



Invista em você! Saiba como a DevMedia pode ajudar sua carreira. ➔

MySQL Tutorial

Veja neste artigo algumas técnicas para você utilizar as funções básicas (SQL e peculiaridades) do Sistema de Gerenciamento de Banco de Dados Relacional (SGBDR) MySQL.

Artigos > Banco de D... > MySQL T...

MySQL Tutorial

A **Structured Query Language** (em português – Linguagem de Consulta Estruturada) ou chamado pela abreviação SQL, é conhecida comercialmente como uma “linguagem de consulta” padrão utilizada para manipular bases de dados relacionais. Por ser uma linguagem padrão, é utilizada em inúmeros sistemas, como: **MySQL**; SQL Server; Oracle; Sybase; DB2; PostgreSQL; Access e etc.

Cada sistema pode usar um “dialeto” do SQL, como T-SQL (utilizado por SQL SERVER), PL/SQL (Oracle), JET SQL (Access) entre outros.

No entanto, o SQL possui muitos outros recursos além de consulta ao banco de dados, como meios para a definição da estrutura de dados, para modificação de dados no banco de dados e para a especificação de restrições de segurança.

Esses recursos em SQL são divididos em cinco partes, sendo:

1. Data Definition Language (Linguagem de Definição de Dados), conhecido pela abreviação **DDL**;



-
3. Data Query Language (Linguagem de Consulta de Dados), conhecido pela abreviação **DQL**;
 4. Data Control Language (Linguagem de Controle de Dados), conhecido pela abreviação **DCL**;
 5. Data Transaction Language (Linguagem de Transação de Dados), conhecida pela abreviação **DTL**.

Aprenderemos neste artigo sobre os três primeiros recursos dessa lista.

Instalando o MySQL

Para instalar o banco, precisamos acessar o site **oficial do MySQL** (seção **Links**) e clique no botão de download, de acordo com a sua arquitetura (32 ou 64bits).

Ao executar a instalação, escolha a opção “Developer Default”, como na **Figura 1**.

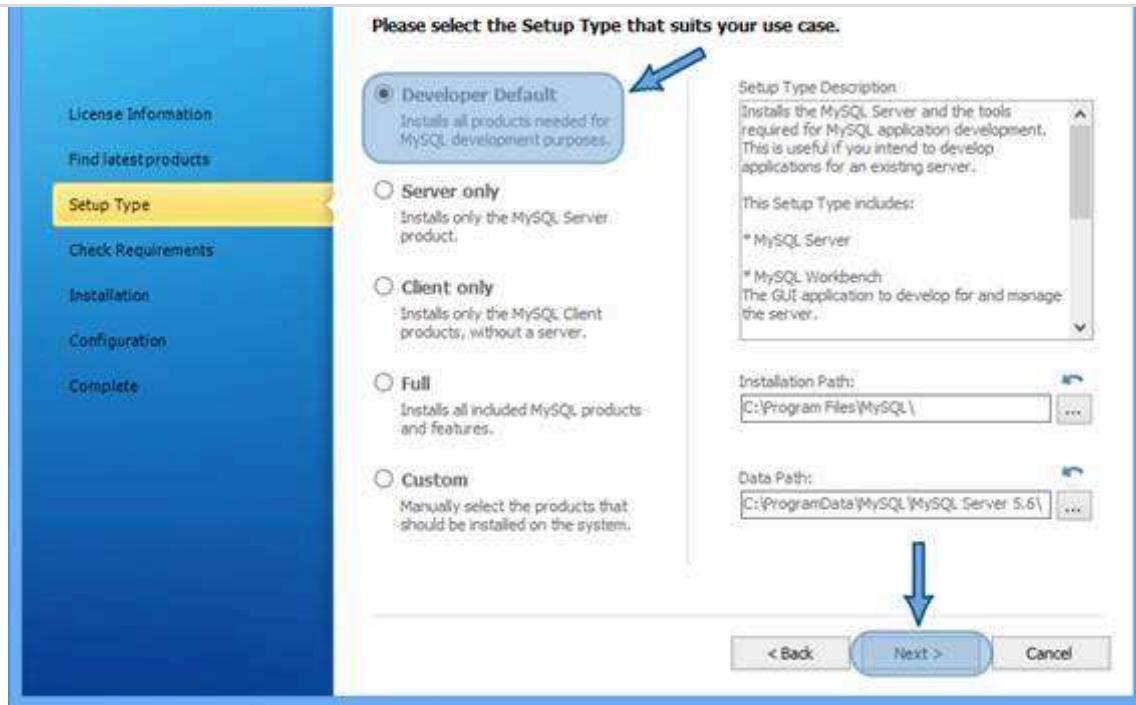


Figura 1. Opção selecionada “Developer Default” e botão “Next”.

Após a instalação dos componentes, aparecerá a mesma tela da **Figura 2** para escolher configurações. Não deixe de marcar a opção “Enable TCP/IP Networking” e colocar a porta 3306.



Figura 2. Escolha das opções demarcadas e botão “Next”.

Em seguida aparecerá a tela de inserção dos dados de login do usuário **root**, como mostra a **Figura 3**.



Figura 3. Definição de senha e botão “Next”.

Na tela seguinte é pedido para definir o nome do seu servidor. Após mais alguns “next” conclui-se a instalação do MySQL.

Podemos utilizá-lo de duas maneiras: A primeira seria utilizando o **MySQL 5.6 Command Line Client**, que tem a mesma aparência do terminal do MS-DOS (**Figura 4**), ou através da ferramenta **MySQL Workbench** (**Figura 5**), que é um **gerenciador do MySQL** com integração ao usuário e totalmente visual.



```
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.  
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.  
mysql> _
```

Figura 4. Aparência do MySQL 5.6 Command Line Client.



Figura 5. Aparência do MySQL 5.6 Workbench.

Utilizando o MySQL Workbench

Para começarmos a trabalhar, vamos nos conectar a nossa DataBase usando a ferramenta gráfica. Para isso, acesse o menu “DataBase” e clique na opção “Connect to Database...”, como na **Figura 6**. Use as configurações padrão vistas na **Figura 7**.



Figura 6. Acessando o menu “Database”.

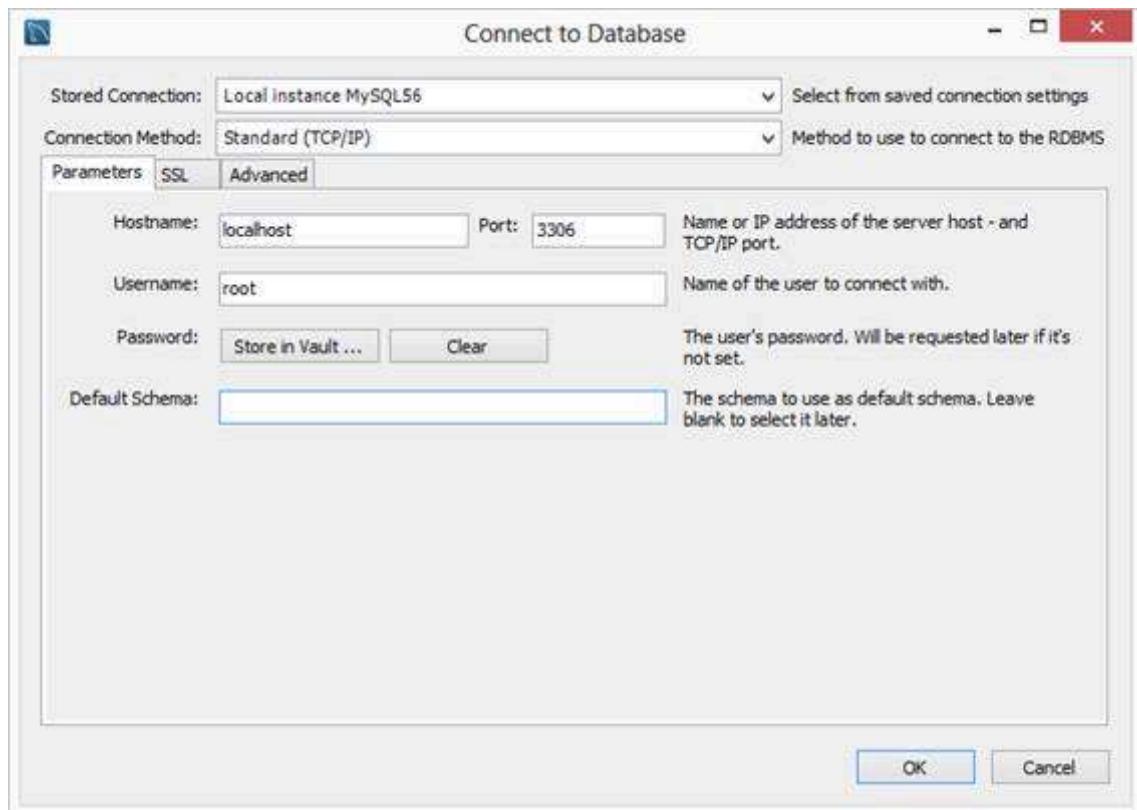


Figura 7. Configurações

Linguagem de Definição de Dados (DDL)



temos para isso a criação e alteração de estruturas que definem como os dados serão armazenados. Logo, quando falamos de comando do tipo `DDL` estamos falando de comandos do tipo `CREATE`, `ALTER` e `DROP` (criar, alterar e excluir, respectivamente).

Para criar o banco de dados `DBDevMedia` utilizaremos a sintaxe `CREATE`, conforme o código a seguir:

```
1 | CREATE DATABASE DBDevMedia;
```

Ao executá-lo teremos o mesmo resultado da **Figura 8**.

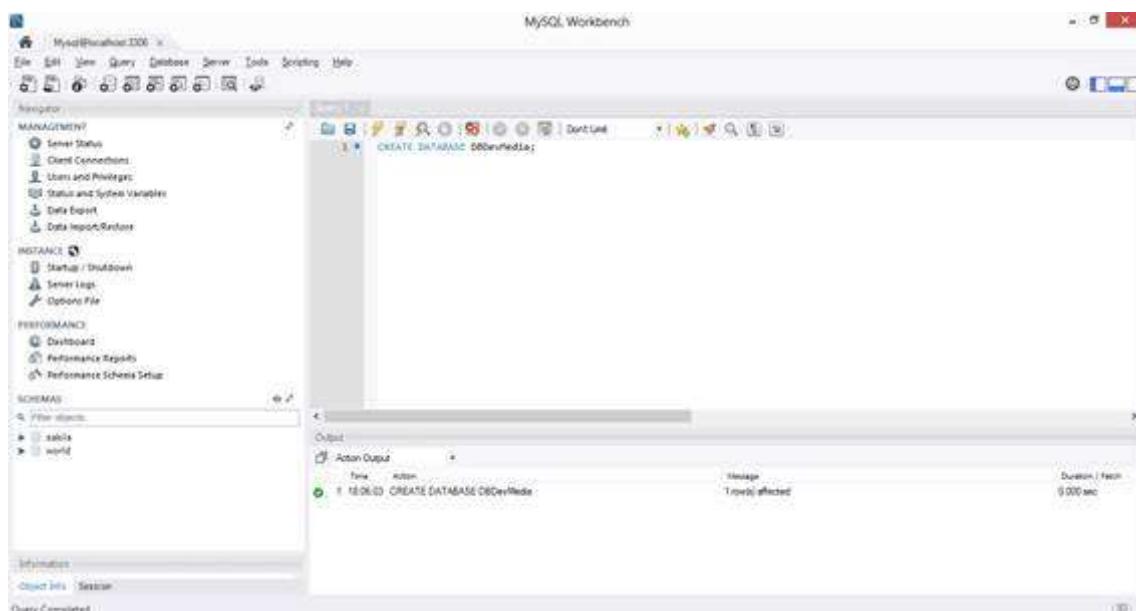


Figura 8. Criando o banco de dados.



servidor, evitando que retorne um erro com a possível existência de dois bancos com o mesmo nome em um mesmo **servidor MySQL**:

```
1 | CREATE DATABASE IF NOT EXISTS DBDevMedia;
```

Para visualizar uma lista com todos os bancos de dados existentes no servidor, use o comando:

```
1 | SHOW DATABASES;
```

Observe que todos os **comandos em MySQL** sempre termina com “;” no final. Essa sintaxe é obrigatória para que o MySQL possa entender o termo do comando.

Para remover os bancos de dados existentes no servidor, utilize o comando a seguir, mas atenção, pois uma vez executado, a ação é **irreversível (Figura 9)**:

```
1 | DROP DATABASE DBDevMedia;
```

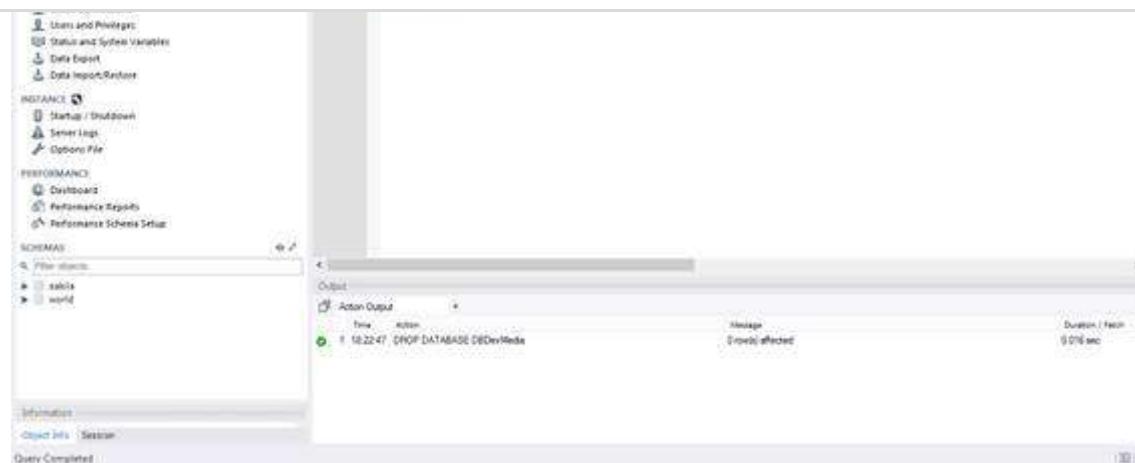


Figura 9. Removendo o banco de dados.

Criando tabelas no MySQL

Dada a grande quantidade de parâmetros aceitos, a declaração `CREATE TABLE` é uma das mais **complexas no MySQL**.

Vamos começar selecionando o banco de dados que ganhará a nova tabela usando a sintaxe:

```
1 | USE DBDevMedia;
```

De acordo com a documentação disponível pela Oracle, a sintaxe simplificada seria:

```
1 | CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
2 | (create_definition, ...)
```



assim que sua **sessão no MySQL** terminar. Use-a sempre que estiver fazendo testes.

- **IF NOT EXISTS:** Verifica a prévia existência da tabela e evita uma interrupção do script causada por erro. Como o MySQL é **case sensitive**, tabelas com nomes iguais, mas usando letras em caixa alta, como em `tbl_name` e `Tbl_name`, são consideradas tabelas totalmente diferentes.

Uma tabela é composta por uma ou mais colunas, cada qual com suas definições.

Vamos começar pela criação de uma agenda telefônica. A tabela contatos terá a seguinte estrutura da **Listagem 1**.

Listagem 1. Tabela Contatos

```
1 | CREATE TABLE contatos (
2 |     nome VARCHAR(50) NOT NULL,
3 |     telefone VARCHAR(25) NOT NULL
4 | );
```

Para verificar se a tabela foi criada use o comando (**Figura 10**):

```
1 | SHOW TABLES;
```



The screenshot shows the MySQL Workbench interface. On the left, there's a sidebar with various tabs like 'Client Connections', 'User and Privileges', 'Status and System Variables', 'Data Export', 'Data Import', 'INSTANCE', 'Startup / Shutdown', 'Server Logs', 'Options Prefs', 'PERFORMANCE', 'Dashboards', 'Performance Reports', and 'Performance Schema Setup'. Below that is the 'SCHEMAS' tree, which includes 'trabalho', 'carr', and 'databasedevmedia', with 'databasedevmedia' expanded to show 'Tables', 'Views', 'Stored Procedures', and 'Functions'. At the bottom of the sidebar are 'Information' and 'Information' buttons. The main area has a title bar 'Result Grid' with a 'Filter Rows' button, 'Export' button, and 'View Cell Content' button. Below the title bar is a table titled 'Tables_in_databasedevmedia' with one row: 'contatos'. To the right of the table is a vertical toolbar with icons for 'Form Editor', 'Grid Editor', and 'Table Editor', and a status bar at the bottom right indicating 'Read Only'. Below the table is a 'Result Set' section with a 'Overview' tab, showing 'Admin Output' with a log entry: 'Time Action Message' (1.09.30.25 SHOW TABLES) and 'Message' (1 rows returned). There's also a 'Duration / Fetch' section showing '0.000 sec / 0.000 sec'.

Figura 10. Listando tabelas existentes.

Podemos melhorar um pouco mais a tabela contatos, ao acrescentar mais alguns campos, como sobrenome dos contatos, DDD, data de nascimento e e-mail. Antes de criar uma nova tabela, com o mesmo nome, vamos remover a anterior usando o comando:

```
1 | DROP TABLE contatos;
```

Agora, vamos criar a nova tabela, conforme a **Listagem 2**.

Listagem 2. Tabela contatos

```
1 | CREATE TABLE IF NOT EXISTS contatos (
2 |     nome VARCHAR(20) NOT NULL,
3 |     sobrenome VARCHAR(30) NOT NULL,
4 |     ddd INT(2) NOT NULL,
5 |     telefone VARCHAR(9) NOT NULL,
6 | );
```



A chave primária é o que torna a linha ou o registro de uma tabela único. Geralmente, é utilizada uma sequência automática para a geração dessa chave para que ela não venha a se repetir. Em nosso caso, o nro_contato será único, com uma sequência numérica que identificará o registro.

A cláusula `auto_increment` é utilizada para incrementar automaticamente o valor da chave primária. Por padrão, essa cláusula inicia com 1. Porém, se houver a necessidade de iniciar por outro valor, podemos fazer como no exemplo a seguir:

```
1 | CREATE TABLE contatos AUTO_INCREMENT=100;
```

ALTER TABLE

Imagine que sua tabela já contenha dados armazenados e você precisa acrescentar mais um campo (chamado Ativo) na tabela de contatos.

Conhecidamente pensariamos em usar o *drop table* para excluir a tabela e recriá-la com o novo campo, mas perder os dados é algo inviável.

Nossa solução é utilizar a sintaxe `ALTER TABLE`, que permite alterar a estrutura da tabela existente. Por exemplo, você pode adicionar ou deletar colunas, criar ou remover índices, alterar o tipo de coluna existentes, ou renomear coluna ou tabelas. Você também pode alterar o comentário para a tabela e tipo de tabela.

Para adicionar colunas use o comando `ADD`, seguido do nome e dos atributos da coluna que será adicionada e, da sua posição dentro da tabela com o auxílio do



```
1 | ALTER TABLE contatos  
2 | ADD ativo SMALLINT NOT NULL AFTER email;
```

Para ver o resultado das alterações, dê o comando:

```
1 | DESCRIBE contatos;
```

Para alterar os atributos e nome de colunas usamos o parâmetro `CHANGE`, seguido da denominação da coluna a ser alterada e dos novos atributos. Para mudar os atributos da coluna nome, utilizaremos a seguinte sintaxe:

```
1 | ALTER TABLE contatos  
2 | CHANGE telefone telefone CHAR(9) NOT NULL;
```

Vocês devem ter percebido que a palavra “telefone” foi utilizada duas vezes. Isso ocorre porque se indica primeiro a coluna e depois seus novos atributos, e o nome da coluna é um de seus atributos.

Para mudar o nome da coluna e manter seus demais atributos usamos a sintaxe a seguir:



Linguagem de Manipulação de Dados (DML) e Linguagem de Transação de Dados (DTL)

Inserindo registros

Depois da tabela pronta precisamos agora de registros em nosso banco de dados. Para esse exemplo não vamos usar nenhuma aplicação para inserir esses dados, mas sim diretamente pelo SGBD através de comando SQL.

Vamos fazer o primeiro `INSERT` na tabela contatos com o comando `INSERT INTO` contatos. Entre parênteses informaremos em quais colunas queremos inserir os registros e depois devemos informar qual o valor para cada coluna, como mostra a **Listagem 3**.

Listagem 3. Inserindo dados

```
1 | INSERT INTO contatos (nome
2 |   ,sobrenome
3 |   ,ddd
4 |   ,telefone
5 |   ,data_nasc
6 |   ,email
7 |   ,ativo)
8 | VALUES('Bruno'
9 |   , 'Santos'
10|   , 11
11|   , 999999999
12|   , '2015-08-22'
13|   , ' contato@dominio.com.br'
14|   , 1);
```



sequência correta, como na **Listagem 4**, onde omitimos estes campos. O SGBD subentende que todos os campos serão populados.

Listagem 4. Inserindo dados sem descrever

```
1 | INSERT INTO contatos VALUES('Bruno'
2 | , 'Santos'
3 | , 11
4 | , 99999999
5 | , '2015-08-22'
6 | , ' contato@dominio.com.br'
7 | , 1);
```

Observe que em nenhum momento foi mencionado o campo nro_contato ou acrescentado um valor diretamente, isso por que este campo foi definido como auto_increment, desta forma, o campo recebe o valor automaticamente.

Alterando registros

Para alterar os registros usamos o comando `UPDATE`.

No exemplo anterior inserimos um sobrenome errado. Para corrigir usamos a sintaxe da **Listagem 5**.

Listagem 5. Alterando dados

```
1 | UPDATE contatos SET
2 | sobrenome= 'Nascimento' WHERE nro_contato= 100;
3 | commit;
```



Listagem 6. Alterando mais de um dado

```
1 | UPDATE contatos SET
2 | sobrenome= 'Nascimento'
3 | , ddd= 015
4 | , telefone= '0123456789'
5 | WHERE nro_contato = 100
6 | commit;
```

Perceba que, além do `UPDATE` utilizamos o `SET` para informar qual campo que queremos alterar. O `WHERE` indica a condição para fazer a alteração e, em seguida, o `commit` diz ao SGBD que ele pode realmente salvar a alteração do registro. Se, por engano, fizermos o `UPDATE` incorreto, antes do `commit` podemos reverter a situação usando a instrução SQL `rollback`, da seguinte maneira:

```
1 | UPDATE contatos SET
2 | sobrenome= 'Nascimento' WHERE nro_contato= 100;
3 | rollback;
```

Com isso, o nosso SGBD vai reverter a última instrução. Porém, se tiver a intenção de utilizar o `rollback`, faça-o antes de aplicar o `commit`, pois se você aplicar o `UPDATE` ou qualquer outro comando que necessite do `commit`, não será possível reverter.

As instruções `commit` e `rollback` são tratadas pela **Linguagem de Transação de Dados (DTL)**.



Para deletar algum registro usamos a instrução SQL `DELETE`. Diferente do `DROP`, ele deleta os registros das colunas do banco de dados.

O `DROP` é usado para excluir objetos do banco, como tabelas, colunas, *views* e *procedures*, enquanto, o `delete` deletará os registros das tabelas, podendo excluir apenas uma linha ou todos os registros. Desta maneira, vamos apagar o primeiro registro da tabela `contatos` usando o seguinte comando:

```
1 | DELETE FROM contatos WHERE nro_contato= 100;  
2 | commit;
```

Para deletar todos os registros da tabela de clientes usamos o comando:

```
1 | DELETE FROM contatos;  
2 | commit;
```

Observe que, ao empregar o `DELETE` você também deve usar o `commit` logo após a instrução. Da mesma maneira, podemos também utilizar o `rollback` para não efetivar uma exclusão de dados incorretos.

Além do `DELETE`, podemos eliminar os dados usando a instrução SQL `TRUNCATE`, que não necessita de `commit`. Nem o `rollback` pode reverter à operação.

Isso ocorre porque, quando você utiliza o `DELETE`, o SGBD salva os seus dados em uma tabela temporária e, quando aplicamos o `rollback`, ele a consulta e restaura



```
1 | TRUNCATE TABLE contatos;
```

Essa instrução não pode ser usada dentro da cláusula `WHERE`.

Linguagem de Consulta de Dados (DQL)

O objetivo de armazenar registros em um banco de dados é a possibilidade de recuperar e utilizá-los em relatórios para análises mais profundas. Essa recuperação é feita através de consultas.

O comando SQL utilizado para fazer consultas é o `SELECT`. Selecionando os dados, devemos dizer ao SGBD de onde queremos selecionar, através do comando `FROM`.

Como exemplo, vamos selecionar os registros da tabela de contato (**Figura 11**).

Quando não queremos selecionar um ou vários campos específicos, utilizamos o asterisco (*):

```
1 | SELECT * FROM contatos;
```

Figura 11. Consultando contatos.



```
1 | SELECT nome, sobrenome FROM contatos;
```

Figura 12. Consultando o nome e o sobrenome da tabela contatos.

Ainda podem surgir situações que necessitem selecionar apenas um registro.

Neste caso, utilizamos o WHERE

Vamos selecionar o cliente com uma cláusula que deve ter `nro_contato= 101`:

```
1 | SELECT nome, sobrenome  
2 | FROM contatos  
3 | WHERE nro_contato= 100;
```

Para colunas do tipo texto será necessário colocar o valor entre aspas simples, assim dizemos ao SGBD que estamos querendo comparar o valor com uma coluna do tipo texto:

```
1 | SELECT nome, sobrenome  
2 | FROM contatos  
3 |
```



E se quiséssemos todos os clientes que sejam diferentes de ‘100’? Faríamos uma consulta utilizando o **operador do MySQL** diferente `<>` (**Figura 13**):

```
1 | SELECT nome, sobrenome  
2 | FROM contatos  
3 | WHERE nro_contato <> 100;
```

Figura 13. Utilizando a clausula “WHERE nro_contato <> 100”.

Além dos operadores de comparação `=` e `<>`, temos os seguintes operadores:

- `>`: maior;
- `<`: menor;
- `>=`: maior e igual;
- `<=`: menor e igual.

A clausula `DISTINCT` retorna apenas uma linha de dados para todo o grupo de linhas que tenha o mesmo valor. Por exemplo, executando a consulta a seguir:

```
1 | SELECT DISTINCT sobrenome FROM contatos;
```



```
1 | Santos
2 | Carvalho
3 | Silva
```

Já a clausula ALL é o oposto de DISTINCT, pois retorna todos os dados. Observe a consulta a seguir:

```
1 | SELECT ALL sobrenome FROM contatos;
```

Repare que o resultado a seguir apresenta o sobrenome Santos duas vezes:

```
1 | Santos
2 | Carvalho
3 | Santos
4 | Silva
```

A clausula ORDER BY retorna os comandos em ordem ascendente (ASC) ou descendente (DESC), sendo o padrão ascendente. Vejamos um exemplo:

```
1 | SELECT nome FROM contatos ORDER BY nome DESC;
```



```
1 | Isabelle  
2 | Elaine  
3 | Cauã  
4 | Bruno
```

A clausula *LIMIT [inicio,] linhas* retorna o número de linhas especificado. Se o valor *inicio* for fornecido, aquelas linhas são puladas antes do dado ser retornado. Lembre-se que a primeira linha é 0.

```
1 | SELECT * FROM contatos LIMIT 3,1;
```

O resultado da consulta será:

```
1 | 103 Isabelle Silva 11 999999999 2013-11-20 contato@rh.com.br
```

Para incrementar as consultas podemos usar algumas funções. A seguir apresentaremos as mais comuns:

- A função ABS retorna o valor absoluto do número, ou seja, só considera a parte numérica, não se importando com o sinal de positivo ou negativo do mesmo. Por exemplo: ABS(-145) retorna 145;
- A função BIN considera o binário de número decimal. Por exemplo: BIN(8) retorna 1000;



- A função `CURTIME()` / `CURRENTTIME()` retorna a hora atual na forma `HH:MM:SS`. Por exemplo: `CURTIME()` retorna `13:02:43`;
- A função `DATABASE` retorna o nome do banco de dados atual: Por exemplo: `DATABASE()` retorna `DBDevMedia`;
- A função `DAYOFMONTH` retorna o dia do mês para a data dada, na faixa de 1 a 31. Por exemplo: `DAYOFMONTH ("2004-04-04")` retorna 04;
- A função `DAYNAME` retorna o dia da semana para a data dada. Por exemplo: `DAYNAME ("2004-04-04")` retorna `Sunday`;
- A função `DAYOFWEEK` retorna o dia da semana em número para a data dada, na faixa de 1 a 7, onde o 1 é domingo. Por exemplo: `DAYOFWEEK ("2004-04-04")` retorna 1;
- A função `DAYOFYEAR` retorna o dia do ano para a data dada, na faixa de 1 até 366. Por exemplo: `DAYOFYEAR ("2004-04-04")` retorna 95;
- A função `FORMAT (NÚMERO, DECIMAIS)` formata o número nitidamente com o número de decimais dado. Por exemplo: `FORMAT (5543.00245,2)` retorna `5.543.002,45`

A função `LIKE` merece um destaque especial, pois faz uma busca sofisticada por uma substring dentro de uma string informada. Temos, dentro da função `LIKE`, os seguintes caracteres especiais utilizados em substrings:

- `%`: busca zero ou mais caracteres;
- `_`: busca somente um caractere.

Vamos a alguns exemplos:

```
1 | SELECT nome From contatos Where nome like 'B%';
```



apenasBruno.

```
1 | SELECT nome From contato Where nome like '_a%';
```

O caractere ‘_’ na consulta indica que estamos procurando nomes nos quais a letra A é a segunda letra do nome, ou seja, o retorno será apenas Cauã.

```
1 | SELECT nome From contato Where nome like '%o';
```

A consulta buscou nomes em que a última letra é o caractere ‘O’, ou seja, teremos como retorno apenas Bruno.

Outra função importante para retorno de consultas é Left, que retorna os primeiros caracteres à esquerda de uma string. Sua sintaxe é apresentada a seguir:

```
1 | LEFT(string,tamanho)
```

A consulta a seguir retornará os três primeiros caracteres à esquerda dos registros da coluna nome:



O resultado será:

1	Bru
2	Ela
3	Cau
4	Isa

A função Right é semelhante a função Left, mas esta retorna os últimos caracteres à direita de uma string. Sua sintaxe também é semelhante:

```
1 | RIGHT(string1,tamanho)
```

Repare que na consulta a seguir são retornados os quatro últimos caracteres à direita dos nomes da tabela contatos:

```
1 | SELECT RIGHT(nome,4) From contatos;
```

O resultado será:



3 | caud
4 | ele

Espero que tenham gostado e até a próxima!

Links:

- [Documentação do MySQL](#)
- [MySQL](#)

Confira também

Curso de MySQL

Curso

Introdução prática
ao comando SQL
SELECT

Curso

Avançando com
Subqueries

Curso

Confira outros conteúdos:





Por Bruno

Em 2015

Comentários nesta publicação

[Escrever um comentário sobre conteúdo](#)

<Formação completa Programador FullStack/>

- ✓ Conteúdo Front-end, Back-end e Mobile
- ✓ Plano de estudo linear
- ✓ +10 mil exercícios gamificados
- ✓ +50 projetos reais
- ✓ Comunidade com + 200 mil alunos



Comece agora

Perguntas Frequentes

Quem somos?

Por que a programação se tornou a profissão mais promissora da atualidade?

Como faço para começar a estudar?

Em quanto tempo de estudo vou me tornar um programador?

Sim, você pode se tornar um programador e não precisa ter diploma de curso superior!

O que eu irei aprender estudando pela DevMedia?

Principais diferenciais da DevMedia



Como funciona a forma de pagamento da DevMedia?

Nossos casos de sucesso

Leonardo Carlos



Eu sabia pouquíssimas coisas de programação antes de começar a estudar com vocês, fui me especializando em várias áreas e ferramentas que tinham na plataforma, e com essa bagagem **consegui um estágio logo no início do meu primeiro período na faculdade.**

Lucas Rodrigues



Estudo aqui na Dev desde o meio do ano passado! Nesse período a Dev me ajudou a crescer muito aqui no trampo.

Fui o primeiro desenvolvedor contratado pela minha empresa. Hoje eu lidero um time de desenvolvimento!

Minha meta é continuar estudando e praticando para ser um Full-Stack Dev!

Heráclito Júnior



Economizei 3 meses para assinar a plataforma e sendo sincero valeu muito a pena, pois a **plataforma é bem intuitiva e muuuuito didática a metodologia de ensino.** Sinto que estou EVOLUINDO a cada dia. Muito obrigado!

Julio Cahlen



Nossa! Plataforma maravilhosa. To amando o curso de desenvolvimento front-end, tinha coisas que eu ainda não tinha visto. **A didática é do jeito**

**Joelberth Sena**

Adquiri o curso de vocês e logo percebi que são os melhores do Brasil. É um passo a passo incrível. **Só não aprende quem não quer. Foi o melhor investimento da minha vida!**

Felipe Nunes

Foi um dos melhores investimentos que já fiz na vida e tenho aprendido bastante com a plataforma. Vocês estão fazendo parte da minha jornada nesse mundo da programação, **irei assinar meu contrato como programador graças a plataforma.**

Wanderson Oliveira

Comprei a assinatura tem uma semana, aprendi mais do que 4 meses estudando outros cursos. Exercícios práticos que não tem como não aprender, estão de parabéns!

José Lucas

Obrigado DevMedia, nunca presenciei **uma plataforma de ensino tão presente na vida acadêmica de seus alunos**, parabéns!

Eduardo Dorneles

Aprendi React na plataforma da DevMedia há cerca de 1 ano e meio... **Hoje estou há 1 ano empregado** trabalhando 100% com React!

Adauto Junior



[Ver todos os casos de sucesso](#)

Menu

[Assine agora](#)

Hospedagem web por Porta 80 Web Hosting.

[Quem somos](#)

[FAQ - Fale conosco](#)

[Plano para Instituição de ensino](#)

[Assinatura para empresas](#)

[Política de privacidade](#)

[Termos de uso](#)

[Política de estorno](#)



DevMedia: 08.401.613/0001-42

Rua Victor Civita, 66 - Salas 306, 307 e 308 -

Jacarepaguá

Rio de Janeiro - RJ, 22775-044

Tecnologia:

HTML CSS Algoritmo Javascript React React Native Node.js SQL MySQL UML Scrum
Levantamento de Requisitos Padrão de Projeto Teste de Software C# Delphi Dart Java Kotlin
PHP Python TypeScript Angular Vue.js Django Laravel Spring .NET Flutter Modelagem de
Dados Oracle REST PostgreSQL SQL Server MVC Orientação a Objeto Docker Git Scrum

Cursos:

HTML e CSS Javascript Programação para Iniciantes Angular React Vue.js Node.js Spring .NET
Core Mobile React Native Android Flutter Algoritmo Automação Delphi Java PHP Python
SQL e Banco de Dados Engenharia de Software Canal Mais Gratuitos

Artigos:

Front-End Javascript Iniciantes Angular Dart Engenharia Mobile Node.js Python React Native
Vue.js Android Banco de Dados Delphi Flutter Java Kotlin .Net PHP React Spring
Gratuitos

DevCast:

**Guia:**

Fundamentos .NET PHP Python Java Delphi HTML e CSS JavaScript Node React Native
Flutter Banco de Dados Mobile Spring Arquitetura Automação Engenharia + Assuntos