HAVING - ANY – ALL - SELECT INTO – CASE – EXISTS – NULL - IFNULL

Prof. Ewerton

HAVING

a cláusula HAVING é usada em conjunto com a cláusula GROUP BY para filtrar os resultados de uma consulta após a agregação dos dados. Enquanto a cláusula WHERE é usada para filtrar os dados antes da agregação, a cláusula HAVING permite filtrar os dados após a agregação.

A cláusula HAVING é usada para definir uma condição que deve ser atendida pelos grupos resultantes da cláusula GROUP BY. Ela é aplicada a valores agregados, como médias, somas, contagens, etc. Com o HAVING, é possível filtrar os grupos com base em valores calculados.

A sintaxe básica do comando HAVING é a seguinte:

```
SELECT column1, column2, ...
FROM table
WHERE conditions
GROUP BY column1, column2, ...
HAVING conditions;
```

A cláusula HAVING segue a cláusula GROUP BY e pode conter qualquer condição que envolva funções agregadas ou colunas agregadas. Por exemplo, você pode usar expressões como SUM(), AVG(), COUNT(), etc., juntamente com operadores como =, >, <, BETWEEN, IN, etc., para filtrar os grupos resultantes.

No exemplo abaixo, estamos calculando a média de preços dos produtos por categoria e, em seguida, filtrando os grupos que possuem uma média de preço superior a 50. A cláusula HAVING nos permite aplicar essa condição após a agregação dos dados.

```
SELECT categoria, AVG(preco) AS media_preco
FROM produtos
GROUP BY categoria
HAVING AVG(preco) > 50;
```

Ex: Retornar os funcionários que têm mais de uma venda realizada com um valor total superior a 1000:

```
SELECT funcionario_id, COUNT(*) AS total_vendas, SUM(valor) AS total_vendas_valor
FROM vendas
GROUP BY funcionario_id
HAVING COUNT(*) > 1 AND SUM(valor) > 1000;
```

Nesse exemplo, estamos contando o número de vendas e calculando o valor total das vendas para cada funcionário. Em seguida, estamos filtrando os funcionários que têm mais de uma venda realizada e um valor total de vendas superior a 1000.

Ex: Retornar as cidades onde a média de temperatura é maior que 25 graus:

```
SELECT cidade, AVG(temperatura) AS media_temperatura
FROM dados_climaticos
GROUP BY cidade
HAVING AVG(temperatura) > 25;
```

Nesse exemplo, estamos calculando a média de temperatura para cada cidade com base nos dados climáticos. Em seguida, estamos retornando apenas as cidades onde a média de temperatura é maior que 25 graus.

ANY

o comando ANY é um operador usado em conjunto com operadores de comparação para verificar se algum dos valores em um conjunto de valores satisfaz uma condição. O ANY é usado como uma abreviação para a cláusula IN combinada com um subquery.

A sintaxe básica do comando ANY é a seguinte:

valor_coluna operador_comp ANY (subquery)

- valor_coluna é o valor com o qual você deseja comparar;
- operador_comp é um operador de comparação, como "=", ">", "<", ">=", "<=", etc.;
- subquery é uma subquery que retorna um conjunto de valores.

O comportamento do comando ANY é avaliar a condição para cada valor retornado pela subquery e retornar verdadeiro se a condição for verdadeira para pelo menos um dos valores.

Exemplos:

```
SELECT nome
FROM produtos
WHERE preco > ANY (SELECT preco FROM produtos WHERE categoria = 'Eletrônicos’)
```

Neste exemplo, a subquery SELECT preco FROM produtos WHERE categoria = 'Eletrônicos' retorna um conjunto de preços de produtos na categoria "Eletrônicos". O operador ANY verifica se o preço de cada produto na tabela "produtos" é maior do que qualquer um dos preços retornados pela subquery. Se a condição for verdadeira para pelo menos um dos produtos, o nome desse produto será retornado na consulta.

O comando ANY pode ser útil quando você precisa realizar comparações com múltiplos valores retornados por uma subquery e deseja verificar se pelo menos um valor satisfaz a condição.

Exemplos

Retornar os produtos com preço superior ao menor preço de qualquer produto na categoria "Eletrônicos":

```
SELECT nome
FROM produtos
WHERE preco > ANY (SELECT MIN(preco) FROM produtos WHERE categoria = 'Eletrônicos’)
```

Neste exemplo, a subquery SELECT MIN(preco) FROM produtos WHERE categoria = 'Eletrônicos' retorna o menor preço entre os produtos da categoria "Eletrônicos". A consulta principal compara o preço de cada produto com o menor preço retornado pela subquery e retorna os nomes dos produtos com preço superior.

Exemplos

Retornar os pedidos que possuem quantidade superior a pelo menos um pedido anterior:

```
SELECT numero_pedido
FROM pedidos
WHERE quantidade > ANY (SELECT quantidade FROM pedidos WHERE data_pedido < '2023-01-01’)
```

Neste exemplo, a subquery SELECT quantidade FROM pedidos WHERE data_pedido < '2023-01-01' retorna as quantidades de pedidos anteriores a uma determinada data. A consulta principal compara a quantidade de cada pedido com as quantidades retornadas pela subquery e retorna os números dos pedidos com quantidade superior.

Exemplos

Retornar os funcionários que possuem salário superior a qualquer salário na tabela "salarios_maximos":

```
SELECT nome
FROM funcionarios
WHERE salario > ANY (SELECT salario_maximo FROM salarios_maximos)
```

Neste exemplo, a subquery SELECT salario_maximo FROM salarios_maximos retorna os salários máximos registrados na tabela "salarios_maximos". A consulta principal compara o salário de cada funcionário com os salários retornados pela subquery e retorna os nomes dos funcionários com salário superior.

ALL

É usado para realizar comparações entre uma expressão e todos os valores de um conjunto retornado por uma subconsulta. Ele retorna verdadeiro se a expressão for verdadeira para todos os valores da subconsulta.

Exemplo: Retornar os produtos com preço maior do que todos os preços da categoria "Eletrônicos":

```
SELECT nome
FROM produtos
WHERE preco > ALL (SELECT preco FROM produtos WHERE categoria = 'Eletrônicos’)
```

Neste exemplo, a subconsulta SELECT preco FROM produtos WHERE categoria = 'Eletrônicos' retorna todos os preços dos produtos na categoria "Eletrônicos". A consulta principal compara o preço de cada produto com todos os preços retornados pela subconsulta e retorna os nomes dos produtos com preço maior do que todos os preços da categoria "Eletrônicos".

O comando ALL é útil quando você deseja verificar se uma expressão é verdadeira para todos os valores retornados por uma subconsulta. Ele pode ser combinado com operadores de comparação, como >, <, >=, <=, =, <>, para realizar comparações complexas em conjuntos de valores.

É importante observar que o comando ALL só pode ser usado com operadores de comparação, e a subconsulta deve retornar um conjunto de valores. Além disso, se a subconsulta retornar um conjunto vazio, o resultado da expressão com ALL será considerado falso.

Em resumo, o comando ALL no MySQL permite comparar uma expressão com todos os valores de um conjunto retornados por uma subconsulta e retorna verdadeiro se a expressão for verdadeira para todos os valores.

Mais exemplos

Retornar os departamentos nos quais todos os funcionários têm um salário maior que 5000:

```
SELECT departamento
FROM funcionarios
WHERE salario > ALL (SELECT 5000)
```

Neste exemplo, a subconsulta retorna um único valor constante de 5000. A consulta principal compara o salário de cada funcionário com esse valor e retorna os departamentos nos quais todos os funcionários têm um salário maior que 5000.

Mais exemplos

Retornar os alunos que obtiveram notas maiores que todas as notas dos alunos da mesma turma:

```
SELECT nome, turma, nota
FROM alunos
WHERE nota > ALL (SELECT nota FROM alunos WHERE turma = 'A’)
```

Neste exemplo, a subconsulta retorna todas as notas dos alunos da turma "A". A consulta principal compara a nota de cada aluno com todas as notas retornadas pela subconsulta e retorna os alunos que obtiveram notas maiores que todas as notas da turma "A".

Mais exemplos

Retornar os produtos que têm um preço maior que todos os produtos de uma determinada categoria:

SELECT nome, preco, categoria

FROM produtos

WHERE preco > ALL (SELECT preco FROM produtos WHERE categoria = 'Eletrônicos’)

Neste exemplo, a subconsulta retorna todos os preços dos produtos na categoria "Eletrônicos". A consulta principal compara o preço de cada produto com todos os preços retornados pela subconsulta e retorna os produtos que têm um preço maior do que todos os produtos da categoria "Eletrônicos".

Mais exemplos

Retornar os clientes que têm uma idade maior que todos os funcionários:

```
SELECT nome, idade
FROM clientes
WHERE idade > ALL (SELECT idade FROM funcionarios)
```

Neste exemplo, a subconsulta retorna todas as idades dos funcionários. A consulta principal compara a idade de cada cliente com todas as idades retornadas pela subconsulta e retorna os clientes que têm uma idade maior do que todos os funcionários.

SELECT INTO

É usado para criar uma nova tabela a partir dos resultados de uma consulta. Ele permite selecionar dados de uma ou várias tabelas e inseri-los em uma nova tabela que será criada no processo.

O comando SELECT INTO não está disponível no MySQL para criar uma nova tabela a partir de uma consulta direta.

Para criar uma nova tabela a partir de uma consulta no MySQL, você pode usar a cláusula CREATE TABLE AS SELECT.

CREATE TABLE AS SELECT

O comando CREATE TABLE AS SELECT no MySQL permite criar uma nova tabela com base nos resultados de uma consulta SELECT. Ele combina a criação de uma tabela e a inserção de dados em uma única instrução.

A sintaxe básica é a seguinte:

```
CREATE TABLE new_table_name AS
SELECT column1, column2, ...
FROM source_table
WHERE condition;
```

Aqui está uma explicação de cada parte do comando:

new_table_name: É o nome da nova tabela que será criada.

SELECT column1, column2, ...: É a consulta SELECT que define quais colunas e dados serão inseridos na nova tabela. Você pode especificar as colunas desejadas ou usar * para selecionar todas as colunas da tabela de origem.

FROM source_table: É a tabela de origem dos dados.

WHERE condition: É uma condição opcional que permite filtrar os dados da tabela de origem antes de serem inseridos na nova tabela.

A nova tabela será criada com a estrutura das colunas e os tipos de dados inferidos da consulta SELECT. Os nomes das colunas na nova tabela serão determinados pelas colunas selecionadas na consulta SELECT.

Exemplo

```
CREATE TABLE employees_backup AS
SELECT employee_id, first_name, last_name, hire_date
FROM employees
WHERE hire_date >= '2022-01-01’;
```

Neste exemplo, uma nova tabela chamada employees_backup será criada com as colunas employee_id, first_name, last_name e hire_date. Os dados serão selecionados da tabela employees apenas para os funcionários contratados em 2022.

O comando CREATE TABLE AS SELECT é útil quando você deseja criar rapidamente uma nova tabela com base em uma consulta existente sem a necessidade de criar explicitamente a estrutura da tabela antes.

CASE

O comando representa uma expressão condicional que permite realizar avaliações condicionais e retornar diferentes valores com base em condições específicas. Ele é usado para realizar lógica condicional em consultas SQL.

Existem duas formas principais de usar o comando CASE no MySQL: a forma simples e a forma de pesquisa.

- A forma simples do CASE é usada quando você precisa avaliar uma expressão e retornar um valor correspondente.
- A forma de pesquisa do CASE é usada quando você precisa realizar avaliações mais complexas com várias condições.

Forma Simples:

```
CASE expression
    WHEN value1 THEN result1
    WHEN value2 THEN result2
    ...
    ELSE result
END
```

Aqui está uma explicação de cada parte do comando:

- expression: É a expressão a ser avaliada.
- value1, value2, ...: São os valores a serem comparados com a expressão.
- result1, result2, ...: São os resultados retornados quando a expressão corresponde a um valor.
- ELSE result: É um resultado opcional retornado quando a expressão não corresponde a nenhum dos valores fornecidos.

Forma Simples exemplo:

```
SELECT name,
    CASE department_id
        WHEN 1 THEN 'Sales'
        WHEN 2 THEN 'Marketing'
        WHEN 3 THEN 'Finance'
        ELSE 'Other'
    END AS department
FROM employees;
```

Neste exemplo, o comando CASE é usado para avaliar o valor da coluna department_id e retornar o departamento correspondente com base nos valores especificados. Se o valor de department_id for 1, será retornado "Sales"; se for 2, será retornado "Marketing"; se for 3, será retornado "Finance"; caso contrário, será retornado "Other".

Forma de pesquisa:

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    ...
    ELSE result
END
```

Aqui, cada WHEN possui uma condição específica e o resultado correspondente é retornado quando a condição é verdadeira. A condição pode ser qualquer expressão lógica válida.

Forma de pesquisa exemplo:

```
SELECT name,
    CASE
        WHEN salary >= 5000 THEN 'High'
        WHEN salary >= 3000 THEN 'Medium'
        ELSE 'Low'
    END AS salary_category
FROM employees;
```

Neste exemplo, o CASE é usado para avaliar o salário dos funcionários e atribuir uma categoria com base em faixas salariais. Se o salário for maior ou igual a 5000, será atribuída a categoria "High"; se for maior ou igual a 3000, será atribuída a categoria "Medium"; caso contrário, será atribuída a categoria "Low".

Outro exemplo

```
SELECT jog_id, jog_nome, jog_posicao, jog_idade,
    CASE
        WHEN jog_idade < 18 THEN 'Menor de 18'
        WHEN jog_idade >= 18 AND jog_idade < 25 THEN '18-24'
        WHEN jog_idade >= 25 AND jog_idade < 30 THEN '25-29'
        ELSE '30 ou mais'
    END AS faixa_etaria
FROM jogadores;
```

Aqui seria calculada a faixa etária com base na idade dos jogadores

EXISTS

O comando EXISTS é uma cláusula condicional no MySQL que verifica a existência de registros em uma subconsulta. Ele retorna verdadeiro se a subconsulta retornar algum resultado e falso se a subconsulta não retornar nenhum resultado.

A estrutura básica do comando EXISTS é a seguinte:

SELECT colunas

FROM tabela

WHERE EXISTS (subconsulta);

A subconsulta é uma consulta aninhada que normalmente é usada para verificar a existência de registros em outra tabela, com base em uma condição específica. A subconsulta pode ser qualquer consulta SQL válida.

Exemplo de uso do comando EXISTS

```
SELECT tme_nome
FROM times
WHERE EXISTS (
    SELECT *
    FROM partidas
    WHERE tme_id_mandante = times.tme_id
        OR tme_id_visitante = times.tme_id
);
```

No exemplo anterior, a consulta retorna o nome das equipes (tme_nome) que estão envolvidas em pelo menos uma partida registrada na tabela partidas. A condição EXISTS verifica se há alguma partida em que o tme_id_mandante ou tme_id_visitante corresponda ao tme_id da tabela times.

O comando EXISTS é útil quando você precisa verificar a existência de registros em uma tabela com base em determinados critérios e tomar decisões condicionais com base nessa verificação.

NULL

No MySQL, o valor NULL representa a ausência de um valor válido ou desconhecido em uma coluna de uma tabela. Ele indica que não há nenhum valor atribuído à coluna para uma determinada linha.

Aqui estão algumas características importantes sobre o uso do NULL no MySQL:

1. O NULL é diferente de uma string vazia (''): Uma string vazia é um valor válido, enquanto o NULL representa a ausência de um valor.
2. O NULL é tratado como um valor desconhecido: Quando uma coluna contém o valor NULL, isso significa que o valor é desconhecido ou não está disponível.

NULL

3. Operações aritméticas com NULL resultam em NULL: Qualquer operação aritmética envolvendo o valor NULL resultará em NULL. Por exemplo, NULL + 5 é NULL.
4. A comparação com NULL usa o operador IS NULL ou IS NOT NULL: Para verificar se um valor é NULL, você deve usar o operador IS NULL. Para verificar se um valor não é NULL, você deve usar o operador IS NOT NULL. Comparações com NULL usando operadores de igualdade (=, <>, etc.) não funcionarão corretamente.
5. O NULL pode ser armazenado em colunas de qualquer tipo de dados: O NULL pode ser atribuído a colunas de qualquer tipo de dados, como números, strings, datas, etc.

Exemplo de uso do NULL em uma consulta

```
SELECT nome
FROM clientes
WHERE data_nascimento IS NULL;
```

Neste exemplo, estamos selecionando os clientes cuja data de nascimento é desconhecida, ou seja, possui o valor NULL.

O uso correto do NULL é importante para lidar com valores ausentes ou desconhecidos em suas consultas e operações no MySQL.