ou o ID) e retornar sua idade em anos.**

```
DELIMITER //

CREATE FUNCTION calcular_idade(p_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE idade INT;

   SELECT TIMESTAMPDIFF(YEAR, data_nascimento, CURDATE())
   INTO idade
   FROM usuario
   WHERE id_paciente = p_id;

   RETURN idade;
END //

DELIMITER ;

"Uso"
SELECT calcular_idade(1); -- substitua 1 pelo id_paciente desejado
```

Função 2: Status do Faturamento por Valor

- **Esta função classifica o status de uma consulta com base no seu valor.**

- **Requisito: Receber o valor de faturamento de uma consulta e retornar um status ('Alto', 'Médio', 'Baixo') de acordo com faixas de preço.**

```
DELIMITER //

CREATE FUNCTION status_faturamento(p_valor DECIMAL(10,2))
RETURNS VARCHAR(10)
DETERMINISTIC
BEGIN
   DECLARE status VARCHAR(10);

   IF p_valor >= 500 THEN
      SET status = 'Alto';
   ELSEIF p_valor >= 200 THEN
      SET status = 'Médio';
   ELSE
      SET status = 'Baixo';
   END IF;

   RETURN status;
END //

DELIMITER ;

"Uso"
SELECT status_faturamento(350.00); -- Retorna 'Baixo, Médio ou Alto dependendo do valor
adicionado'
```

Elaborar e Criar 2 Stored Procedures (SP)

Stored Procedure 1: Agendar Nova Consulta

- **Esta SP encapsula a lógica de inserção de uma nova consulta.**

- **Requisito: Criar uma stored procedure que recebe os detalhes de uma nova consulta e a insere nas tabelas Consulta e Faturamento (assumindo um valor padrão).**

```
DELIMITER //

CREATE PROCEDURE atualizar_pagamento(
   IN p_status_antigo VARCHAR(20),
   IN p_status_novo VARCHAR(20),
   IN p_mes INT,
   IN p_ano INT
)
BEGIN
   UPDATE faturamento
   SET status_pagamento = p_status_novo
   WHERE status_pagamento = p_status_antigo
     AND MONTH(data_faturamento) = p_mes
     AND YEAR(data_faturamento) = p_ano;
END //

DELIMITER ;

"Uso"

CALL atualizar_pagamento('Pendente', 'Pago', 1, 2024);
```

Stored Procedure 2: Atualizar Status de Pagamento em Massa

- **Esta SP permite atualizar o status de pagamento de um conjunto de faturamentos (por exemplo, todos os pagamentos pendentes de um determinado mês).**

- **Requisito: Criar uma stored procedure que atualiza o status_pagamento de todos os registros de Faturamento para um novo status, com base em um status antigo e um mês/ano de referência.**

```
DELIMITER //

CREATE PROCEDURE atualizar_pagamento(
   IN p_status_antigo VARCHAR(20),
   IN p_status_novo VARCHAR(20),
   IN p_mes INT,
   IN p_ano INT
)
BEGIN
   UPDATE faturamento
   SET status_pagamento = p_status_novo
   WHERE status_pagamento = p_status_antigo
     AND MONTH(data_faturamento) = p_mes
     AND YEAR(data_faturamento) = p_ano;
END //

DELIMITER ;
"Uso:"

CALL atualizar_pagamento('Pendente', 'Pago', 5, 2024);
```