

Funções de Grupo, Subqueries e Pesquisa Avançada em SQL

Transformando dados brutos em informação açãoável



Décima Segunda Aula (UC2)
Ciência de Dados

Objetivos da Aula

-  Compreender e aplicar **funções de grupo** para análise de dados (COUNT, SUM, AVG, MAX, MIN)
-  Dominar a cláusula **GROUP BY** para agrupamento de resultados por categorias
-  Utilizar **HAVING** para filtrar grupos de dados após o agrupamento
-  Implementar **subqueries** para criar consultas complexas e aninhadas
-  Aplicar técnicas de **pesquisa avançada** com LIKE e CASE
-  Finalizar o projeto de banco de dados com **recursos avançados** de segurança e procedimentos armazenados

Progressão de Habilidades em SQL

Abertura e Contexto

“ Um analista não precisa de todos os dados, precisa dos resumos. ”

Da Coleta à Informação Acionável

Esta é nossa **Última aula prática** da unidade curricular, focada em transformar dados brutos em informação açãovel usando as poderosas ferramentas de agregação e subconsultas do SQL.

Em um mundo onde o volume de dados cresce exponencialmente, a capacidade de **resumir, filtrar e analisar** informações de forma eficiente se torna uma habilidade essencial para qualquer profissional de dados.

-  As funções de grupo e subqueries são ferramentas fundamentais que permitem extrair insights valiosos de grandes conjuntos de dados.
-  Dominar estas técnicas avançadas de SQL marca a transição de um usuário básico para um analista de dados proficiente.

Funções de Grupo (Agregação)

Funções de grupo (ou funções de agregação) trabalham em um conjunto de linhas para retornar um único valor resumido. São essenciais para análise de dados e geração de relatórios.

COUNT()

Conta o número de linhas ou valores não nulos em uma coluna.

```
-- Conta todos os funcionários  
SELECT COUNT(*) AS TotalFuncionarios  
FROM Funcionarios;  
  
-- Conta funcionários com email cadastrado  
SELECT COUNT>Email) AS FuncionariosComEmail  
FROM Funcionarios;
```

SUM()

Calcula a soma dos valores em uma coluna numérica.

```
-- Calcula o total da folha de pagamento  
SELECT SUM(Salario) AS TotalFolhaPagamento  
FROM Funcionarios;
```

AVG()

Calcula a média dos valores em uma coluna numérica.

```
-- Calcula o salário médio  
SELECT AVG(Salario) AS SalarioMedio  
FROM Funcionarios;
```

↑ MAX()

Retorna o maior valor em uma coluna.

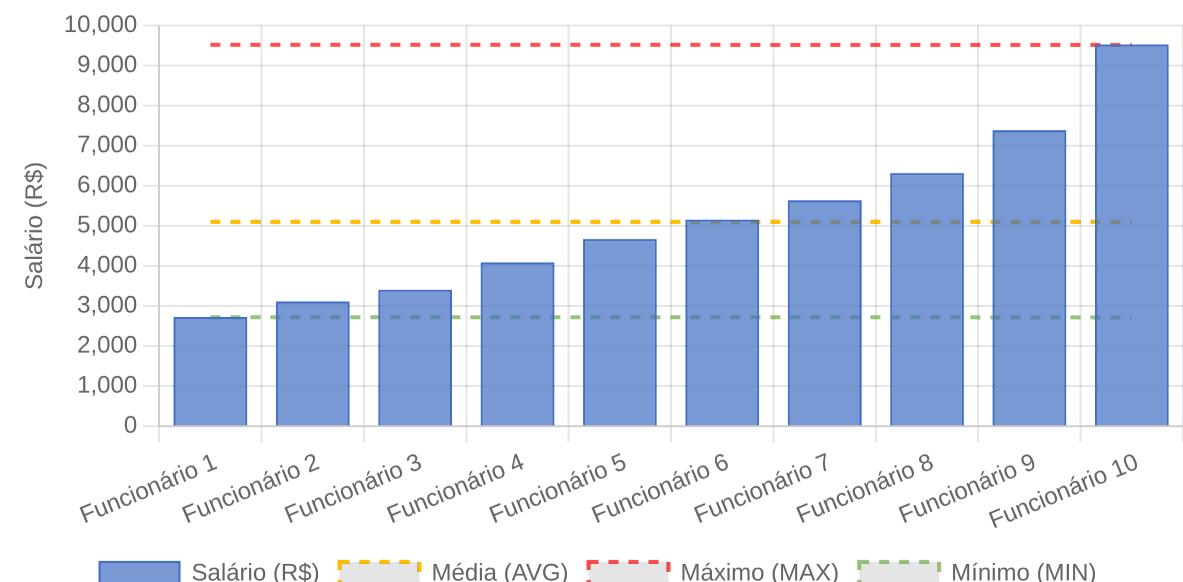
```
-- Encontra o maior salário  
SELECT MAX(Salario) AS MaiorSalario  
FROM Funcionarios;
```

↓ MIN()

Retorna o menor valor em uma coluna.

```
-- Encontra o menor salário  
SELECT MIN(Salario) AS MenorSalario  
FROM Funcionarios;
```

Visualização de Funções de Agregação



Função	Resultado	Descrição
COUNT(*)	10	Total de funcionários
SUM(Salario)	52.500,00	Soma de todos os salários
AVG(Salario)	5.250,00	Média salarial
MAX(Salario)	9.800,00	Maior salário
MIN(Salario)	2.800,00	Menor salário

Cláusula GROUP BY

Cláusula HAVING

💡 O que é HAVING?

A cláusula **HAVING** é utilizada para **filtrar grupos** de registros após o agrupamento com GROUP BY. Ela funciona como um WHERE para grupos, permitindo aplicar condições às funções de agregação.

-- Estrutura básica

```
SELECT coluna1, COUNT(coluna2)
FROM tabela
GROUP BY coluna1
HAVING COUNT(coluna2) > 5;
```

WHERE vs. HAVING

WHERE Filtra **linhas individuais** antes do agrupamento.
Não pode conter funções de agregação.

HAVING Filtra **grupos** após o agrupamento.
Pode conter funções de agregação.

💡 Use HAVING quando precisar filtrar com base em resultados de funções de agregação como COUNT(), SUM(), AVG(), MAX() ou MIN().

</> Exemplos Práticos

-- Exemplo 1: Departamentos com mais de 5 funcionários

```
SELECT DepartamentoID, COUNT(FuncionarioID) AS NumFuncionarios
FROM Funcionarios
GROUP BY DepartamentoID
HAVING COUNT(FuncionarioID) > 5;
```

-- Exemplo 2: Departamentos com média salarial > 5000

```
SELECT DepartamentoID, AVG(Salario) AS MediaSalarial
FROM Funcionarios
GROUP BY DepartamentoID
HAVING AVG(Salario) > 5000;
```

-- Exemplo 3: Combinando WHERE e HAVING

```
SELECT DepartamentoID, AVG(Salario) AS MediaSalarial
FROM Funcionarios
WHERE DataContratacao >= '2020-01-01'
GROUP BY DepartamentoID
HAVING AVG(Salario) > 5000;
```

Processo de Filtragem com WHERE e HAVING



Atividade Prática 1: Agrupamento e Restrição

☰ Tarefas

1 Funções Simples

Criar uma consulta que calcule o salário médio, mínimo e máximo de todos os funcionários.

-- Passo 1: Selecionar a tabela Funcionarios
SELECT * FROM Funcionarios;

-- Passo 2: Adicionar funções de agregação
SELECT
 AVG(Salario) AS SalarioMedio,
 MIN(Salario) AS SalarioMinimo,
 MAX(Salario) AS SalarioMaximo
FROM Funcionarios;

2 Agrupamento

Criar uma consulta que agrupe os resultados por Departamento e calcule a média salarial para cada um.

-- Passo 1: Agrupar por departamento
SELECT
 d.Nome AS Departamento,
 AVG(f.Salario) AS MediaSalarial

☰ Modelo de Dados

Para esta atividade, usaremos as seguintes tabelas:

Funcionarios			
FuncionarioID	Nome	DepartamentoID	Salario
1	Ana Silva	1	4500.00
2	Carlos Santos	2	5200.00
3	Mariana Oliveira	1	6800.00

Departamentos

DepartamentoID Nome

1	TI
2	RH
3	Financeiro

Conversão de Tipos e Ordenação

➡ Conversão de Tipos de Dados

A conversão de tipos é essencial quando precisamos realizar operações entre dados de tipos diferentes ou quando precisamos garantir a consistência dos dados em operações matemáticas ou comparações.

```
-- Conversão explícita com CAST  
SELECT CAST('100' AS INT) AS NumeroConvertido;  
  
-- Conversão explícita com CONVERT (SQL Server)  
SELECT CONVERT(DATE, '2023-10-15') AS DataConvertida;  
  
-- Conversão em operações matemáticas  
SELECT ValorTexto,  
       CAST(ValorTexto AS DECIMAL(10,2)) + 100 AS Resultado  
FROM TabelaExemplo;
```

Cuidado: Conversões implícitas podem ocorrer automaticamente, mas podem causar erros ou resultados inesperados. Sempre prefira conversões explícitas.

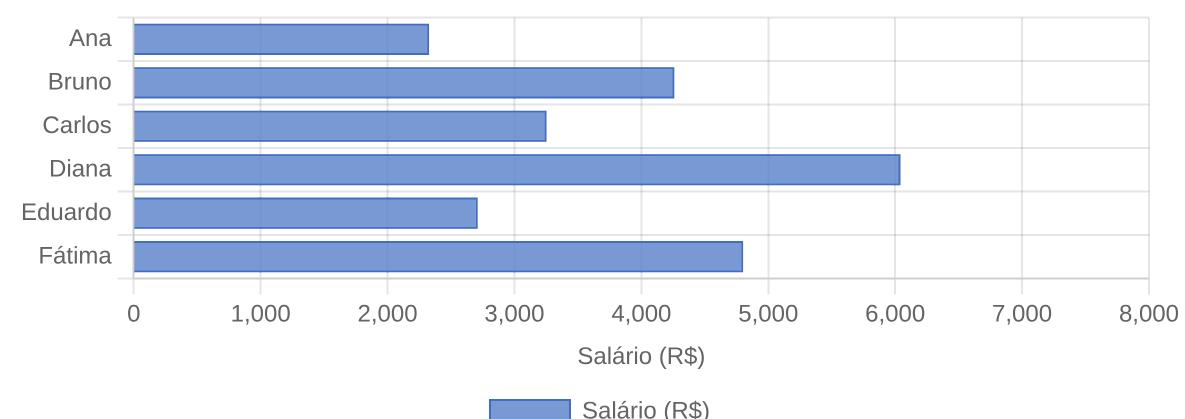
Função	Descrição	Exemplo
CAST()	Padrão SQL	CAST(valor AS tipo)
CONVERT()	SQL Server	CONVERT(tipo, valor)
TO_NUMBER()	Oracle	TO_NUMBER(string)
TO_DATE()	Oracle	TO_DATE(string, formato)

⬇️ Ordenação (ORDER BY)

A cláusula ORDER BY organiza os resultados de uma consulta por uma ou mais colunas, facilitando a análise e visualização dos dados. É uma das operações mais comuns e úteis em SQL.

```
-- Ordenação simples (crescente - padrão)  
SELECT Nome, Salario  
FROM Funcionarios  
ORDER BY Salario;  
  
-- Ordenação decrescente  
SELECT Nome, Salario  
FROM Funcionarios  
ORDER BY Salario DESC;  
  
-- Ordenação por múltiplas colunas  
SELECT DepartamentoID, Nome, Salario  
FROM Funcionarios  
ORDER BY DepartamentoID ASC, Salario DESC;
```

Funcionários ordenados por salário (crescente)



Dica: A ordenação pode ser feita por posição da coluna (ORDER BY 1, 2) ou por expressões calculadas (ORDER BY Salario * 12).

Tratamento de NULL: Em PostgreSQL, use NULLS FIRST ou NULLS LAST para controlar a posição dos valores NULL na ordenação.

Subqueries (Subconsultas)

💡 O que são Subqueries?

Subqueries (ou subconsultas) são consultas aninhadas dentro de outra consulta.

A subquery é executada primeiro e seu resultado é usado como entrada para a consulta externa.

Subquery de Uma Linha

Retorna apenas um valor. Usada com operadores de comparação simples (=, >, <).

Subquery de Múltiplas Linhas

Retorna uma lista de valores. Usada com operadores IN, ANY, ALL.

</> Exemplos Básicos

-- Subquery de uma linha na cláusula WHERE

```
SELECT Nome, Salario  
FROM Funcionarios  
WHERE Salario > (SELECT AVG(Salario) FROM Funcionarios);
```

-- Subquery de múltiplas linhas com IN

```
SELECT Nome  
FROM Funcionarios  
WHERE DepartamentoID IN (  
    SELECT DepartamentoID  
    FROM Departamentos  
    WHERE Nome LIKE '%TI%'  
)
```

💡 Subqueries Avançadas

-- Subquery correlacionada

```
SELECT d.Nome AS Departamento, f.Nome AS Funcionario  
FROM Departamentos d  
JOIN Funcionarios f ON d.DepartamentoID = f.DepartamentoID  
WHERE f.Salario = (  
    SELECT MAX(Salario)  
    FROM Funcionarios f2  
    WHERE f2.DepartamentoID = d.DepartamentoID  
);
```

Consulta Externa (Outer Query)

```
SELECT Nome, Salario
```

```
FROM Funcionarios
```

```
WHERE Salario >
```

```
    SELECT AVG(Salario) FROM Funcionarios
```

💡 **Dica:** Subqueries podem ser usadas em cláusulas SELECT, FROM, WHERE e HAVING.

⚠️ **Cuidado:** Subqueries correlacionadas podem impactar o desempenho.

Atividade Prática 2: Subqueries e Ordenação

☰ Tarefas

1 Subquery de Uma Linha

Criar uma consulta que liste produtos com preço superior à média.

```
SELECT ProdutoID, Nome, Preco  
FROM Produtos  
WHERE Preco > (SELECT AVG(Preco) FROM Produtos)  
ORDER BY Preco;
```

2 Subquery de Múltiplas Linhas

Listar funcionários de departamentos específicos usando IN.

```
SELECT FuncionarioID, Nome, Cargo  
FROM Funcionarios  
WHERE DepartamentoID IN (  
    SELECT DepartamentoID  
    FROM Departamentos  
    WHERE Nome LIKE '%TI%'  
)
```

3 Ordenação Complexa

Ordenar por múltiplas colunas para melhor organização.

```
SELECT FuncionarioID, Nome, Sobrenome, Cargo  
FROM Funcionarios  
ORDER BY Sobrenome ASC, Nome ASC;
```

ProdutoID	Nome	Preco	Acima da Média?
1	Notebook	3500.00	Sim
2	Mouse	89.90	Não
3	Monitor	1200.00	Sim

Passo a Passo para Resolução

- 1 Identifique o tipo de subquery necessária para a tarefa.
- 2 Desenvolva a subquery interna e teste-a separadamente.
- 3 Integre a subquery na consulta principal com o operador adequado.
- 4 Adicione ordenação para melhorar a legibilidade dos resultados.

Pesquisa Avançada (LIKE e CASE)

Q Operador LIKE

O operador **LIKE** é usado para buscar padrões em texto, permitindo encontrar registros que correspondam a um padrão específico.

%

Representa qualquer sequência de caracteres (incluindo nenhum).

```
-- Nomes que começam com 'A'  
SELECT Nome  
FROM Funcionarios  
WHERE Nome LIKE 'A%';
```

```
-- Nomes que terminam com 'o'  
SELECT Nome  
FROM Funcionarios  
WHERE Nome LIKE '%o';
```

—

Representa exatamente um caractere.

Padrão	Descrição	Exemplo
'A%'	Começa com 'A'	Ana, André
'%a'	Termina com 'a'	Ana, Carla
'%ar%'	Contém 'ar'	Carlos, Maria

♂ Expressão CASE

A expressão **CASE** implementa lógica condicional em SQL, funcionando como uma estrutura if-then-else. É útil para criar colunas calculadas com base em condições.

```
-- Classificação de salários  
SELECT Nome, Salario,  
CASE  
    WHEN Salario < 3000 THEN 'Baixo'  
    WHEN Salario BETWEEN 3000 AND 6000 THEN 'Médio'  
    WHEN Salario > 6000 THEN 'Alto'  
END AS FaixaSalarial  
FROM Funcionarios;
```

```
-- CASE com agregação  
SELECT  
CASE  
    WHEN Salario < 3000 THEN 'Baixo'  
    WHEN Salario BETWEEN 3000 AND 6000 THEN 'Médio'  
    ELSE 'Alto'  
END AS FaixaSalarial,  
COUNT(*) AS NumeroFuncionarios  
FROM Funcionarios  
GROUP BY FaixaSalarial;
```

Objetos Avançados e Segurança

⚙️ Procedimentos Armazenados

Procedimentos Armazenados (Stored Procedures) são blocos de código SQL pré-compilados que podem ser reutilizados, melhorando a performance e a segurança do banco de dados.

⌚ Desempenho

Pré-compilados e otimizados, reduzindo o tráfego de rede e melhorando o tempo de resposta.

leftrightarrow Modularidade

Encapsula lógica complexa, facilitando manutenção e reutilização de código.

```
-- Criação de um procedimento armazenado
DELIMITER //
CREATE PROCEDURE GetFuncionariosPorDepartamento(IN p_DepartamentoID INT)
BEGIN
    SELECT Nome, Cargo, Salario
    FROM Funcionarios
    WHERE DepartamentoID = p_DepartamentoID
    ORDER BY Nome;
END //
DELIMITER ;

-- Chamada do procedimento
CALL GetFuncionariosPorDepartamento(2);
```

🛡️ Segurança

Controle de acesso granular, evitando acesso direto às tabelas e prevenindo injeção de SQL.

leftrightarrow Transações

Gerencia transações complexas, garantindo consistência dos dados.

🔒 Recursos de Segurança

A segurança em bancos de dados é fundamental para proteger informações sensíveis e garantir a integridade dos dados. Implementar controles de acesso adequados é essencial para qualquer sistema de banco de dados.

Recurso	Descrição	Exemplo
Usuários e Roles	Controle de acesso baseado em funções	CREATE USER, GRANT
Permissões	Controle granular de operações	SELECT, INSERT, UPDATE
Criptografia	Proteção de dados sensíveis	ENCRYPT, SSL
Auditoria	Registro de atividades	Logs, Triggers

-- Criação de usuário com permissões restritas

```
CREATE USER 'analista'@'localhost' IDENTIFIED BY 'senha_segura';
```

-- Conceder permissões de leitura apenas

```
GRANT SELECT ON empresa.* TO 'analista'@'localhost';
```

-- Revogar permissões sensíveis

```
REVOKE DROP, TRUNCATE ON empresa.* FROM 'analista'@'localhost';
```

Encerramento e Entrega Final

☰ Requisitos para Entrega

- Modelo Conceitual** completo do banco de dados, incluindo entidades, relacionamentos e cardinalidades.
- Implementação do Banco de Dados** com todas as tabelas, relacionamentos, chaves primárias e estrangeiras.
- Recursos de Segurança** implementados, incluindo usuários, permissões e controle de acesso.
- Procedimentos Armazenados** para automatizar tarefas complexas e garantir a integridade dos dados.
- Consultas Complexas** utilizando Joins, Subqueries, funções de grupo e cláusulas de agrupamento.

↖ Critérios de Avaliação

Critério	Peso	Descrição
Modelagem	25%	Qualidade e correção do modelo conceitual
Implementação	25%	Correta implementação das tabelas e relacionamentos
Consultas	20%	Complexidade e eficiência das consultas SQL

Encerramento da Unidade Curricular

Chegamos ao final da nossa jornada de aprendizado em SQL e Banco de Dados. O domínio dessas ferramentas de agregação e consulta é o que define a proficiência em SQL e permite a transição para a análise de dados complexa.

▶ Próximos Passos

- Finalizar a documentação** do projeto, incluindo diagramas, descrições de tabelas e procedimentos.
- Revisar a implementação** para garantir que todos os requisitos foram atendidos e que o banco de dados está funcionando corretamente.
- Preparar a apresentação** do projeto final, destacando os principais aspectos do modelo, implementação e consultas.
- Aplicar o conhecimento adquirido** em projetos futuros e continuar aprofundando os estudos em análise de dados.