Agora que temos uma noção do conteúdo da nossa base de dados, podemos começar a fazer consultas. No MySQL Workbench, criaremos um novo *script* SQL, clicando no primeiro ícone da barra superior do programa (ou usando o atalho "Ctrl + T").

De início, é importante escolher a base de dados com que vamos trabalhar. Para selecionar um banco, basta dar um clique duplo sobre seu nome e ele ficará destacado em negrito, significando que está selecionado. Por exemplo, posso escolher "sakila" (um banco padrão do MySQL Workbench) e tudo que eu fizer será aplicado a esta base. Nesse sentido, como pretendo me dedicar à base "sucos\_vendas", darei um clique duplo nela.

Outra alternativa é "forçar" a conexão com um banco através do comando USE. Em nosso *script*, digitamos:

USE sucos\_vendas;

Em seguida, execute o *script*, clicando no terceiro ícone da barra acima da área de digitação (um símbolo de raio).

Não se esqueça de colocar um ponto e vírgula (**;**) ao final de cada linha.

Na sintaxe SQL, não há distinção entre letras maiúsculas e minúsculas, portanto você pode digitar USE ou use. Opto pela primeira opção para destacar as partes que são comandos SQL.

Com o banco selecionado, vamos examinar as tabelas. Para conferir todo o conteúdo da "tabela\_de\_clientes", por exemplo, basta usar o comando SELECT, seguido das colunas que gostaríamos de ver (separadas por vírgula), seguidas de FROM e o nome da tabela a que estamos nos referindo:

**SELECT** CPF, NOME, ENDERECO\_1, ENDERECO\_2, BAIRRO, CIDADE, ESTADO, CEP, DATA\_DE\_NASCIMENTO, IDADE, SEXO, LIMITE\_DE\_CREDITO, VOLUME\_DE\_COMPRA, PRIMEIRA\_COMPRA **FROM** tabela\_de\_clientes;

Selecionaremos esse trecho do *script* e clicaremos no botão de execução. Ao rodar o código, logo abaixo teremos uma tabela como retorno. Note que, caso algum termo esteja escrito incorretamente, o programa acusará um erro na área *output* (saída), no painel inferior. Se isso ocorrer, corrija e execute novamente.

À direita do resultado da consulta, há três opções de visualização. A padrão é *Result Grid*, que organiza os dados em forma de grade. *Form Editor* mostra o resultado com um leiaute semelhante a uma entrada de dados, com opções de navegação na parte superior dessa área. Já *Field Types* foca nos tipos de dados dos resultados das seleções. Ademais, também temos a aba *Query Stats*, que nos informa sobre o tempo de execução da *query* (consulta).

Voltando ao nosso ambiente, um modo mais simples de consultar todos os campos de uma tabela é usar o asterisco em vez de escrever o nome de todas as colunas:

Texto

O conteúdo gerado por IA pode estar incorreto.

Em outras palavras, ao colocar o asterisco, estamos indicando que queremos consultar **todos** os campos da tabela. O código é diferente, porém o resultado é o mesmo.

Vale destacar que nem sempre há necessidade de consultar todas as colunas, é possível especificar apenas os campos desejados:

Uma imagem contendo Texto

O conteúdo gerado por IA pode estar incorreto.

Em consultas SQL, existe a opção do uso de *alias*, que funciona como um apelido que atribuímos a determinado campo. Isso é feito com o uso de as:



Nesse caso, "identificador" representa o CPF e "cliente" representa o NOME. Assim, quando o script for executado, notaremos que as colunas do resultado são os *aliases* e não os nomes originais dos campos (CPF e NOME).

Como expliquei anteriormente, as nomenclaturas usadas para tabelas nem sempre são facilmente identificáveis e, às vezes, isso também ocorre em relação às colunas das tabelas. Nesses casos, o uso de *aliases* nos auxilia nas leituras dos dados, pois tornam os campos mais reconhecíveis.

Outro cenário em que os *aliases* são relevantes são as consultas que empregam o JOIN. A cláusula JOIN (que aprenderemos mais adiante no curso) permite a junção de duas tabelas em uma mesma consulta e, nesse processo, pode ser que essas tabelas tenham campos com os mesmos nomes. Aqui, então, o uso de *aliases* é interessante para distinguir esses campos.

A seguir, vamos consultar a "tabela\_de\_produtos":



Até agora, fizemos consultas que abrangiam tudo de uma tabela, mas por meio da cláusula WHERE temos a opção de segregar os dados que nos interessa ver:



Nesse caso, determinamos uma **condição lógica** (CODIGO\_DO\_PRODUTO = "1000889") de modo que o resultado dessa consulta mostra apenas o produto que possui esse código específico. Aqui, trata-se do "Sabor da Montanha - 700ml - Uva".

Às vezes, o editor de texto do MySQL Workbench traz opções de preenchimento automático enquanto digitamos. Por exemplo, caso você erre o nome de uma tabela, ele possivelmente trará uma caixa de sugestões com nomes de tabelas que existem no seu banco de dados.

"CODIGO\_DO\_PRODUTO" é uma chave primária, no entanto, qualquer campo pode ser usado para filtrar resultados, por exemplo, sabor:

Texto

O conteúdo gerado por IA pode estar incorreto.

Essa consulta retorna apenas um produto, pois somente um suco tem sabor uva. Trocando o valor "uva" por "limão", nenhum produto será mostrado, pois não há sucos com esse sabor. Já ao escrever "laranja", são cinco os produtos que atendem a essa condição.

Mas qual é a grande diferença interna na execução da consulta com WHERE CODIGO\_DO\_PRODUTO = '1000889' e da consulta com WHERE SABOR = 'Uva'? A resposta está na performance!

Buscas com condições de filtro que possuem chaves primárias (como "CODIGO\_DO\_PRODUTO") são mais rápidas, pois chaves primárias têm índices que facilitam muito esse processo. "SABOR", por outro lado, não é uma chave primária nem estrangeira, logo não tem índice e consequentemente resulta numa busca um pouco mais lenta.

Como nosso banco de dados atual é relativamente pequeno, essa lentidão não é perceptível, porém essa diferença de performance pode ser problemática em bancos mais volumosos. Uma forma de solucionar esse problema é atribuindo um índice à coluna, algo que aprenderemos mais para frente, em outro curso de MySQL.

Seguindo com as consultas, agora vamos fazer um filtro referente à embalagem:



Ao executar esse *script*, note que na tabela o valor "PET" está escrito com letras **maiúsculas**. O que será que acontece se eu fizer a consulta usando letras **minúsculas**?



O resultado é o mesmo! Como vimos anteriormente, o MySQL não distingue entre letras maiúsculas e minúsculas, ele fará a busca da mesma forma.

Também podemos fazer seleções usando valores como critérios. Na nossa base, temos o produto "Videira do Campo - 1,5 Litros - Melancia" cujo "PRECO\_DE\_LISTA" é 19.51. Vamos criar um filtro usando esse valor como critério:



Ao rodar essa consulta, o MySQL retorna vazio. Mas por que isso acontece, se vimos que existe um produto que corresponde a esse filtro? Ao verificar mais detalhes sobre "PRECO\_DE\_LISTA", vê-se que é um dado do tipo *float*. Isso significa que é um **ponto flutuante**, ou seja, não é **exatamente 19.51** mas, sim, um número com muitas casas decimais além das que estamos vendo. Em outras palavras, não corresponde completamente à condição que descrevemos.

Para solucionar esse problema e filtrar um valor cravado, bem específico, podemos usar os operadores BETWEEN (entre) e AND (e):

**SELECT** \* **FROM** tabela\_de\_produtos **WHERE** PRECO\_DE\_LISTA **BETWEEN** 19.50 **AND** 19.52;

ssim, selecionamos tudo de "tabela\_de\_produtos" cujo "PRECO\_DE\_LISTA" está entre 19.50 e 19.52. Percebemos que fazer buscas com campos *float* é um pouquinho mais complicado.

Dessa forma, nesse vídeo procurei fazer uma apresentação dos comandos de SELECT (com o asterisco ou especificando campos), usando alguns filtros simples que usam **=\**, \**>**, **≥\**, \**<**, **≤** ou BETWEEN entre dois valores. Foi uma revisão do que vimos no [curso de Introdução ao SQL com MySQL](https://cursos.alura.com.br/course/mysql-manipule-dados-com-sql).