Aula 7 – Consultas SQL Intermediárias para Negócios

No dinâmico universo da análise de dados, ter acesso a informações é apenas o primeiro passo. O verdadeiro poder reside na capacidade de transformar um mar de dados brutos em insights acionáveis, que podem guiar decisões estratégicas e impulsionar o crescimento de um negócio. É aqui que o SQL, a Linguagem de Consulta Estruturada, se revela uma ferramenta indispensável, atuando como a espinha dorsal para qualquer profissional que deseje ir além do básico.

Se você já se aventurou pelas consultas SQL mais simples, como selecionar dados ou filtrá-los com WHERE, prepare-se para elevar o nível. Esta aula foi desenhada para expandir seu horizonte, permitindo que você manipule e agregue informações de maneiras muito mais sofisticadas. Imagine poder não apenas ver as vendas de um produto, mas entender o faturamento médio por região, identificar os produtos mais lucrativos ou até mesmo cruzar dados de clientes e pedidos para descobrir padrões de compra.

Ao longo desta jornada, você desenvolverá as habilidades necessárias para extrair valor máximo de seus bancos de dados. Nosso objetivo é que, ao final, você seja capaz de utilizar funções de agregação para resumir grandes volumes de dados, agrupar informações para análises segmentadas, filtrar esses grupos com critérios específicos, e unir dados de diferentes tabelas para construir uma visão completa. Além disso, exploraremos as subconsultas, uma técnica poderosa para resolver problemas complexos com elegância e eficiência. Prepare-se para desvendar os segredos que os dados guardam e transformá-los em inteligência para os negócios.

Funções de Agregação: O Poder dos Resumos Inteligentes

Imagine que você é o gerente de uma grande loja de varejo e precisa entender o desempenho das vendas do último trimestre. Olhar para cada venda individualmente seria como tentar beber água de uma mangueira de incêndio: muita informação, pouco discernimento. O que você realmente precisa são resumos, métricas que condensem essa vasta quantidade de dados em números significativos.

É exatamente para isso que servem as funções de agregação no SQL. Elas são como um assistente inteligente que pega uma pilha de documentos e te entrega apenas os totais, as médias, os maiores e menores valores, ou a contagem de itens. Em vez de se perder nos detalhes, você obtém uma visão panorâmica que permite identificar tendências e tomar decisões rápidas. Elas são a base para qualquer painel de Business Intelligence, transformando dados brutos em KPIs (Key Performance Indicators) essenciais.

Vamos explorar as principais ferramentas desse kit de resumo: COUNT, SUM, AVG, MIN e MAX. Cada uma delas tem um papel crucial na hora de transformar um conjunto de dados em uma informação valiosa para o seu negócio. Elas são a porta de entrada para a Data Literacy, permitindo que você "leia" os dados de forma mais eficaz.

Contando, Somando e Calculando Médias

COUNT()

Sua aliada quando você precisa saber quantos registros existem em uma tabela ou quantos registros atendem a uma condição específica. Pense nela como um contador de itens: quantos clientes você tem? Quantos pedidos foram feitos no mês passado?

SUM()

Perfeita para totalizar valores numéricos. Se você precisa saber o faturamento total, o custo total de estoque ou a soma de todas as despesas, SUM() faz o trabalho pesado.

AVG()

Calcula a média aritmética de uma coluna numérica. É ideal para entender o comportamento médio, como o valor médio de um pedido, a idade média dos clientes ou a duração média de um projeto.

Exemplos Práticos

```
-- Contar o número total de produtos cadastrados
SELECT COUNT(id_produto) FROM Produtos;

-- Contar o número de vendas realizadas
SELECT COUNT(*) FROM Vendas;
```

```
-- Somar o valor total de todas as vendas
SELECT SUM(valor_total) FROM Vendas;

-- Somar a quantidade de itens em estoque
SELECT SUM(quantidade_estoque) FROM Produtos;
```

```
-- Calcular o valor médio de uma venda
SELECT AVG(valor_total) FROM Vendas;

-- Calcular a avaliação média de um produto
SELECT AVG(avaliacao_estrelas) FROM AvaliacoesProdutos;
```

Essas funções são a base para construir relatórios gerenciais e dashboards que oferecem uma visão clara e concisa do desempenho do seu negócio, permitindo que você identifique rapidamente o que está funcionando e o que precisa de atenção.

Encontrando Extremos: Mínimo e Máximo

Continuando nossa exploração das funções de agregação, MIN() e MAX() são essenciais para identificar os limites de um conjunto de dados. Elas atuam como um radar, apontando para os pontos mais baixos e mais altos, o que é fundamental para análises de desempenho, precificação ou identificação de anomalias.

Imagine que você está analisando o tempo de entrega de seus produtos. Saber o tempo médio é útil, mas conhecer o menor tempo de entrega e o maior tempo de entrega pode revelar gargalos ou, ao contrário, processos de excelência. Essas funções nos dão a capacidade de ver os extremos, os "melhores" e "piores" cenários dentro de um conjunto de dados.

Elas são particularmente úteis em cenários de controle de qualidade, otimização de processos ou até mesmo para definir faixas de valores aceitáveis. Por exemplo, qual foi o preço mais baixo que um produto já foi vendido? E o mais alto? Essas informações podem ser cruciais para estratégias de precificação e promoções.

MIN() e MAX() em Ação

MIN()

Retorna o menor valor de uma coluna. Pode ser o menor preço, a data mais antiga, a menor quantidade em estoque, ou a menor avaliação.

```
-- Encontrar o menor valor de venda registrado
SELECT MIN(valor_total) FROM Vendas;

-- Encontrar a data da primeira venda
SELECT MIN(data_venda) FROM Vendas;
```

MAX()

Retorna o maior valor de uma coluna. Ela é usada para identificar o maior preço, a data mais recente, a maior quantidade em estoque, ou a maior avaliação.

```
-- Encontrar o maior valor de venda registrado
SELECT MAX(valor_total) FROM Vendas;

-- Encontrar a data da última venda
SELECT MAX(data_venda) FROM Vendas;
```

Ao combinar MIN() e MAX() com outras funções de agregação, você constrói um panorama completo do comportamento dos seus dados. Por exemplo, você pode querer saber o valor médio de venda, mas também o mínimo e o máximo para entender a dispersão dos valores. Essa visão detalhada é um pilar da Data Literacy, permitindo uma compreensão profunda dos fenômenos que os dados representam.

Agrupando Dados com GROUP BY: Organizando o Caos em Categorias

Até agora, vimos como resumir dados de forma geral. Mas e se você precisar de resumos para categorias específicas? Por exemplo, qual o faturamento total *por região*? Ou a média de vendas *por tipo de produto*? Olhar o total geral é bom, mas a inteligência de negócios muitas vezes reside na capacidade de segmentar e comparar.

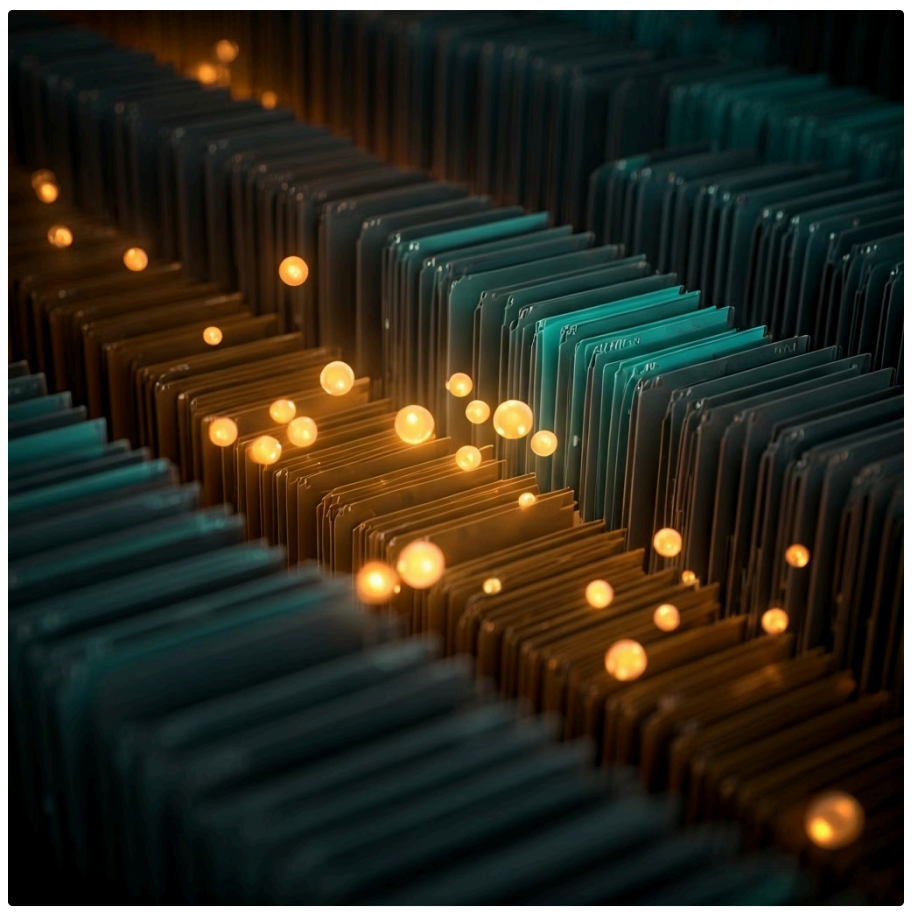
É aqui que a cláusula GROUP BY entra em cena, atuando como um organizador de dados. Pense nela como um sistema de classificação: você tem uma pilha de documentos misturados e decide agrupá-los por assunto, por data ou por autor. O GROUP BY faz exatamente isso com suas linhas de dados, reunindo todas as linhas que possuem o mesmo valor em uma ou mais colunas especificadas.

Essa capacidade de segmentação é crucial para análises mais profundas. Sem o GROUP BY, você estaria limitado a ver apenas o "grande total", perdendo a oportunidade de identificar quais regiões estão performando melhor, quais produtos são mais populares ou quais vendedores estão batendo suas metas. É a ferramenta que transforma um conjunto de dados homogêneo em uma série de insights categorizados.

Como o GROUP BY Funciona

A cláusula GROUP BY é utilizada em conjunto com as funções de agregação. Ela instrui o SQL a aplicar a função de agregação (como SUM, COUNT, AVG) a cada grupo de linhas, em vez de aplicar ao conjunto total de dados.

Vamos a um exemplo prático. Suponha que você queira saber o total de vendas por cada vendedor. Sem GROUP BY, você obteria o total de vendas de todos os vendedores juntos. Com GROUP BY, você terá uma linha para cada vendedor, mostrando o total de vendas *daquele* vendedor.



Exemplos de GROUP BY

```
-- Calcular o faturamento total por região
SELECT regiao, SUM(valor_venda) AS FaturamentoTotal
FROM Vendas
GROUP BY regiao;

-- Contar o número de produtos por categoria
SELECT categoria, COUNT(id_produto) AS TotalProdutos
FROM Produtos
GROUP BY categoria;
```

> **Regra de Ouro:** É importante notar que qualquer coluna que você inclua no SELECT e que não seja uma função de agregação, *deve* ser incluída na cláusula GROUP BY. Isso garante que o resultado faça sentido, mostrando o valor agregado para cada combinação única das colunas agrupadas. Essa é uma regra fundamental para evitar erros e garantir a consistência dos seus relatórios.

Agrupando por Múltiplas Colunas

Aprofundando no uso do GROUP BY, é fundamental entender suas regras e como ele pode ser aplicado para análises ainda mais granulares. A flexibilidade dessa cláusula permite que você crie agrupamentos complexos, revelando padrões que seriam invisíveis em uma análise superficial.

Uma regra de ouro ao usar GROUP BY é que todas as colunas listadas no SELECT que não são funções de agregação devem, obrigatoriamente, aparecer na cláusula GROUP BY. Se você tentar selecionar uma coluna que não está agregada nem agrupada, o SQL não saberá como exibi-la, pois haveria múltiplos valores para essa coluna dentro de cada grupo.

Além disso, o poder do GROUP BY se expande quando você agrupa por múltiplas colunas. Isso permite criar subcategorias dentro de categorias, oferecendo uma visão ainda mais detalhada dos seus dados. Pense em como você organiza seus arquivos: primeiro por ano, depois por mês, e talvez por tipo de documento. O SQL pode fazer o mesmo com seus dados.

01

Agrupamento Simples

Uma única coluna define os grupos

02

Agrupamento Múltiplo

Combinações únicas de múltiplas colunas criam subgrupos

03

Análise Detalhada

Insights granulares por categoria e subcategoria

```
-- Calcular o faturamento total por região e por categoria de produto
SELECT regiao, categoria_produto, SUM(valor_venda) AS FaturamentoPorCategoria
FROM Vendas
GROUP BY regiao, categoria_produto;

-- Contar o número de clientes por cidade e estado
SELECT estado, cidade, COUNT(id_cliente) AS TotalClientes
FROM Clientes
GROUP BY estado, cidade;
```

Essa capacidade de detalhamento é extremamente valiosa para identificar nichos de mercado, entender variações regionais no desempenho de produtos ou analisar a eficácia de campanhas de marketing em diferentes segmentos. É uma ferramenta poderosa para aprofundar sua Data Literacy, permitindo que você desvende as nuances por trás dos números.

Uma observação importante: a cláusula WHERE é usada para filtrar linhas *antes* do agrupamento. Ou seja, ela seleciona quais linhas farão parte do processo de agregação e agrupamento. Mas e se você precisar filtrar os *grupos* resultantes da agregação? Por exemplo, se você quiser ver apenas as regiões que faturaram mais de um determinado valor? Para isso, temos a cláusula HAVING.

Filtrando Grupos com a Cláusula HAVING: Refinando Seus Insights

Você já aprendeu a agrupar seus dados e a aplicar funções de agregação para obter resumos por categoria. Agora, imagine que você tem uma lista de faturamento por região, mas só está interessado nas regiões que geraram um faturamento acima de um certo limiar. A cláusula WHERE, que usamos para filtrar linhas individuais, não pode ser aplicada aqui, pois ela opera *antes* do agrupamento.

É nesse cenário que a cláusula HAVING se torna indispensável. Pense nela como um filtro de segunda camada, que atua *depois* que os grupos foram formados e as agregações calculadas. Se o WHERE decide quais ingredientes vão para a panela, o HAVING decide quais pratos prontos serão servidos, com base em critérios sobre o prato em si (como o tamanho da porção ou o custo total).

Dominar o HAVING é um passo crucial para análises mais sofisticadas, permitindo que você refine seus relatórios e se concentre nos grupos de dados que realmente importam para sua tomada de decisão. É a ferramenta que permite ir além do "o que aconteceu" para o "o que é relevante".

HAVING em Ação: Filtrando Resultados Agregados

A cláusula HAVING é sempre utilizada após a cláusula GROUP BY e antes de ORDER BY. Ela permite aplicar condições a resultados de funções de agregação.

Exemplo 1: Categorias com Muitos Produtos

Mostrar apenas as categorias com mais de 100 produtos cadastrados:

```
-- Mostrar categorias com mais de 100 produtos
SELECT categoria, COUNT(id_produto) AS TotalProdutos
FROM Produtos
GROUP BY categoria
HAVING COUNT(id_produto) > 100;
```

Exemplo 2: Regiões com Alto Faturamento Médio

Identificar as regiões onde o faturamento médio por venda foi superior a R$ 500:

```
-- Identificar regiões com faturamento médio por venda acima de R$ 500
SELECT regiao, AVG(valor_venda) AS MediaVenda
FROM Vendas
GROUP BY regiao
HAVING AVG(valor_venda) > 500;
```

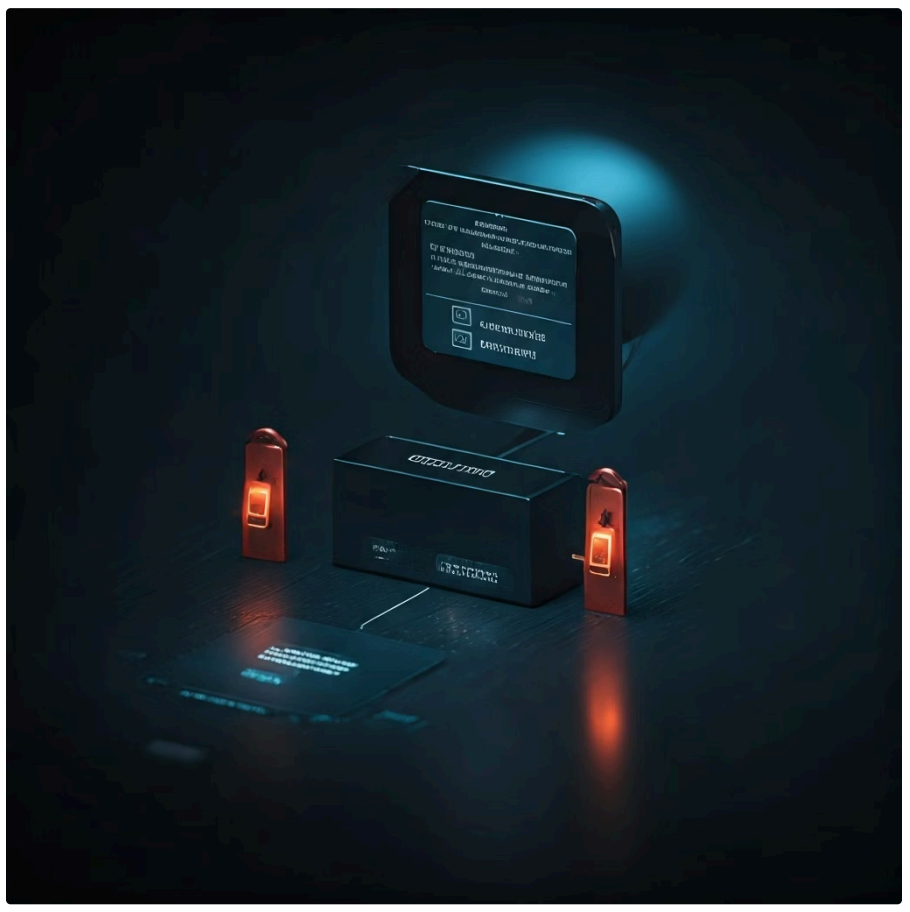
Distinção Crucial: A distinção entre WHERE e HAVING é fundamental. WHERE filtra linhas individuais *antes* do agrupamento, enquanto HAVING filtra grupos *após* o agrupamento e a aplicação das funções de agregação.

WHERE vs. HAVING: Entendendo a Diferença Crucial

A confusão entre WHERE e HAVING é comum, mas entender a diferença é vital para escrever consultas SQL eficientes e corretas. Ambas são cláusulas de filtragem, mas operam em estágios diferentes do processamento da consulta, como se fossem dois portões de segurança em um processo.

Pense no fluxo de dados em uma consulta SQL. Primeiro, os dados são selecionados da tabela (FROM). Em seguida, o WHERE entra em ação, eliminando linhas que não atendem a uma condição específica *antes* que qualquer agrupamento ou agregação ocorra. É como remover os ingredientes estragados antes de começar a cozinhar.

Somente depois que as linhas foram filtradas pelo WHERE é que o SQL agrupa os dados (GROUP BY) e calcula as funções de agregação (SUM, COUNT, AVG, etc.). É nesse ponto que o HAVING assume o controle, filtrando os *grupos* que foram formados, com base em condições aplicadas aos resultados das funções de agregação. É como decidir quais pratos prontos (grupos) serão servidos, com base em seu peso ou custo total (agregações).



Quadro Comparativo: WHERE vs. HAVING

Característica	WHERE	HAVING
Filtra	Linhas individuais	Grupos de linhas
Quando age	Antes do GROUP BY e funções de agregação	Depois do GROUP BY e funções de agregação
Usa	Colunas da tabela original	Colunas da tabela original OU resultados de funções de agregação
Exemplo	WHERE valor_venda > 100	HAVING SUM(valor_venda) > 1000

Ordem das Cláusulas em uma Consulta SQL

A ordem das cláusulas em uma consulta SQL é crucial e segue uma lógica interna de processamento:

1. **FROM:** Define as tabelas de origem.
2. **WHERE:** Filtra linhas individuais.
3. **GROUP BY:** Agrupa as linhas restantes.
4. **HAVING:** Filtra os grupos resultantes.
5. **SELECT:** Seleciona as colunas a serem exibidas.
6. **ORDER BY:** Ordena o resultado final.

Compreender essa sequência e a função específica de cada cláusula é um pilar para a construção de consultas SQL eficientes e para a obtenção de insights precisos, um elemento chave para a Data Literacy e para o uso de ferramentas de BI como o Power BI.

Junção de Tabelas: INNER JOIN – Conectando Pontos Essenciais

No mundo real, os dados raramente residem em uma única tabela monolítica. Para manter a organização e evitar redundância, os bancos de dados são geralmente estruturados com múltiplas tabelas, cada uma focada em um tipo específico de informação (clientes, produtos, vendas, etc.). O desafio surge quando você precisa combinar informações de duas ou mais dessas tabelas para obter uma visão completa.

Imagine que você tem uma tabela de clientes com nomes e endereços, e outra tabela de vendas com os IDs dos clientes e os valores das compras. Para saber o nome do cliente que fez uma determinada compra, você precisa "juntar" essas duas tabelas. É aqui que as cláusulas JOIN entram em jogo, atuando como pontes que conectam informações relacionadas.

O INNER JOIN é o tipo mais comum e fundamental de junção. Pense nele como um casamento perfeito: ele só retorna as linhas onde há uma correspondência exata em *ambas* as tabelas. Se um cliente não tem vendas, ou se uma venda não tem um cliente correspondente, essas informações não aparecerão no resultado. Ele garante que você veja apenas os dados que se "encaixam" perfeitamente.

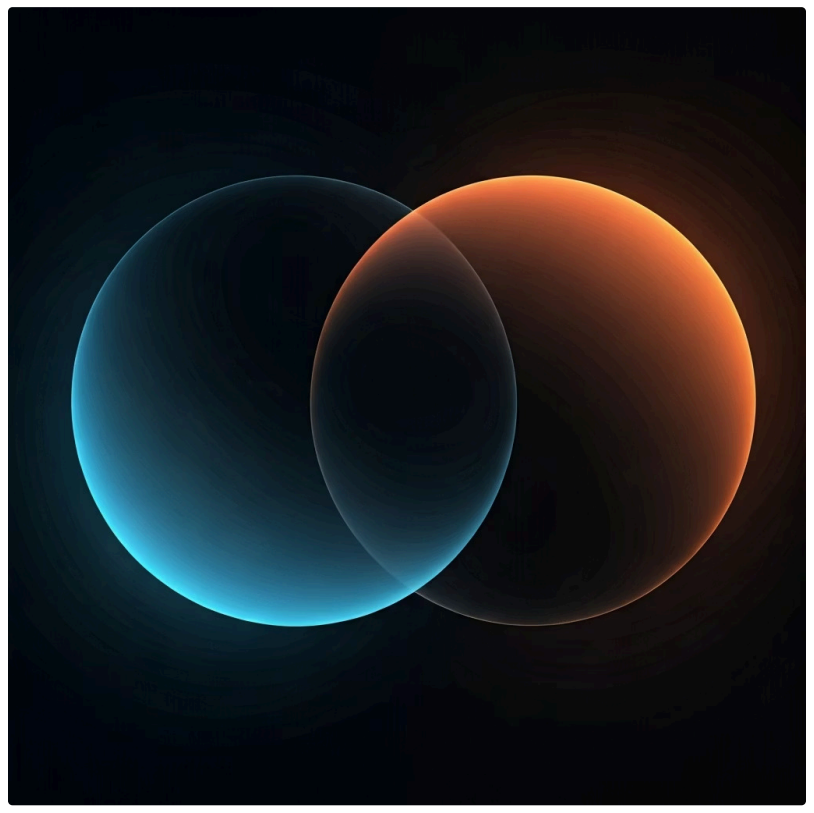
INNER JOIN em Detalhes

A cláusula INNER JOIN é utilizada para combinar linhas de duas ou mais tabelas com base em uma coluna comum entre elas. A condição de junção é especificada após a palavra-chave ON.

Considere duas tabelas: Clientes (com id_cliente, nome_cliente) e Pedidos (com id_pedido, id_cliente, valor_total). Para ver o nome do cliente e o valor de seus pedidos, você faria:

```
-- Listar o nome do cliente e o valor de seus pedidos
SELECT C.nome_cliente, P.valor_total
FROM Clientes C
INNER JOIN Pedidos P ON C.id_cliente = P.id_cliente;
```

Neste exemplo, C e P são apelidos (aliases) para as tabelas Clientes e Pedidos, respectivamente, tornando a consulta mais concisa e legível. O `ON C.id_cliente = P.id_cliente` é a condição que define como as tabelas serão unidas: as linhas serão combinadas quando o id_cliente for o mesmo em ambas as tabelas.



O INNER JOIN é ideal quando você precisa de um conjunto de dados onde todas as informações relacionadas estão presentes. Ele é a base para construir relatórios que exigem a integração de diferentes fontes de dados, um passo essencial para qualquer análise de negócios robusta.

LEFT JOIN: Mantendo a Base e Adicionando Detalhes Opcionais

O INNER JOIN é excelente para encontrar correspondências exatas, mas e se você precisar de todos os registros de uma tabela, mesmo que não haja correspondência na outra? Por exemplo, você quer listar *todos* os seus clientes, e se eles tiverem feito pedidos, mostrar esses pedidos. Se um cliente nunca comprou, ele ainda deve aparecer na lista.

Lista Completa

Mantém todos os registros da tabela esquerda

Detalhes Adicionais

Adiciona informações da tabela direita quando disponíveis

Valores NULL

Preenche com NULL quando não há correspondência

É aqui que o LEFT JOIN (também conhecido como LEFT OUTER JOIN) se torna a ferramenta ideal. Pense nele como uma lista de presença: você tem a lista completa de todos os alunos (tabela da esquerda), e para cada um, você anota se ele entregou o trabalho (tabela da direita). Se um aluno não entregou, ele ainda está na lista, mas a coluna do trabalho estará vazia (NULL).

Essa capacidade de manter todos os registros da tabela "esquerda" é crucial para análises que buscam identificar lacunas, como clientes inativos, produtos sem vendas ou funcionários sem projetos atribuídos. Ele oferece uma visão mais abrangente, revelando não apenas o que está conectado, mas também o que está faltando.

LEFT JOIN em Detalhes

O LEFT JOIN retorna todas as linhas da tabela da esquerda (a primeira tabela mencionada após FROM) e as linhas correspondentes da tabela da direita. Se não houver correspondência na tabela da direita, as colunas da tabela da direita terão valores NULL para aquela linha.

Vamos usar o exemplo de clientes e pedidos novamente. Queremos listar todos os clientes e, se eles tiverem pedidos, mostrar os detalhes desses pedidos.

```
-- Listar todos os clientes e seus pedidos (se houver)
SELECT C.nome_cliente, P.id_pedido, P.valor_total
FROM Clientes C
LEFT JOIN Pedidos P ON C.id_cliente =
P.id_cliente;
```

Nesta consulta, todos os clientes serão listados. Para os clientes que fizeram pedidos, você verá o id_pedido e o valor_total. Para os clientes que *não* fizeram pedidos, as colunas id_pedido e valor_total virão como NULL.

O LEFT JOIN é uma ferramenta poderosa para identificar oportunidades de negócio, como clientes que precisam ser reengajados, ou para garantir que nenhum dado primário seja perdido em sua análise. Ele é fundamental para construir relatórios que exigem uma visão completa de uma entidade principal, independentemente de suas relações com outras entidades.

RIGHT JOIN: O Outro Lado da Moeda da Inclusão

Assim como o LEFT JOIN prioriza a tabela da esquerda, o RIGHT JOIN (ou RIGHT OUTER JOIN) faz o oposto: ele prioriza a tabela da direita. Embora menos comum na prática, pois muitas vezes podemos reescrever uma consulta RIGHT JOIN como um LEFT JOIN invertendo a ordem das tabelas, ele é importante para entender a simetria das operações de junção.

Imagine que você tem uma lista de todos os produtos que foram vendidos (tabela da direita) e quer ver quais clientes os compraram (tabela da esquerda). Se um produto foi vendido, mas por algum erro de dados não está associado a nenhum cliente, o RIGHT JOIN ainda o incluirá no resultado, mostrando NULL para as informações do cliente.

Ele é útil em cenários específicos onde a "tabela da direita" é a sua fonte primária de interesse e você quer garantir que todos os seus registros sejam incluídos, mesmo que não haja correspondência na tabela da esquerda. Por exemplo, para identificar vendas que não foram associadas a um cliente válido, o que pode indicar um problema de integridade de dados.

RIGHT JOIN em Detalhes

O RIGHT JOIN retorna todas as linhas da tabela da direita (a segunda tabela mencionada após FROM) e as linhas correspondentes da tabela da esquerda. Se não houver correspondência na tabela da esquerda, as colunas da tabela da esquerda terão valores NULL para aquela linha.

```
-- Listar todos os pedidos e o nome do cliente (se houver)
SELECT C.nome_cliente, P.id_pedido, P.valor_total
FROM Clientes C
RIGHT JOIN Pedidos P ON C.id_cliente = P.id_cliente;
```

Embora o RIGHT JOIN possa ser reescrito como um LEFT JOIN invertendo a ordem das tabelas, entender seu funcionamento é crucial para uma compreensão completa das operações de junção e para escolher a abordagem mais clara e eficiente para cada problema de análise de dados.

Quadro Comparativo: INNER, LEFT e RIGHT JOIN

A escolha do tipo de JOIN é uma decisão fundamental na construção de consultas SQL, pois ela define quais dados serão incluídos no seu resultado final. Cada tipo de junção tem um propósito específico e é otimizado para diferentes cenários de análise. Compreender as nuances entre INNER, LEFT e RIGHT JOIN é essencial para garantir que suas consultas retornem exatamente as informações que você precisa, sem omissões ou inclusões indesejadas.

Pense nos JOINs como diferentes maneiras de combinar duas listas. O INNER JOIN é como encontrar os itens que aparecem em *ambas* as listas. O LEFT JOIN é como pegar todos os itens da primeira lista e, se eles aparecerem na segunda, adicionar as informações correspondentes. O RIGHT JOIN faz o mesmo, mas priorizando a segunda lista.

Essa distinção é vital para a integridade dos seus relatórios e para a precisão das suas análises. Uma escolha errada de JOIN pode levar a dados incompletos ou a uma superestimação/subestimação de métricas importantes.

Comparando os Tipos de JOIN

Tipo de JOIN	Descrição	Cenário de Uso Comum	Exemplo (Clientes C, Pedidos P)
INNER	Retorna apenas as linhas que têm correspondência em *ambas* as tabelas.	Quando você precisa de dados completos e relacionados de todas as tabelas envolvidas.	Clientes que fizeram pedidos.
LEFT	Retorna todas as linhas da tabela da esquerda e as correspondências da direita. NULL se não houver.	Quando você quer manter todos os registros de uma tabela base e adicionar informações opcionais.	Todos os clientes, e seus pedidos (se houver). Identificar clientes sem pedidos.
RIGHT	Retorna todas as linhas da tabela da direita e as correspondências da esquerda. NULL se não houver.	Quando você quer manter todos os registros de uma tabela secundária e adicionar informações opcionais.	Todos os pedidos, e o cliente que os fez (se houver). Identificar pedidos sem cliente associado.

> A ordem das cláusulas JOIN também é importante. Você pode encadear múltiplos JOINs para combinar mais de duas tabelas, sempre especificando a condição ON para cada junção. Por exemplo, para juntar Clientes, Pedidos e ItensPedido, você faria um JOIN de Clientes com Pedidos, e depois um JOIN do resultado com ItensPedido.

Dominar os diferentes tipos de JOIN é um marco na sua jornada de Data Literacy, permitindo que você construa consultas complexas e extraia insights valiosos de bancos de dados relacionais, um conhecimento fundamental para quem trabalha com Power BI e outras ferramentas de BI.

Subconsultas (Subqueries): Mergulhando Mais Fundo na Análise

Às vezes, para responder a uma pergunta de negócio, você precisa de uma informação que só pode ser obtida através de outra consulta. Imagine que você quer encontrar todos os produtos cujo preço é maior que o preço médio de *todos* os produtos. Primeiro, você precisa calcular o preço médio, e só então usar esse valor para filtrar os produtos.

É exatamente para isso que servem as subconsultas, ou subqueries. Elas são consultas SQL aninhadas dentro de outra consulta, como se fossem perguntas dentro de perguntas. A subconsulta é executada primeiro, e seu resultado é então usado pela consulta externa. Essa técnica permite resolver problemas complexos de forma elegante e modular, dividindo um grande desafio em etapas menores e gerenciáveis.

As subconsultas são uma ferramenta poderosa para análises que exigem múltiplos passos lógicos ou para criar filtros dinâmicos baseados em resultados de outras agregações. Elas expandem significativamente sua capacidade de interrogar os dados e extrair insights mais profundos.

Tipos e Aplicações de Subconsultas

Subconsultas podem ser usadas em diversas partes de uma consulta SQL:

No WHERE (para filtrar linhas)

Subconsultas Escalares: Retornam um único valor (uma linha, uma coluna). São usadas com operadores de comparação (=, >, <, etc.).

```
-- Encontrar produtos com preço acima da média
SELECT nome_produto, preco
FROM Produtos
WHERE preco > (SELECT AVG(preco) FROM Produtos);
```

Subconsultas de Linha/Tabela

Retornam uma lista de valores ou uma tabela. Usadas com operadores como IN, NOT IN, EXISTS, NOT EXISTS.

```
-- Encontrar clientes que fizeram pedidos
SELECT nome_cliente
FROM Clientes
WHERE id_cliente IN (SELECT DISTINCT id_cliente FROM Pedidos);
```

No FROM (como uma tabela derivada)

A subconsulta atua como uma tabela temporária que a consulta externa pode consultar. Isso é útil para pré-processar dados antes de realizar a consulta principal.

```
-- Calcular o total de vendas por cliente e depois filtrar
SELECT cliente_id, total_vendas
FROM (
  SELECT id_cliente AS cliente_id, SUM(valor_total) AS total_vendas
  FROM Pedidos
  GROUP BY id_cliente
) AS VendasPorCliente
WHERE total_vendas > 1000;
```

As subconsultas são um pilar para análises complexas e para a construção de relatórios dinâmicos. Elas permitem que você crie lógica de negócios sofisticada diretamente no SQL, um conhecimento valioso para qualquer analista de dados.

Subconsultas Correlacionadas e Considerações de Desempenho

As subconsultas são ferramentas poderosas, mas é crucial entender suas variações e as implicações de desempenho. Nem todas as subconsultas são criadas da mesma forma. Algumas são independentes da consulta externa, executando-se apenas uma vez. Outras, conhecidas como subconsultas correlacionadas, são mais complexas e podem impactar significativamente a performance.

Uma subconsulta correlacionada é aquela que depende da consulta externa para seus valores. Isso significa que ela é executada *uma vez para cada linha* processada pela consulta externa. Imagine que você quer encontrar todos os clientes que fizeram um pedido cujo valor é maior que a média dos pedidos *daquele cliente*. A média precisa ser calculada para cada cliente individualmente, o que exige que a subconsulta seja "correlacionada" com a linha atual da consulta externa.

Embora as subconsultas correlacionadas possam resolver problemas que seriam difíceis de abordar de outra forma, elas geralmente são menos eficientes do que as junções (JOIN) para o mesmo propósito, pois a execução repetida pode consumir muitos recursos. A escolha entre uma subconsulta e um JOIN muitas vezes se resume a clareza do código versus otimização de desempenho.

Exemplo de Subconsulta Correlacionada

```
-- Encontrar o pedido mais recente para cada cliente
SELECT C.nome_cliente, P.data_pedido, P.valor_total
FROM Clientes C
JOIN Pedidos P ON C.id_cliente = P.id_cliente
WHERE P.data_pedido = (
  SELECT MAX(P2.data_pedido)
  FROM Pedidos P2
  WHERE P2.id_cliente = C.id_cliente -- Correlação: P2.id_cliente depende de C.id_cliente
);
```

Priorize JOINs

Sempre que possível, tente reescrever subconsultas (especialmente as correlacionadas) como JOINs. Os otimizadores de consulta dos bancos de dados são geralmente muito eficientes em processar junções.

Indexação

Garanta que as colunas usadas nas condições de JOIN e WHERE (incluindo as subconsultas) estejam indexadas. Isso acelera drasticamente a busca de dados.

Clareza vs. Performance

Em alguns casos, uma subconsulta pode ser mais legível, mesmo que ligeiramente menos performática. O equilíbrio entre clareza e desempenho é uma decisão que o analista deve tomar.

Dominar as subconsultas e entender suas implicações é um sinal de maturidade na análise de dados, permitindo que você escreva consultas mais eficientes e extraia insights complexos com maior precisão.

Revisão e Conexão com o Mundo Real da Análise de Dados

Chegamos ao final de nossa jornada pelas consultas SQL intermediárias, e você agora possui um arsenal de ferramentas poderosas para transformar dados brutos em inteligência de negócios. Começamos com as funções de agregação, que nos permitem resumir grandes volumes de dados em métricas-chave. Em seguida, aprendemos a agrupar esses dados com GROUP BY, revelando insights por categoria, e a refinar esses grupos com HAVING.

Exploramos a arte de conectar tabelas com INNER JOIN, LEFT JOIN e RIGHT JOIN, permitindo-nos construir uma visão holística dos nossos dados, integrando informações de diferentes fontes. Finalmente, mergulhamos nas subconsultas, uma técnica avançada para resolver problemas complexos e criar filtros dinâmicos, expandindo ainda mais suas capacidades analíticas.

Dominar essas técnicas de SQL não é apenas uma habilidade técnica; é um pilar fundamental para a **Data Literacy**. No cenário de 2025, onde a tomada de decisão baseada em dados é imperativa, a capacidade de "conversar" com os bancos de dados e extrair informações precisas é o que diferencia um profissional. Ferramentas de Business Intelligence como o Power BI, que lideram o Quadrante Mágico do Gartner, dependem diretamente da qualidade e da estrutura dos dados que você consegue extrair e manipular com SQL.

O Próximo Nível da Análise

Com essas habilidades, você está apto a:

Criar relatórios gerenciais detalhados

Gerar KPIs e métricas de desempenho para diferentes segmentos de negócio.

Identificar tendências e padrões

Descobrir quais produtos vendem mais em quais regiões, ou quais clientes são mais valiosos.

Otimizar processos

Encontrar gargalos ou oportunidades de melhoria analisando dados de operações.

Preparar dados para BI

Estruturar e limpar dados para serem consumidos por ferramentas como Power BI, criando dashboards interativos e visuais impactantes.

> **O SQL é a linguagem universal dos dados**, e sua fluência nela é um diferencial competitivo inegável. Ele é a base sobre a qual toda a análise de dados moderna é construída, permitindo que você não apenas responda a perguntas, mas também formule as perguntas certas.

Consolidação e Autoavaliação

Chegamos ao fim desta aula, e esperamos que você se sinta mais confiante em sua capacidade de manipular e extrair insights valiosos de seus dados usando SQL intermediário. Lembre-se que a prática leva à perfeição. Quanto mais você experimentar com diferentes tipos de consultas e cenários, mais fluente você se tornará.

Em prática

Comece aplicando esses conceitos em pequenos projetos. Tente responder a perguntas de negócio que você tem no seu dia a dia usando as funções de agregação, agrupamentos e junções. Explore bases de dados públicas ou crie suas próprias tabelas de exemplo para simular situações reais. A capacidade de "pensar em SQL" é uma habilidade que se desenvolve com a experiência contínua.

Autoavaliação

01

Qual das seguintes cláusulas é utilizada para filtrar grupos de linhas após a aplicação de funções de agregação?

- a) WHERE
- b) GROUP BY
- c) HAVING
- d) ORDER BY

02

Para listar todos os clientes e, se eles tiverem feito pedidos, mostrar os detalhes desses pedidos (incluindo clientes sem pedidos), qual tipo de JOIN você utilizaria?

- a) INNER JOIN
- b) RIGHT JOIN
- c) FULL OUTER JOIN
- d) LEFT JOIN

03

Considere a seguinte consulta:

```
SELECT categoria,
COUNT(id_produto)
FROM Produtos
GROUP BY categoria
HAVING COUNT(id_produto) <
50;
```

O que essa consulta retorna?

- a) Todas as categorias com menos de 50 produtos.
- b) O número total de produtos em todas as categorias.
- c) As categorias que têm exatamente 50 produtos.
- d) Erro, pois HAVING não pode ser usado com COUNT.

04

Uma subconsulta escalar é aquela que:

- a) Retorna múltiplas linhas e múltiplas colunas.
- b) Retorna um único valor (uma linha, uma coluna).
- c) É sempre correlacionada com a consulta externa.
- d) Só pode ser usada na cláusula FROM.

05

Explique a diferença fundamental entre WHERE e HAVING e forneça um exemplo de uso para cada um em um contexto de análise de vendas.

Gabarito

Questão 1

c) HAVING

Questão 2

d) LEFT JOIN

Questão 3

a) Todas as categorias com menos de 50 produtos.

Questão 4

b) Retorna um único valor (uma linha, uma coluna).

Próximos Passos e Recursos Adicionais

Próxima Aula

Na **Aula 8 – Estatística Descritiva Essencial para Negócios**, vamos aprender a interpretar os dados que você agora é capaz de extrair e manipular. Entenderemos como medidas de tendência central, dispersão e distribuição podem revelar ainda mais sobre o comportamento dos seus dados e ajudar na tomada de decisões.

Recursos Adicionais

Documentação Oficial do SQL

Para aprofundar em sintaxe específica do seu banco de dados (SQL Server, MySQL, PostgreSQL, Oracle).

Livros e Cursos Online

Para explorar exemplos práticos e desafios de codificação.

Comunidades de Dados

Fóruns e grupos online para trocar experiências e tirar dúvidas.

NOTA IMPORTANTE

As informações técnicas desta aula estão atualizadas até 2025. Consulte sempre a documentação oficial do seu sistema de gerenciamento de banco de dados (SGBD) para verificar sintaxes e funcionalidades específicas, pois podem haver pequenas variações entre diferentes implementações de SQL.