Nesse vídeo, aprenderemos sobre o comando UNION, que é responsável por juntar duas consultas. Para exemplificar, vamos supor que fizemos duas consultas separadas e cada uma delas retornou uma das seguintes tabelas:

Ao rodar o comando UNION com essas duas consultas, será retornada uma lista única com todos os registros:

Note que há uma restrição para esse comando: é necessário que as tabelas que serão unidas tenham o mesmo número e tipo de campo. No caso, não tivemos problemas, pois as duas tabelas têm duas colunas e os tipos dos campos correspondem.

Os nomes das colunas em si não precisam ser iguais para que o UNION funcione. Veremos mais adiante o que acontece quando temos nomes diferentes.

Agora, vamos modificar a segunda tabela para que haja um registro idêntico ao da primeira (praia):

Executando o UNION simples, o DISTINCT automaticamente será aplicado, de forma que os **registros iguais também serão unidos**. No caso, o registro "praia" será mostrado um única vez:

Mas se nosso objetivo seja unir **sem aplicar o DISTINCT**, então utilizaremos a cláusula UNION ALL:

Observe que agora o registro "praia" aparece duas vezes.

Vamos criar alguns exemplos na nossa base "sucos\_vendas". Abriremos o MySQL Workbench, criaremos um novo script e começaremos com duas seleções:

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Como resultado, teremos duas tabelas - uma apresentará 11 registros referentes aos bairros dos clientes, outra mostrará 4 bairros onde há escritórios dos vendedores. Note que alguns dados se cruzam: Tijuca, por exemplo, é um elemento que aparece em ambas.

A seguir, vamos executar um comando com UNION:

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

No retorno, note que Tijuca é o terceiro item e não se repete mais na lista. Por ser comum às duas tabelas, esse bairro torna-se um registro só. Em outras palavras, o DISTINCTfoi aplicado. Para mudar esse cenário, teríamos que rodar o UNION ALL:

Texto

O conteúdo gerado por IA pode estar incorreto.

Nesse caso, Tijuca aparecerá duas vezes.

É interessante notar que o DISTINCT funciona somente se os registros forem **idênticos em todos os campos** que aparecem na consulta. Para fazer uma demonstração, vamos acrescentar o campo "NOME" na seleção:

Uma imagem contendo Interface gráfica do usuário

O conteúdo gerado por IA pode estar incorreto.

Perceba que o campo "NOME" no primeiro SELECT refere-se aos nomes dos clientes, enquanto no segundo é relativo aos vendedores.

Nesse contexto, o DISTINCT só vai agrupar registros caso o bairro **E** o nome forem iguais.

Podemos continuar incrementando a consulta, colocando mais campos:

Tela preta com letras brancas

O conteúdo gerado por IA pode estar incorreto.

Note que as consultas permanecem com o mesmo número de colunas e os tipos das colunas correspondem. Além do mais, usamos o *alias* "TIPO" nos dois SELECTs para a nova coluna.

Entretando, ao optar por apelidos diferentes em cada SELECT, apenas o primeiro será considerado. Os nomes das colunas correspondem aos da primeira seleção:

**SELECT** **DISTINCT** BAIRRO, NOME, 'CLIENTE' **as** TIPO\_CLIENTE **FROM** tabela\_de\_clientes

**UNION**

**SELECT** **DISTINCT** BAIRRO, NOME, 'VENDEDOR' **as** TIPO\_VENDEDOR **FROM** tabela\_de\_vendedores;

No retorno, vê-se que a terceira coluna é "TIPO\_CLIENTE", o *alias* "TIPO\_VENDEDOR" foi ignorado. Independentemente do nome, os valores permanecem os mesmos.

A título de exemplo, vamos tentar rodar um UNION com número de colunas diferentes, incluindo o campo "CPF" da tabela de clientes:

**SELECT** **DISTINCT** BAIRRO, NOME, 'CLIENTE' **as** TIPO\_CLIENTE, CPF **FROM** tabela\_de\_clientes

**UNION**

**SELECT** **DISTINCT** BAIRRO, NOME, 'VENDEDOR' **as** TIPO\_VENDEDOR, MATRICULA **FROM** tabela\_de\_vendedores;

O MySQL Workbench acusará um problema, como esperado, pois temos 3 colunas no primeiro SELECT e duas no outro. Podemos resolver essa situação adicionando uma coluna ao segundo SELECT. No caso, vamos adicionar a matrícula, que é do mesmo **tipo** da coluna "CPF":

**SELECT** **DISTINCT** BAIRRO, NOME, 'CLIENTE' **as** TIPO\_CLIENTE, CPF **FROM** tabela\_de\_clientes

**UNION**

**SELECT** **DISTINCT** BAIRRO, NOME, 'VENDEDOR' **as** TIPO\_VENDEDOR, MATRICULA **FROM** tabela\_de\_vendedores;

Assim, solucionamos o erro. Perceba, no entanto, que o nome da última coluna será "CPF". Como vimos há pouco, serão considerados apenas os nomes do primeiro SELECT.

Finalmente, vamos retomar aquele problema que tivemos no último vídeo, quando descobrimos que o MySQL não suporta o FULL JOIN. Agora que conhecemos o comando UNION, somos capazes de simular o FULL JOIN ao fazer a união de uma consulta LEFT JOIN com outra RIGHT JOIN:

**SELECT** tabela\_de\_vendedores.BAIRRO,

tabela\_de\_vendedores.NOME, DE\_FERIAS,

tabela\_de\_clientes.BAIRRO,

tabela\_de\_clientes.NOME **FROM** tabela\_de\_vendedores **LEFT** **JOIN** tabela\_de\_clientes

**ON** tabela\_de\_vendedores.BAIRRO = tabela\_de\_clientes.BAIRRO

**UNION**

**SELECT** tabela\_de\_vendedores.BAIRRO,

tabela\_de\_vendedores.NOME, DE\_FERIAS,

tabela\_de\_clientes.BAIRRO,

tabela\_de\_clientes.NOME **FROM** tabela\_de\_vendedores **RIGHT** **JOIN** tabela\_de\_clientes

**ON** tabela\_de\_vendedores.BAIRRO = tabela\_de\_clientes.BAIRRO;

Os dois SELECTs tem a mesma quantidade de colunas e os tipos dos campos correspondem, então a consulta será bem-sucedida. Enfim, poderemos analisar em uma única consulta quais são os bairros que têm clientes e vendedores (por exemplo, Tijuca); quais possuem vendedores mas não dispõem de compradores cadastrados (Copacabana); e em quais moram clientes porém não têm vendedores (por exemplo, Água Santa). Essas eram exatamanente as informações que esperávamos do FULL JOIN, ou seja, esse comando pode ser simulado com o LEFT JOIN e o RIGHT JOIN com o UNION entre eles.