Consultas SQL Intermediárias

Aprofundando seus conhecimentos!

SELECT Avançado com WHERE

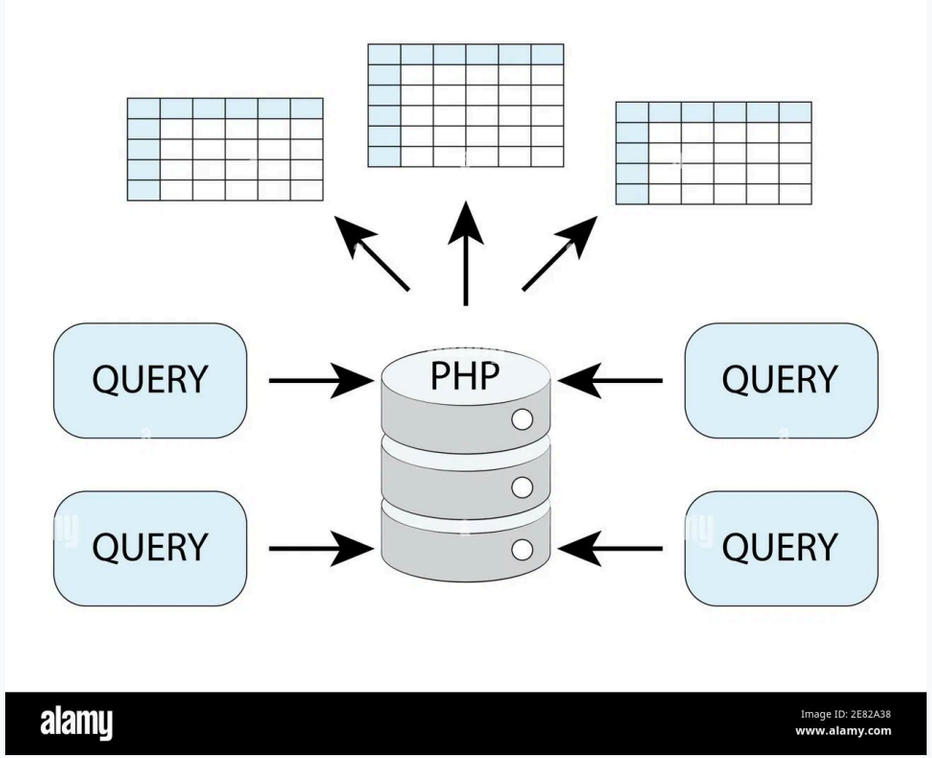
Agora que você já conhece o SELECT básico, vamos aprender a fazer consultas mais específicas usando a cláusula **WHERE** com diferentes operadores de comparação.

EXEMPLO COM MÚLTIPLAS CONDIÇÕES:

```
SELECT nome, idade, cidade
FROM alunos
WHERE idade >= 18 AND cidade = 'São Paulo';
```

OPERADORES DE COMPARAÇÃO:

`=`	Igual a
`!= ou <>`	Diferente de
`> < >= <=`	Maior, Menor, Maior/Menor ou igual
`BETWEEN`	Entre dois valores
`LIKE`	Padrão de texto
`IN`	Lista de valores



Operadores Lógicos: AND, OR, NOT

Operadores lógicos permitem combinar múltiplas condições em uma consulta SQL, tornando seus filtros mais poderosos e precisos.

AND

Retorna registros quando **TODAS** as condições são verdadeiras.

```
SELECT * FROM alunos
WHERE idade >= 18
AND cidade = 'São Paulo';
```

OR

Retorna registros quando **PELO MENOS UMA** condição é verdadeira.

```
SELECT * FROM alunos
WHERE cidade = 'Rio'
OR cidade = 'São Paulo';
```

NOT

Inverte o resultado, retornando o **OPOSTO** da condição.

```
SELECT * FROM alunos
WHERE NOT cidade = 'Brasília';
```

COMBINADO

Você pode combinar operadores usando **parênteses** para controlar a ordem.

```
SELECT * FROM alunos
WHERE (idade >= 18 AND cidade = 'Rio')
OR (idade < 18 AND cidade = 'SP');
```

ORDER BY – Ordenação Avançada

JOINs – Tipos e Aplicações

INNER JOIN

```
SELECT * FROM tabela1 INNER JOIN tabela2 ON tabela1.id = tabela2.id;
```

Retorna apenas os registros que têm correspondência em **ambas** as tabelas.

LEFT JOIN

```
SELECT * FROM tabela1 LEFT JOIN tabela2 ON tabela1.id = tabela2.id;
```

Retorna **todos** os registros da tabela da esquerda e os correspondentes da direita. Se não houver correspondência, retorna NULL.

RIGHT JOIN

```
SELECT * FROM tabela1 RIGHT JOIN tabela2 ON tabela1.id = tabela2.id;
```

Retorna **todos** os registros da tabela da direita e os correspondentes da esquerda. Se não houver correspondência, retorna NULL.

JOINs na Prática

Tabela: ALUNOS

id_aluno	nome
1	Ana
2	João
3	Maria

+

Tabela: NOTAS

id_aluno	nota
1	8.5
2	7.0
3	9.5

COMANDO INNER JOIN:

```
SELECT alunos.nome, notas.nota
FROM alunos
INNER JOIN notas ON alunos.id_aluno = notas.id_aluno;
```

RESULTADO DA CONSULTA:

nome	nota
Ana	8.5
João	7.0
Maria	9.5

Funções de Agregação

Funções de Agregação realizam cálculos em um conjunto de valores e retornam um único resultado. São essenciais para análise de dados e geração de relatórios.

COUNT()
Conta o número de registros ou valores não nulos

SUM()
Soma todos os valores numéricos de uma coluna

AVG()
Calcula a média aritmética dos valores

MIN()
Retorna o menor valor encontrado

MAX()
Retorna o maior valor encontrado

EXEMPLOS PRÁTICOS:

```
SELECT COUNT(*) FROM alunos;
```

→ Retorna o total de alunos cadastrados

```
SELECT AVG(nota) FROM provas
WHERE disciplina = 'SQL';
```

→ Calcula a nota média nas provas de SQL

```
SELECT MIN(idade), MAX(idade)
FROM alunos;
```

→ Mostra a menor e maior idade dos alunos

```
SELECT SUM(salario) FROM funcionarios
WHERE departamento = 'TI';
```

→ Soma todos os salários do departamento de TI

GROUP BY e HAVING

GROUP BY

```
SELECT cidade, COUNT(*) as total
FROM alunos
GROUP BY cidade;
```

Agrupa registros com valores iguais em uma ou mais colunas. Usado com funções de agregação (COUNT, SUM, AVG, etc).

HAVING

```
SELECT cidade, COUNT(*) as total
FROM alunos
GROUP BY cidade
HAVING COUNT(*) > 2;
```

Filtra os grupos criados pelo GROUP BY. É como o WHERE, mas funciona depois do agrupamento.

WHERE vs HAVING

WHERE

Filtra linhas ANTES do agrupamento

HAVING

Filtra grupos DEPOIS do agrupamento

DADOS ORIGINAIS (Tabela ALUNOS)

Nome	Cidade
Ana	São Paulo
João	Rio
Maria	São Paulo
Pedro	Brasília
Lucas	São Paulo
Julia	Rio

RESULTADO (GROUP BY cidade HAVING COUNT(*) > 2)

Cidade	Total de Alunos
São Paulo	**3**

Consulta Completa - Juntando Tudo

Veja como combinar **SELECT, WHERE, JOIN, GROUP BY, HAVING e ORDER BY** em uma única consulta!

CONSULTA SQL COMPLETA:

```sql
SELECT alunos.cidade, COUNT(*) as total_alunos,
       AVG(notas.nota) as media_nota
FROM alunos
INNER JOIN notas
    ON alunos.id_aluno = notas.id_aluno
WHERE alunos.idade >= 18
    AND notas.disciplina = 'SQL'
GROUP BY alunos.cidade
HAVING AVG(notas.nota) >= 7.0
ORDER BY media_nota DESC;
```

1. **SELECT e Funções**
   Seleciona cidade, conta alunos e calcula média das notas
2. **INNER JOIN**
   Une as tabelas alunos e notas pelo id_aluno
3. **WHERE**
   Filtra apenas alunos maiores de 18 anos na disciplina SQL
4. **GROUP BY**
   Agrupa os resultados por cidade
5. **HAVING**
   Filtra apenas cidades com média >= 7.0
6. **ORDER BY**
   Ordena pela média das notas (maior para menor)

Exercícios Práticos

Teste seus conhecimentos com estes exercícios práticos!

1 Filtragem Básica

Você tem uma tabela **produtos** com colunas: nome, preco, categoria.

> *Escreva uma consulta que retorne todos os produtos da categoria 'Eletrônicos' com preço menor que R$ 500.*

2 Operadores Lógicos

Tabela **funcionarios**: nome, salario, departamento, idade.

> *Selecione funcionários do departamento 'TI' OU 'Marketing' que ganham mais de R$ 5.000.*

3 JOIN e Agregação

Tabelas: **clientes** (id, nome, cidade) e **pedidos** (id, id_cliente, valor).

> *Calcule o valor total de pedidos por cidade usando JOIN e SUM.*

4 GROUP BY e HAVING

Tabela **vendas**: vendedor, produto, quantidade, valor.

> *Liste vendedores que venderam mais de 10 produtos no total, ordenados por quantidade vendida (DESC).*

DESAFIO EXTRA!

Combine SELECT, JOIN, WHERE, GROUP BY, HAVING e ORDER BY em uma única consulta para resolver um problema real do seu dia a dia. Compartilhe com seus colegas!

Continue Praticando!

Você aprendeu os conceitos intermediários de consultas SQL. Agora é hora de aplicar esse conhecimento em projetos reais e continuar evoluindo!

Pratique Diariamente

Explore Dados Reais

Evolua Seus Conhecimentos