UNIVERSIDADE DE SÃO PAULO

ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

# FASE 4 - CONSULTAS SQL

Guilherme Oliveira de Souza NºUSP: 1467138

Júlio César Cordeiro Batista NºUSP: 12684333

Kennedy Rohab Menezes da Silva NºUSP: 12683395

São Paulo

2025

Com o intuito de evitar a descrição extensiva de código será mostrada apenas o cabeçalho resumido do método (com seu caminho no código fonte e sem o tipo retornado) e a consulta em SQL.

1. CONSULTAS COM SQL ANINHADO
    a. **Informações dos grupos a que um cliente pertence**

**Caminho**: \src\main\java\service\GrupoService.java

**Código:**

```
public getGrupos(Cliente cliente){
	…
	SELECT * FROM GRUPO
	WHERE id IN (
		SELECT id_grupo FROM MEMBROGRUPO
		WHERE id_cliente = ?)
	…
}
```

? é o id do cliente passado como parâmetro para `getGrupos()` o mesmo acontece para as outras funções que recebem um identificador para retornar o resultado da consulta.

**Descrição:**

A consulta SQL é exatamente o que está descrito no item desta seção: ele primeiro busca os ids dos grupos ao qual o cliente pertence, depois a consulta mais externa retorna as informações deste grupo como: id, nome, status, data_criacao e descricao

    b. **Maiores gastos de um cliente**

**Caminho**:\src\main\java\service\RelatorioService.java

**Código:**

```
public maioresGastos(int idClienteLogado) {
	SELECT t.data_transacao,
		g.nome as grupo,
		c.nome as cliente,
		cat.nome as categoria,
		t.valor
	FROM Transacao t
	JOIN Cliente c ON t.id_cliente = c.id
	JOIN Grupo g ON t.id_grupo = g.id
	JOIN Categoria cat ON t.id_categoria = cat.id
	WHERE t.id_grupo IN (
		SELECT id_grupo
```

```
            FROM MembroGrupo
            WHERE id_cliente = ?
        )
        AND t.valor < 0
        ORDER BY t.valor ASC
}
```

**Descrição:**

A consulta pega todas as transações negativas (gastos) feitas em qualquer grupo do qual o cliente logado participa. Ela junta as tabelas necessárias para trazer o nome do grupo, o nome do cliente, a categoria e a data da transação. Filtra apenas os grupos onde o cliente é membro e ordena os valores em ordem crescente, ou seja, os maiores gastos (valores mais negativos) aparecem primeiro.

2. CONSULTAS COM FUNÇÕES DE GRUPO

**a. Gastos por categoria**

**Caminho**: \src\main\java\service\GrupoService.java

**Código:**

```
public gastosDetalhadosPorCategoria(int idClienteLogado) {
        …
        SELECT cat.nome as categoria,
                COUNT(t.id) as quantidade,
                COALESCE(SUM(ABS(t.valor)), 0) as total,
                COALESCE(AVG(ABS(t.valor)), 0) as media,
                ROUND(
                    COALESCE(SUM(ABS(t.valor)), 0) * 100.0 / NULLIF(
                          (SELECT SUM(ABS(valor)) FROM Transacao
                                WHERE id_grupo IN (
                                      SELECT id_grupo
                                      FROM MembroGrupo
                                      WHERE id_cliente = ?
                          )AND valor < 0),
                                0
                    ), 2
                  )as percentual
        FROM Categoria cat
        LEFT JOIN Transacao t ON cat.id = t.id_categoria
              AND t.id_grupo IN (
                    SELECT id_grupo
                    FROM MembroGrupo
                    WHERE id_cliente = ?
              )
              AND t.valor < 0
              GROUP BY cat.id, cat.nome
              HAVING COUNT(t.id) > 0
              ORDER BY total DESC
        …
```

```
}
```

**Descrição:**

A consulta calcula um resumo dos gastos por categoria para todos os grupos dos quais o cliente participa. Ela usa funções de grupo para agregar os dados: COUNT() conta quantas transações existem em cada categoria; SUM(ABS(t.valor)) soma o total gasto (considerando valores negativos como positivos); AVG(ABS(t.valor)) calcula a média desses gastos; e o percentual mostra quanto aquela categoria representa dentro do total de gastos do usuário. Mesmo categorias sem transações aparecem no LEFT JOIN, mas o HAVING COUNT(t.id) > 0 garante que só entram categorias onde houve gasto. No fim, os resultados são ordenados pelo total gasto em cada categoria.

### b. Divisão de Gastos por Membro

**Caminho**: \src\main\java\service\GrupoService.java

**Código:**

```
public divisaoGastosPorMembro(int idClienteLogado) {
	…
	SELECT g.nome as grupo,
		c.nome as membro,
		COUNT(t.id) as transacoes,
		COALESCE(SUM(t.valor), 0) as total_gasto,
		COALESCE(AVG(t.valor), 0) as media_gasto
	FROM Grupo g
	JOIN MembroGrupo mg ON g.id = mg.id_grupo
	JOIN Cliente c ON mg.id_cliente = c.id
	LEFT JOIN Transacao t ON t.id_cliente = c.id AND t.id_grupo = g.id
	WHERE g.id IN (
		SELECT id_grupo
		FROM MembroGrupo
		WHERE id_cliente = ?
	)
	GROUP BY g.id, g.nome, c.id, c.nome
	ORDER BY g.nome, total_gasto DESC
	…
}
```

**Descrição:**

A consulta monta um panorama de quanto cada membro gasta em cada grupo do qual o cliente logado participa. Ela combina grupos, membros e transações, e usa funções de grupo para resumir os valores: COUNT(t.id) contabiliza quantas transações cada membro fez dentro do grupo; SUM(t.valor) soma tudo que ele gastou; AVG(t.valor) calcula a média desses gastos. Como o LEFT JOIN é usado nas transações, membros que não fizeram nenhuma transação ainda aparecem, mas

com valores zerados via COALESCE. No final, tudo é agrupado por grupo e por membro e ordenado pelo nome do grupo e pelo total gasto dentro de cada grupo.

3. OPERADORES DE CONJUNTO

**a. Lista transação por período**

**Caminho**: \src\main\java\service\TransacaoService.java

**Código:**

```
public getTransacoesPorPeriodo(int idCliente,
                                java.sql.Date dataInicio,
                                java.sql.Date dataFim){
      SELECT t.id,
            t.descricao,
            t.valor,
            t.data_transacao,
            t.id_cliente,
            t.id_grupo,
            t.id_categoria,
            c.nome as categoria_nome,
            c.descricao as categoria_desc
      FROM Transacao t
      JOIN Categoria c ON t.id_categoria = c.id
      WHERE t.id_cliente = ?
      AND DATE(t.data_transacao) BETWEEN ? AND ?
      …
}
```

**Descrição:**

Essa consulta lista todas as transações de um cliente dentro de um intervalo de datas. Ela junta cada transação com sua categoria para trazer o nome e a descrição da categoria, filtra pelo cliente e pelo período usando DATE(t.data_transacao) BETWEEN ? AND ?, e devolve tudo em ordem da transação mais recente para a mais antiga.

**b. Lista transações por categoria**

**Caminho**: \src\main\java\service\TransacaoService.java

**Código:**

```
      public getTransacoesPorCategoria(int idCliente, int
idCategoria){
            …
            SELECT t.id,
                  t.descricao,
                  t.valor,
                  t.data_transacao,
```

```
            t.id_cliente,
            t.id_grupo,
            t.id_categoria,
            c.nome as categoria_nome,
            c.descricao as categoria_desc
        FROM Transacao t
        JOIN Categoria c ON t.id_categoria = c.id
        WHERE t.id_cliente = ? AND t.id_categoria = ?
        ORDER BY t.data_transacao DESC
        …
}
```

**Descrição:**

Essa consulta lista todas as transações de um cliente dentro de um intervalo de datas. Ela junta cada transação com sua categoria para trazer o nome e a descrição da categoria, filtra pelo cliente e pelo período usando DATE(t.data_transacao) BETWEEN ? AND ?, e devolve tudo em ordem da transação mais recente para a mais antiga.