Nesse vídeo, faremos alguns exemplos com CROSS JOIN e FULL JOIN.

Durante a explicação, vamos aproveitar para aprender como realizar essas consultas sem utilizar os *aliases* para as tabelas. No vídeo anterior, por exemplo, apelidamos a tabela de clientes de "A" e a tabela de notas fiscais de "B" e, para nos referir aos seus respectivos campos, usamos essas letras como prefixos (A.NOMEe B.CPF). Agora, veremos que também é possível usar os próprios nomes das tabelas.

Quanto aos prefixos e sufixos, notaremos que eles são obrigatórios somente quando lidamos com campos em comum nas tabelas. Costumamos usá-los porque, em geral, não sabemos de antemão se os campos se repetem, porém por vezes os prefixos são opcionais e vamos demonstrar isso.

Então, começaremos abrindo um novo script no MySQL Workbench e selecionando a tabela de vendedores:



Uma das colunas do resultado é "BAIRRO", referente ao local onde o vendedor possui escritório. Consultando a tabela de clientes, veremos que ela também contém esse campo:



Sabendo dessa relação entre as tabelas, criaremos uma seleção com JOIN:

Interface gráfica do usuário, Texto

O conteúdo gerado por IA pode estar incorreto.

Note que nessa seleção não usamos *alias*! Em vez disso, utilizamos os próprios nomes das tabelas como prefixo, como em tabela\_de\_vendedores.BAIRRO. Inclusive, quando digitamos o ponto depois do nome da tabela, o MySql Workbench até mostra algumas sugestões de preenchimento para agilizar o processo.

O retorno mostrará os clientes que estão em bairros onde há escritórios da empresa de sucos. Nessa consulta, obtemos somente 7 registros, o que significa que 8 clientes estão em bairros que não têm escritório, pois no vídeo anterior descobrimos que são 15 clientes cadastrados no total.

No momento, essa seleção traz **todos** os campos das duas tabelas, então vamos melhorar essa organização e trazer somente os quatro campos que nos interessam:

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

tabela\_de\_vendedores.NOME,

tabela\_de\_clientes.BAIRRO,

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

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

Assim, fica mais fácil de visualizar as informações.

Para demonstrar um ponto interessante sobre os prefixos, acrescentaremos também a coluna "DE\_FERIAS", da tabela de vendedores:

Texto

O conteúdo gerado por IA pode estar incorreto.

A coluna "DE\_FERIAS" não existe na tabela de clientes, ela está presente apenas na de vendedores. Isso nos permite omitir o prefixo, pois o MySQL consegue deduzir e localizar sozinho o campo a que nos referimos, já que ele só existe em uma tabela. No caso de "BAIRRO" e "NOME", por exemplo, o prefixo é **obrigatório** para se fazer a distinção, pois são **campos em comum** nas duas tabelas que estamos usando:

Texto

O conteúdo gerado por IA pode estar incorreto.

A seguir, vamos rodar a consulta com LEFT JOIN:

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

tabela\_de\_vendedores.NOME,

DE\_FERIAS,

tabela\_de\_clientes.BAIRRO,

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;

Essa seleção retornará **todos os vendedores e apenas os clientes correspondentes**. Encontraremos, por exemplo, o registro da vendedora Roberta Martins, cujo bairro (Copacabana) não tem correspondência na tabela de clientes. Chegamos a essa conclusão porque a terceira e a quarta coluna estão com valor *null*. Ou seja, o seu escritório não está em um lugar estratégico, pois não há clientes cadastrados que comprem sucos nesse bairro.

Substituindo o comando por RIGHT JOIN, o MySQL trará **todos os clientes e apenas os vendedores correspondentes**. Com esse resultado, é possível verificar vários compradores que moram em bairros em que não há escritórios da empresa de sucos, como Água Santa e Brás - os três primeiros campos são nulos. Esse tipo de análise seria interessante, por exemplo, para investigar onde há mais demanda para abrir um novo escritório.

Podemos ver todas essas informações ao mesmo tempo usando o FULL JOIN - todos os vendedores, inclusive os que tem escritórios nos bairros onde não há compradores; e todos os clientes, inclusive os que moram em bairros em que não há escritórios da empresa de sucos:

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

tabela\_de\_vendedores.NOME,

DE\_FERIAS,

tabela\_de\_clientes.BAIRRO,

tabela\_de\_clientes.BAIRRO,

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

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

Ao executar essa consulta, o programa vai alegar um erro. Como foi explicado no começo do curso, a linguagem SQL segue o padrão *ANSI*, que respeita uma série de regras, mas nem todo gerenciador de banco de dados realiza 100% do que esse padrão especifica. O FULL JOIN está contido no padrão *ANSI*, porém o MySQL não suporta esse comando. Ou seja, não conseguiremos fazer o FULL JOIN no MySQL Workbench. Existe, no entanto, uma alternativa para o FULL JOIN que é fazer o LEFT JOIN e o RIGHT JOIN simultaneamente. Nesse momento, ainda não aprendemos como fazer essa união de consultas, então vamos reservar esse erro e, mais adiante, quando estudarmos mais sobre o assunto, voltaremos a ele.

Para finalizar esse vídeo, vamos criar um exemplo com CROSS JOIN, lembrando que esse comando não requer que seja especificado o campo em comum nem que seja escrito o termo CROSS JOIN:

Texto

O conteúdo gerado por IA pode estar incorreto.

Assim, o resultado será uma análise combinatória entre bairros de vendedores e bairros de clientes.