PROGRAMAÇÃO DE BANCO DE DADOS

Prof. Rafael Tápia

2º BIMESTRE

- 5 AULAS - 17/04, 24/04, 08/05, 15/05, 22/05
- 1 REVISÃO - 29/05
- 1 PROVA - 05/06

ARQUIVO DA AULA DE HOJE - LINK

Unidade de Ensino 3

U3 - Seção 2

Aula

SELECT BÁSICO E AVANÇADO

EXERCÍCIO

- Crie um banco de dados chamado **db_aula_20250417**.
- Use esse banco de dados para todos os exercícios da aula de hoje

```
CREATE DATABASE db_aula_20250417;
USE db_aula_20250417;
```

BAIXAR O ARQUIVO

Unidade 3 - Seção 2

Arquivo: SCRIPT-1

CONSULTAS *SELECT*

SELECT

Definição: O comando **SELECT** é usado para consultar dados em uma ou mais tabelas de um banco de dados. Ele é a base de qualquer consulta SQL e é amplamente utilizado em praticamente todas as interações com bases de dados relacionais.

Estrutura básica do SELECT:
SELECT: Especifica quais colunas você deseja visualizar.
FROM: Especifica de qual tabela os dados devem ser recuperados.

SELECT

Podemos trazer todas as colunas, apenas algumas, ou ainda determinar apelidos para cada coluna.

Todas as colunas:

```
SELECT *
FROM Clientes;
```

Apenas colunas especificadas:

```
SELECT nome, email
FROM Clientes;
```

Apelidos (alias) para as colunas :

```
SELECT nome AS 'Nome do Cliente', email AS 'E-mail'
FROM Clientes;
```

WHERE

O WHERE é utilizado para filtrar dados com base em uma condição.

```
SELECT nome, email
FROM Clientes
WHERE nome = 'Maria Silva';
```

Isso irá retornar os clientes cujo nome seja exatamente 'Maria Souza'.

WHERE

Existem alguns operadores que podemos utilizar na condição WHERE. Estes são os operadores mais comuns:

- =
- <> ou !=
- > / >=
- < / <=
- LIKE
- IS NULL / IS NOT NULL
- BETWEEN...AND
- IN / NOT IN
- AND / OR / NOT

WHERE - EXEMPLOS

```
SELECT nome, email
FROM Clientes
WHERE email LIKE '%gmail.com';

SELECT nome, email
FROM Clientes
WHERE nome IN ('Maria Silva', 'Rodrigo Silva');

SELECT nome, email, data_nascimento
FROM Clientes
WHERE data_nascimento
                BETWEEN '1988-01-01' AND '1990-05-12';
```

ORDER BY

O ORDER BY é usado para ordenar os resultados da consulta de acordo com uma ou mais colunas.

```
SELECT nome, email FROM Clientes ORDER BY nome ASC;
```

```
SELECT nome, email FROM Clientes ORDER BY nome DESC;
```

```
SELECT nome, email, data_nascimento
FROM Clientes
ORDER BY data_nascimento DESC, nome ASC;
```

LIMIT

O LIMIT é usado para limitar o número de registros retornados pela consulta.

```
SELECT *
FROM Clientes
LIMIT 5;
```

COUNT()

A função COUNT() conta o número de linhas de uma tabela ou o número de valores não nulos de uma coluna específica.

```
SELECT COUNT(*) AS total_clientes
FROM Clientes;

SELECT COUNT(qtde_pontos_fidelidade) AS
total_clientes_com_pontos FROM Clientes;
```

Aqui, COUNT(qtde_pontos_fidelidade) conta as linhas em que o campo qtde_pontos_fidelidade não é nulo, ou seja, clientes que têm algum valor registrado de pontos de fidelidade. Ele conta, Não soma.

SUM()

A função SUM() soma todos os valores numéricos de uma coluna específica.

```
SELECT SUM(qtde_pontos_fidelidade) AS total_pontos
FROM Clientes;
```

AVG()

A função AVG() calcula a média dos valores de uma coluna.

```
SELECT AVG(qtde_pontos_fidelidade) AS media_pontos
FROM Clientes;
```

MAX()

A função MAX() retorna o maior valor presente em uma coluna.

```
SELECT MAX(qtde_pontos_fidelidade) AS maior_qtde
FROM Clientes;
```

MIN()

A função MIN() retorna o menor valor presente em uma coluna.

```
SELECT MIN(qtde_pontos_fidelidade) AS menor_valor
FROM Clientes;
```

SUB-SELECTS

SUB-SELECT

A subconsulta é uma consulta dentro de outra consulta, onde o resultado da subconsulta é utilizado pela consulta externa.

```
SELECT nome, qtde_pontos_fidelidade
FROM Clientes
WHERE qtde_pontos_fidelidade = (
    SELECT MIN(qtde_pontos_fidelidade)
    FROM Clientes
);
```

SUB-SELECT

Também é possível utilizá-lo no FROM.

```
SELECT
    s.estado, AVG(s.qtde_pontos_fidelidade) AS media
FROM
    (
        SELECT estado, qtde_pontos_fidelidade
        FROM Clientes
        WHERE estado = 'SP'
    ) AS s
```

SUB-SELECT

Também pode ser utilizada como um campo.

```
SELECT
   nome,
   data_nascimento,
   (SELECT MIN(data_nascimento) FROM Clientes) as data_menor,
   (SELECT MAX(data_nascimento) FROM Clientes) as data_maior
FROM
   Clientes
```

AGRUPAMENTO
GROUP BY

GROUP BY

A cláusula **GROUP BY** é usada para agrupar registros que compartilham um ou mais valores em colunas especificadas.

Em conjunto com funções agregadas (como COUNT(), SUM(), AVG(), etc.), ela permite que você realize cálculos em cada grupo de registros, ao invés de calcular para toda a tabela.

GROUP BY - EXEMPLO

Vamos agrupar os clientes pela cidade e calcular a média de pontos de fidelidade para cada cidade.

```
SELECT cidade, AVG(qtde_pontos_fidelidade) AS media
FROM Clientes
GROUP BY cidade;
```

HAVING

A cláusula **HAVING** é usada para filtrar os resultados após a aplicação do **GROUP BY**. Enquanto **WHERE** filtra os dados antes de qualquer agregação, **HAVING** filtra os dados após o agrupamento.

Isso significa que **HAVING** permite aplicar condições nas colunas que envolvem funções agregadas, o que não é possível com **WHERE**.

HAVING - EXEMPLO

Vamos calcular a média de pontos de fidelidade por cidade, mas exibir apenas as cidades onde a média de pontos seja superior a 100.

```
SELECT cidade, AVG(qtde_pontos_fidelidade) AS media
FROM Clientes
GROUP BY cidade
HAVING AVG(qtde_pontos_fidelidade) > 100;
```

Diferença entre WHERE e HAVING:

- **WHERE**: Filtra os dados antes de qualquer agregação. Usado para condições que envolvem colunas não agregadas.

- **HAVING**: Filtra os dados após a agregação. Usado para condições envolvendo funções agregadas (como AVG(), SUM(), COUNT(), etc.).

USANDO WHERE E HAVING JUNTOS:

São instruções diferente e posso utilizar os 2 na mesma SELECT para fins diferentes.

```
SELECT cidade, AVG(qtde_pontos_fidelidade) AS media
FROM Clientes
WHERE estado in ( 'SP','RJ','MG')
GROUP BY cidade
HAVING AVG(qtde_pontos_fidelidade) > 100;
```

JOIN

BAIXAR O ARQUIVO

Unidade 3 - Seção 2

Arquivo: SCRIPT-2

JOIN

O **JOIN** é usado para **combinar dados de duas ou mais tabelas** com base em uma condição comum entre elas.

Existem vários tipos de JOIN, como:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

INNER JOIN

O **INNER JOIN** retorna apenas as linhas que têm correspondências em **ambas as tabelas**. Se um registro na tabela da esquerda não tiver correspondência na tabela da direita, ou vice-versa, esse registro será excluído do resultado.

```
SELECT c.nome, p.total_pedido
FROM Clientes c
INNER JOIN Pedidos p ON c.id = p.id_cliente;
```

LEFT JOIN (ou LEFT OUTER JOIN)

O **LEFT JOIN** retorna todas as linhas da tabela à esquerda (neste caso, Clientes), e as linhas correspondentes da tabela à direita (neste caso, Pedidos). **Se não houver correspondência, os campos da tabela à direita terão valor NULL.**

```
SELECT c.nome, p.total_pedido
FROM Clientes c
LEFT JOIN Pedidos p ON c.id = p.id_cliente;
```

Se um cliente não tiver pedidos, o total_pedido será NULL para esse cliente.

RIGHT JOIN (ou RIGHT OUTER JOIN)

O **RIGHT JOIN** funciona de maneira oposta ao LEFT JOIN. Ele retorna todas as linhas da tabela à direita (neste caso, Pedidos), e as linhas correspondentes da tabela à esquerda (neste caso, Clientes). **Se não houver correspondência, os campos da tabela à esquerda terão valor NULL.**

```
SELECT c.nome, p.total_pedido
FROM Clientes c
RIGHT JOIN Pedidos p ON c.id = p.id_cliente;
```

Se um pedido não tiver um cliente correspondente (por exemplo, se o cliente foi excluído da tabela Clientes), o nome do cliente será NULL.

SELECT COMPLETA

```sql
SELECT
    c.id,
    c.nome,
    COUNT(p.id_pedido) AS total_pedidos,
    SUM(p.total_pedido) AS soma_total_pedidos
FROM
    Clientes c
INNER JOIN
    Pedidos p ON c.id = p.id_cliente
WHERE
    c.estado = 'SP'
GROUP BY
    c.id, c.nome
HAVING
    COUNT(p.id_pedido) > 1;
```

EXERCÍCIOS

Exercício 1: Alias de Campos

Crie uma consulta que traga o nome completo do cliente e o total do pedido, mas renomeie a coluna do nome como cliente_nome e a coluna do total do pedido como valor_total.

```
SELECT nome AS cliente_nome, total_pedido AS valor_total
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente;
```

Exercício 2: Operadores no WHERE

Crie uma consulta que traga todos os clientes cujo estado seja "SP" ou "RJ", que não tenham pontos de fidelidade igual a zero e que o valor do pedido seja superior a 100.

```
SELECT nome, email, qtde_pontos_fidelidade, total_pedido
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente
WHERE (estado = 'SP' OR estado = 'RJ')
  AND qtde_pontos_fidelidade != 0
  AND total_pedido > 100;
```

Exercício 3: ORDER BY

Crie uma consulta que traga todos os pedidos e seus respectivos valores, ordenados pelo total_pedido de forma decrescente.

```
SELECT id_pedido, total_pedido
FROM Pedidos
ORDER BY total_pedido DESC;
```

Exercício 4: LIMIT

Crie uma consulta que traga os 5 clientes com o maior número de pontos de fidelidade.

```
SELECT nome, qtde_pontos_fidelidade
FROM Clientes
ORDER BY qtde_pontos_fidelidade DESC
LIMIT 5;
```

Exercício 5: COUNT

Crie uma consulta que traga o número total de clientes que realizaram pedidos.

```
SELECT COUNT(DISTINCT id_cliente) AS numero_de_clientes
FROM Pedidos;
```

Exercício 6: SUM

Crie uma consulta que calcule o total de vendas feitas por todos os clientes.

```
SELECT SUM(total_pedido) AS total_vendas
FROM Pedidos;
```

Exercício 7: AVG

Crie uma consulta que calcule a média do valor dos pedidos realizados pelos clientes.

```
SELECT AVG(total_pedido) AS media_pedidos
FROM Pedidos;
```

Exercício 8: MAX e MIN

Crie uma consulta que traga o valor do maior pedido realizado e o valor do menor pedido realizado.

```
SELECT MAX(total_pedido) AS maior_pedido, MIN(total_pedido) AS
menor_pedido
FROM Pedidos;
```

Exercício 9: GROUP BY

Crie uma consulta que traga o total de vendas por cliente (soma de total_pedido), agrupado pelo id_cliente.

```
SELECT id_cliente, SUM(total_pedido) AS total_vendas
FROM Pedidos
GROUP BY id_cliente;
```

Exercício 10: HAVING

Crie uma consulta que traga o total de vendas por cliente, mas somente para aqueles clientes que têm um total de vendas superior a 500.

```
SELECT id_cliente, SUM(total_pedido) AS total_vendas
FROM Pedidos
GROUP BY id_cliente
HAVING SUM(total_pedido) > 500;
```

Exercício 11: INNER JOIN

Crie uma consulta que traga o nome do cliente e o valor do pedido, unindo as tabelas Clientes e Pedidos pela chave id_cliente.

```
SELECT Clientes.nome, Pedidos.total_pedido
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente;
```

Exercício 12: LEFT JOIN

Crie uma consulta que traga todos os clientes e seus pedidos, mesmo que alguns clientes não tenham feito pedidos. Exiba o nome do cliente e o total do pedido.

```
SELECT Clientes.nome, Pedidos.total_pedido
FROM Clientes
LEFT JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente;
```

Exercício 13: Subconsulta

Crie uma consulta que traga o nome do cliente e o total do pedido, apenas para os clientes cujo total de pedido é maior que o valor do menor pedido realizado.

```
SELECT nome, total_pedido
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente
WHERE total_pedido > (
    SELECT MIN(total_pedido)
    FROM Pedidos
);
```

Exercício 14: Conceitos combinados

Crie uma consulta que traga o nome do cliente, o número de pedidos realizados por ele, a soma total de seus pedidos, e o valor médio de seus pedidos, somente para os clientes que realizaram mais de 1 pedido, ordenados pela soma total de pedidos, em ordem decrescente.

```
SELECT Clientes.nome, COUNT(Pedidos.id_pedido) AS
numero_de_pedidos,
    SUM(Pedidos.total_pedido) AS soma_total_pedidos,
    AVG(Pedidos.total_pedido) AS media_pedidos
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente
GROUP BY Clientes.nome
HAVING COUNT(Pedidos.id_pedido) > 1
ORDER BY soma_total_pedidos DESC;
```

Exercício 15: Conceitos Combinados

Criar uma consulta que retorne os clientes que realizaram mais de 1 pedido, junto com o número de pedidos e a soma total dos pedidos. Os resultados devem ser ordenados pela soma total dos pedidos, em ordem decrescente. Além disso, exibir apenas os clientes cujo total de pedidos seja superior a 300.

```
SELECT Clientes.nome,
    COUNT(Pedidos.id_pedido) AS numero_de_pedidos,
    SUM(Pedidos.total_pedido) AS soma_total_pedidos
FROM Clientes
INNER JOIN Pedidos ON Clientes.ID = Pedidos.id_cliente
GROUP BY Clientes.nome
HAVING COUNT(Pedidos.id_pedido) > 1
   AND SUM(Pedidos.total_pedido) > 300
ORDER BY soma_total_pedidos DESC;
```

EXERCÍCIOS PARA NOTA

EXERCÍCIO

Usando as mesmas tabelas da aula, criar consultas que retornem apenas os clientes que não possuem nenhum pedido.

Crie **3 consultas diferentes** que retornem o mesmo resultado.

PONTUAÇÃO: 500 pontos
ENVIAR POR EMAIL: rafael.h.tapia@cogna.com.br
ASSUNTO: EXERCÍCIO 3 CONSULTAS - <nome_do_aluno>