Diferença entre

INNER JOIN, LEFT JOIN e RIGHT JOIN

Quando estamos trabalhando com múltiplas tabelas em SQL, usamos os comandos `JOIN` para combinar os dados entre elas. Cada tipo de `JOIN` determina como as tabelas serão combinadas e quais registros serão retornados. Vamos entender a diferença entre **INNER JOIN**, **LEFT JOIN** e **RIGHT JOIN** com exemplos claros.

INNER JOIN

- O **INNER JOIN** retorna **apenas os registros que têm correspondências** em ambas as tabelas.
- Se uma linha da tabela à esquerda ou à direita não tiver uma correspondência na outra tabela, ela será excluída do resultado.

Sintaxe:

```
SELECT colunas
FROM tabela1
INNER JOIN tabela2
ON tabela1.coluna_comum = tabela2.coluna_comum;
```

Exemplo: Se tivermos as tabelas `Clientes` e `Pedidos`:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
INNER JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

- Isso retornará **apenas** os clientes que fizeram pedidos, ou seja, aqueles com um `cliente_id` que exista nas duas tabelas.

LEFT JOIN (ou LEFT OUTER JOIN)

- O **LEFT JOIN** retorna **todos os registros da tabela à esquerda** (primeira tabela mencionada), **mesmo que não haja correspondência** na tabela à direita.
- Se não houver correspondência, os valores das colunas da tabela à direita serão preenchidos com `NULL`.

Sintaxe:

```
SELECT colunas
FROM tabela1
LEFT JOIN tabela2
ON tabela1.coluna_comum = tabela2.coluna_comum;
```

Exemplo:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
LEFT JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

- Isso retornará **todos os clientes**, inclusive aqueles que **não** fizeram pedidos. Se um cliente não tiver feito pedido, a coluna `pedido_id` será `NULL`.
- Neste exemplo, "Carlos Lima" aparece no resultado, mas como ele não fez nenhum pedido, o valor da coluna `pedido_id` será `NULL`.

RIGHT JOIN (ou RIGHT OUTER JOIN)

- O **RIGHT JOIN** é o inverso do **LEFT JOIN**: ele retorna **todos os registros da tabela à direita** (segunda tabela mencionada), **mesmo que não haja correspondência** na tabela à esquerda.
- Se não houver correspondência, os valores das colunas da tabela à esquerda serão preenchidos com `NULL`.

Sintaxe:

```
SELECT colunas
FROM tabela1
RIGHT JOIN tabela2
ON tabela1.coluna_comum = tabela2.coluna_comum;
```

Exemplo:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

- Isso retornará **todos os pedidos**, inclusive os que não estão associados a nenhum cliente.
- Neste exemplo, existe um `pedido_id` com o valor 3 que não tem um cliente associado, então o valor da coluna `nome` será `NULL`.

Comparando os Resultados

INNER JOIN:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
INNER JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

Explicação: Retorna apenas os registros que têm correspondência nas duas tabelas. Carlos Lima, que não tem pedido, e o pedido sem cliente não aparecem.

LEFT JOIN:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
LEFT JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

Explicação: Retorna todos os clientes, incluindo os que não têm pedidos (Carlos Lima). Para esses, o `pedido_id` é `NULL`.

RIGHT JOIN:

```
SELECT Clientes.nome, Pedidos.pedido_id
FROM Clientes
RIGHT JOIN Pedidos ON Clientes.cliente_id = Pedidos.cliente_id;
```

Explicação: Retorna todos os pedidos, incluindo os que não têm cliente associado (o pedido 3). Para esses, o nome do cliente é `NULL`.

Resumo:

- **INNER JOIN**: Retorna **apenas** os registros que têm correspondência em ambas as tabelas.

- **LEFT JOIN**: Retorna **todos os registros da tabela à esquerda**, e os registros da direita apenas quando houver correspondência. Quando não há correspondência, os valores da tabela à direita são `NULL`.

- **RIGHT JOIN**: Retorna **todos os registros da tabela à direita**, e os registros da esquerda apenas quando houver correspondência. Quando não há correspondência, os valores da tabela à esquerda são `NULL`.

Esses três tipos de `JOIN` são essenciais para manipular dados relacionados em múltiplas tabelas, permitindo que você crie consultas complexas e obtenha as informações de maneira estruturada.