

MySQL

Índice

Advertência.....	2
Autor.....	3
Recomendações.....	4
1 – Introdução.....	5
2 - Gerenciadores do MySQL.....	8
2.1 – Console.....	8
2.2 – Adminer.....	9
2.3- phpMyAdmin.....	12
3 - Gerenciamento de bancos.....	17
3.1 – DDL.....	18
3.1.1 - Criando um banco de dados.....	18
3.2 – DML.....	20
3.2.1 – Like.....	25
3.2.2 – LIMIT.....	26
4 - Tipos de dados.....	32
5 - Funções nativas do MySQL.....	34
6 - Relacionamentos entre tabelas (introdução).....	39
7 - Usuários e Privilégios.....	40
8 – Views (introdução).....	42
9 – Dicas.....	43
10 – Metadados.....	46

Advertência

Esta apostila é fruto de pesquisa por vários sites e autores e com alguma contribuição pessoal. Geralmente o devido crédito é concedido, exceto quando eu não mais encontro o site de origem. Isso indica que esta apostila foi criada, na prática, "a várias mãos" e não somente por mim. Caso identifique algo que seja de sua autoria e não tenha o devido crédito, poderá entrar em contato comigo para dar o crédito ou para remover: ribafs @ gmail.com (remova os dois espaços).

É importante ressaltar que esta apostila é distribuída gratuitamente no repositório:

<https://github.com/ribafs/apostilas>

Sob a licença MIT. Fique livre para usar e distribuir para qualquer finalidade, desde que mantendo o crédito ao autor.

Sugestões

Sugestões serão bem vindas, assim como aviso de erros no português ou no MySQL. Crie um issue no repositório ou me envie um e-mail ribafs @ gmail.com.

Autor

Ribamar FS

ribafs @ gmail.com

<https://ribamar.net.br>

<https://github.com/ribafs>

Fortaleza, 11 de dezembro de 2021

Recomendações

O conhecimento teórico é importante para que entendamos como as coisas funcionam e sua origem, mas para consolidar um conhecimento e nos dar segurança precisa de prática e muita prática.

Não vale muito a penas ser apenas mais um programador. É importante e útil aprender muito, muito mesmo, ao ponto de sentir forte segurança do que sabe e começar a transmitir isso para os demais.

Tenha seu ambiente de programação em seu desktop para testar com praticidade todo o código sugerido.

Caso esteja iniciando com MySQL recomendo que leia por inteiro e em sequência. Mas se já entende algo dê uma olhada no índice e vá aos assuntos que talvez ainda não domine.

Não esqueça de ver e testar também o conteúdo da pasta Anexos.

Caso tenha alguma dúvida, me parece que a forma mais prática de receber resposta é através de grupos. Temos também o Google que geralmente ajuda.

Dica: quando tiver uma dúvida não corra para pedir ajuda. Antes analise o problema, empenhe-se em entender o contexto e procure você mesmo resolver. Assim você está passando a responsabilidade para si mesmo, para seu cérebro, que passará a ser mais confiante e poderá te ajudar ainda mais. É muito importante que você confie em si mesmo, que é capaz de solucionar os problemas. Isso vai te ajudar. Somente depois de tentar bastante então procure ajuda.

Veja que este material não tem vídeos. Isso em si nos nossos dias é algo que atrai pouca gente, pois vídeos e fotos são mais confortáveis de ler e acompanhar. Ler um texto como este requer mais motivação e empenho. Lembrando que para ser um bom programador precisamos ser daqueles capaz de se empenhar e suportar a leitura e escrita.

Autodidata

Não tive a pretensão de que esta apostila fosse completa, contendo tudo sobre MySQL, que seria praticamente impossível. Aquele programador que quando não sabe algo seja capaz de pesquisar, estudar, testar e praticamente sozinho. Este é um profissional importante para as empresas e procurado.

1 – Introdução

O MySQL é um sistema de gerenciamento de banco de dados (SGBD), que utiliza a linguagem SQL (Linguagem de Consulta Estruturada, do inglês Structured Query Language) como interface. É atualmente um dos bancos de dados mais populares, com mais de 10 milhões de instalações pelo mundo.

Entre os usuários do banco de dados MySQL estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S Army, US. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems, Google CanaVialis S.A. e outros.

História

O MySQL foi criado na Suécia por dois suecos e um finlandês: David Axmark, Allan Larsson e Michael "Monty" Widenius, que têm trabalhado juntos desde a década de 1980. Hoje seu desenvolvimento e manutenção empregam aproximadamente 400 profissionais no mundo inteiro, e mais de mil contribuem testando o software, integrando-o a outros produtos, e escrevendo a respeito dele.

No dia 16 de Janeiro de 2008, a MySQL AB, desenvolvedora do MySQL foi adquirida pela Sun Microsystems, por US\$ 1 bilhão, um preço jamais visto no setor de licenças livres. No dia 20 de Abril de 2009 a Oracle compra a Sun Microsystems e todos o seu produtos, incluindo o MySQL. A Comissão Europeia ainda não aprovou a aquisição[3].

O sucesso do MySQL deve-se em grande medida à fácil integração com o PHP incluído, quase que obrigatoriamente, nos pacotes de hospedagem de sites da Internet oferecidos atualmente. Empresas como Yahoo! Finance, MP3.com, Motorola, NASA, Silicon Graphics e Texas Instruments usam o MySQL em aplicações de missão crítica[4]. A Wikipédia é um exemplo de utilização do MySQL em sites de grande audiência.

O MySQL hoje suporta Unicode, Full Text Indexes, replicação, Hot Backup, GIS, OLAP e muitos outros recursos.

Características

- Portabilidade (suporta praticamente qualquer plataforma atual);

- Compatibilidade (existem drivers ODBC, JDBC e .NET e módulos de interface para diversas linguagens de programação, como Delphi, Java, C/C++, Python, Perl, PHP, ASP e Ruby)

- Excelente desempenho e estabilidade;

- Pouco exigente quanto a recursos de hardware;

- Facilidade de uso;

- É um Software Livre com base na GPL;

- Contempla a utilização de vários Storage Engines como MyISAM, InnoDB, Falcon, BDB, Archive, Federated, CSV, Solid...

Suporta controle transacional;

Suporta Triggers;

Suporta Cursors (Non-Scrollable e Non-Updatable);

Suporta Stored Procedures e Functions;

Replicação facilmente configurável;

Interfaces gráficas (MySQL Toolkit) de fácil utilização cedidos pela MySQL Inc.

Utilização

Uma característica fundamental do MySQL, quicá na origem do seu sucesso, é ser desenvolvido em código aberto e funcionar num grande número de sistemas operacionais : Windows, Linux, FreeBSD, BSDI, Solaris, Mac OS X, SunOS, SGI, etc.

É reconhecido pelo seu desempenho e robustez e também por ser multi-tarefa e multi-usuário. A própria Wikipédia, usando o programa MediaWiki, utiliza o MySQL para gerenciar seu banco de dados, demonstrando que é possível utilizá-lo em sistemas de produção de alta exigência e em aplicações sofisticadas.

No passado (até à versão 3.x), devido a não possuir funcionalidades consideradas essenciais em muitas áreas, como stored procedures, two-phase commit, subselects, foreign keys ou integridade referencial, era frequentemente considerado um sistema mais "leve" e para aplicações menos exigentes, sendo preterido por outros sistemas como o PostgreSQL.

Fonte: Wikipédia

https://www.oficinadanet.com.br/artigo/2227/mysql_-_o_que_e

Quando a Oracle comprou o MySQL então a comunidade criou um fork dele, chamado MariaDb.

Sobre o Mariadb

MariaDB é baseado no MySQL e está disponível sob os termos da licença GPL v2.

É desenvolvido pela Comunidade MariaDB com o principal administrador a Monty Program Ab.

MariaDB é mantido atualizado com a última versão do MySQL.

Na maioria dos aspectos o MariaDB vai funcionar exatamente como o MySQL: todos os comandos, interfaces, bibliotecas e APIs que existem no MySQL também existem no MariaDB. Não há nenhuma necessidade de converter um bancos de dados para migrar para o MariaDB. MariaDB é um verdadeiro substituto para o MySQL! Além disso, o MariaDB tem um monte de novas funcionalidades agradáveis que você pode aproveitar.

Veja o FAQ do MariaDB para mais informações.

A atual versão estável do MariaDB é a MariaDB 5.2.

A versão anterior a estável é a MariaDB 5.1.

A versão de desenvolvimento é MariaDB 5.3, que agora é alfa, mas deve ser declarado beta em breve.

<https://mariadb.com/kb/pt-br/sobre-o-mariadb/>

O MariaDB é um dos bancos de dados mais conhecidos do mundo, criado pelos mesmos desenvolvedores do MySQL, que mantiveram a estrutura de código aberto. Sua principal característica é a rapidez, escalabilidade e robustez de suas ferramentas, plugins e, claro, capacidade de armazenamento.

<https://rockcontent.com/br/blog/mariadb/>

Atualmente a maioria das distribuições Linux trazem em seus repositórios o MariaDb, assim como o Xampp e outros pacotes também.

2 - Gerenciadores do MySQL

2.1 – Console

Gerenciamento dos bancos do MySQL via console

Ao instalar o MySQL ele já vem com um recurso para gerenciamento dos seus bancos, que é a console.

Ao executar seu binário via terminal/prompt ele nos permite executar vários comandos para isso.

No Windows com Xampp

```
cd c:\xampp\mysql
```

```
bin\mysql -uroot
```

No Linux geralmente ele está no path

```
mysql -uroot -psenha
```

Mostrar todos os bancos de dados

```
SHOW DATABASES;
```

Mostrar detalhes da criação de um banco

```
SHOW CREATE DATABASE novo_banco;
```

Acessar um certo banco

```
USE NomeBanco;
```

Mostrar todas as tabelas do banco atual

```
SHOW TABLES;
```

Mostrar estrutura de uma tabela

```
DESCRIBE clientes;
```

Mostrar estrutura na forma de script de criação

```
SHOW CREATE TABLE clientes;
```


2.2 – Adminer

Um gerenciador web para o MySQL. Não é o mais popular mas é o meu preferido, por ser bem leve, ser apenas um arquivo PHP e conter todos os recursos de que preciso. Veja se te agrada.

Download

<https://www.adminer.org/>

Faça o download do arquivo e copie para o diretório web. Eu gosto de renomear para adminer.php, mas fique a vontade.

Depois abra no navegador

<http://localhost/adminer.php>



Idioma: Português (Brazil) ▼ MySQL » Servidor Sair

Adminer 4.8.1

DB: ▼

Comando SQL Importar Exportar

Selecionar Base de dados

Criar Base de dados Privilégios Lista de processos Variáveis Estado

Versão MySQL: **5.5.5-10.3.32-MariaDB-0ubuntu0.20.04.1** através da extensão PHP **MySQLi**

Logado como: **root@localhost**

	Base de dados - Atualizar	Colação	Tabelas	Size - Compute
<input type="checkbox"/>	information_schema	utf8_general_ci	?	?
<input type="checkbox"/>	laravel	utf8mb4_unicode_ci	?	?
<input type="checkbox"/>	mysql	utf8mb4_general_ci	?	?
<input type="checkbox"/>	novo_banco	utf8mb4_general_ci	?	?
<input type="checkbox"/>	performance_schema	utf8_general_ci	?	?
<input type="checkbox"/>	phpecia	utf8mb4_unicode_ci	?	?

Selected (0)

Adicionar

Uma grande vantagem deste para mim é que suporta também o PostgreSQL e vários outros SGBDs.



Saída bem sucedida. Thanks for using Adminer, consider donating.

Sistema	PostgreSQL ▼
Servidor	localhost
Usuário	
Senha	
Base de dados	

Entrar ☐ Login permanente

Na tela de login podemos escolher o SGBD.

No painel da esquerda temos as tabelas do banco selecionado, que por padrão é nenhuma. Acima temos um menu simples, que inicia com Criar base de dados. Ao centro temos a listagem dos bancos.

Clique no banco novo_banco

Veja que à esquerda aparece o nome da tabela e a esquerda a palavra selecionar:

- Ao clicar no nome da tabela vemos sua estrutura no centro
- Ao clicar em selecionar vemos os registros da tabela

Alterar estrutura de uma tabela

Ao clicar no nome da tabela à esquerda aparece

Tabela: clientes

[Selecionar dados](#) **[Mostrar estrutura](#)** [Alterar estrutura](#) [Novo Registro](#)

Coluna	Tipo	Comentário
id	mediumint(8) unsigned <i>Incremento Automático</i>	
nome	varchar(255) NULL	
email	varchar(255) NULL	
nascimento	varchar(255) NULL	
cpf	varchar(255) NULL	

Índices

PRIMARY *id*

[Alterar índices](#)

Chaves estrangeiras

[Adicionar Chave Estrangeira](#)

Podemos:

- Alterar a estrutura
- Adicionar um novo registro
- Alterar índices
- Adicionar chave estrangeira

À esquerda também temos:

[Comando SQL](#) [Importar](#)
[Exportar](#) [Criar tabela](#)

[selecionar **clientes**](#)

- Acesso ao painel SQL
- Importar script
- Exportar para script
- Criar tabela

Quando vamos selecionando os recursos o menu acima se altera

MySQL » Servidor » novo_banco » Tabela: clientes

Tabela: clientes

Ao clicar em Servidor ele mostra a listagem dos bancos e podemos também criar um novo banco.

Criar um novo banco

MySQL » Servidor » Criar Base de dados

Criar Base de dados

meu_banco utf8mb4_unicode_ci ? Salvar +

O phpMyAdmin já traz este como default, já o Adminer traz a caixa vazia. Tenha o cuidado de selecionar utf8mb4_unicode_ci para atender ao grandes frameworks também.

Criar nova Tabela

- Selecione o banco onde deseja criar a tabela
- À esquerda clique em Criar tabela

Criar tabela

Nome da tabela: produtos (motor) (collation) Salvar

Nome da coluna	Tipo	Tamanho	Opções	NULL	AI?	+
id	int			<input type="checkbox"/>	<input checked="" type="radio"/>	+ ↑ ↓ ×
descricao	varchar	30	(collation)	<input type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×
preco	decimal	8,2		<input type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×
	int			<input type="checkbox"/>	<input type="radio"/>	+ ↑ ↓ ×

Incremento Automático: ☐ Valores padrões ☐ Comentário

Salvar

Criar os campos e clicar no botão + para adicionar novo campo e finalmente clicar em Salvar

Então mostra

Tabela: produtos

A Tabela foi criada. 07:57:30 Comando SQL

Selecionar dados **Mostrar estrutura** Alterar estrutura Novo Registro

Coluna	Tipo	Comentário
id	int(11) <i>Incremento Automático</i>	
descricao	varchar(30)	
preco	decimal(8,2)	

Inserir no novo Registro na tabela criada

Temos um botão para criar graficamente um novo registro mas vamos criar via Comando SQL:

```
insert into produtos (descricao, preco) values ('Abacate', 2.85);
```

Atualizar o registro criado

```
update produtos set preco = 3.65 where descricao = 'Abacate';
```

Excluir um registro

```
delete from produtos where descricao = 'Abacate';
```

2.3- phpMyAdmin

Este é o gerenciador web do MySQL mais popular, pois já acompanha o Xampp e outros pacotes similares, como também por estar disponível como pacote para o Linux.

Caso queira baixar e instalar

<https://www.phpmyadmin.net/>

Basta descompactar e mover a pasta para seu diretório web.

Ao chamar pelo navegador

<http://localhost/phpmyadmin>

Teremos



Painel da Esquerda

Este painel mostra os nomes dos bancos de dados existentes no MySQL, tanto os bancos nativos quanto os bancos criados pelo programador.

Menu Principal

Acima temos o menu principal, que começa com Bancos de Dados, SQL, etc.

Criar um Novo banco de dados

Podemos clicar no botão Novo à esquerda ou em Bancos de dados no menu.

Bancos de dados

 **Criar banco de dados** 

Entre o nome, tenha cuidado de deixar "utf8mb4_general_ci" e clique em Criar


Observe que após criar um banco ele já abre com o banco selecionado e pronto para criar uma nova tabela.

Criar uma nova tabela

As tabelas são o centro dos bancos de dados, pois são elas que guardam as informações em seus campos.

Para criar uma nova tabela basta selecionar à esquerda o banco onde deseja criar, que irá aparecer abaixo a opção para criar tabela.

Como o nosso banco novo_banco selecionado, entre

 **Criar tabela**

Nome: Número de colunas:


Executar


Vamos criar a tabela clientes com 3 campos. Então clique em Executar
Então ele abre a janela para a criação dos 3 campos, como abaixo.










Nome	Tipo	Tamanho/Valores	Padrão	Colação	Atributos	Nulo	Índice
<input type="text" value="id"/>	<input type="text" value="INT"/>	<input type="text"/>	<input type="text" value="Nenhum"/>	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="text" value="PRIMARY"/>
<input type="text" value="nome"/>	<input type="text" value="VARCHAR"/>	<input type="text" value="50"/>	<input type="text" value="Nenhum"/>	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="text" value="---"/>
<input type="text" value="email"/>	<input type="text" value="VARCHAR"/>	<input type="text" value="100"/>	<input type="text" value="Nenhum"/>	<input type="text"/>	<input type="text"/>	<input type="checkbox"/>	<input type="text" value="---"/>

Abaixo clique em Salvar

Então ele nos mostra a estrutura da tabela criada

 **Estrutura da tabela**

 **Visão de relação(ões)**

#	Nome	Tipo	Colação	Atributos	Nulo	Padrão	Comentários	Extra	Ação
<input type="checkbox"/>	1	id	int(11)		Não	Nenhum			 Alterar  Eliminar  Mais
<input type="checkbox"/>	2	nome	varchar(50)	utf8mb4_general_ci	Não	Nenhum			 Alterar  Eliminar  Mais
<input type="checkbox"/>	3	email	varchar(100)	utf8mb4_general_ci	Não	Nenhum			 Alterar  Eliminar  Mais


Vamos clicar em Alterar no id para tornar AUTO_INCREMENT clicando à direita em A_I e Salvar
Isso possibilita que o MySQL se encarregue de adicionar o ID automaticamente, de forma que ao fazer um INSERT podemos incluir apenas os campos nome e e-mail e ele completa o id.


Também podemos criar uma tabela pelo menu SQL, que exige os comandos SQL. Aliás, podemos fazer de tudo por este menu.


Criar tabela via SQL


Vamos criar a mesma tabela, clientes, pelo menu SQL


A primeira providência é selecionar o banco onde desejamos criar a tabela, então clicar em SQL acima.


 **Estrutura**

 **SQL**

 **Procurar**

 **Consulta**

 **Expor**

Rodar consulta(s) SQL no banco de dados novo_banco: 

```
1 create table clientesSQL(  
2     id int primary key AUTO_INCREMENT,  
3     nome varchar(50) not null,  
4     email varchar(100)  
5 );
```

Rolar a tela e clicar em Executar

Veja que o campo nome ficou com not null, pois não tem sentido permitir o cadastro de clientes sem o nome. Já o e-mail não é obrigatório.

Sempre que possível é importante que os campos recebam not null.

Inserir no novo Registro na tabela criada

Temos um botão Inserir para inserir graficamente um novo registro mas vamos criar via aba SQL:

```
insert into produtos (descricao, preco) values ('Abacate', 2.85);
```

Atualizar o registro criado

```
update produtos set preco = 3.65 where descricao = 'Abacate';
```

Excluir um registro

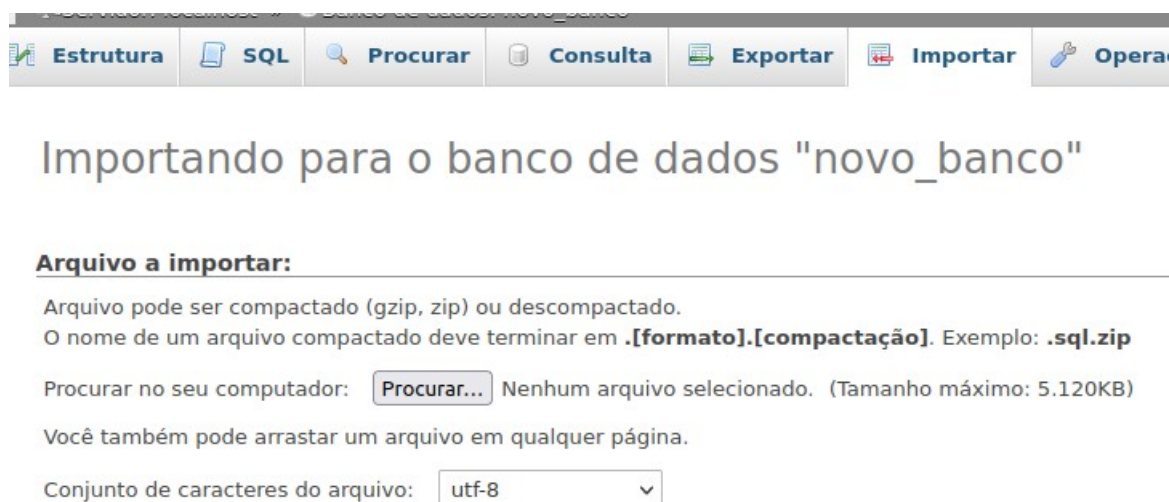
```
delete from produtos where descricao = 'Abacate';
```

Obs.: os comandos SQL podem ser utilizados em todos os gerenciadores do MySQL, inclusive na console.

Importar um Script SQL

É muito comum que ao testar um exemplo baixado ele venha com o script do banco no formato SQL. Neste caso criamos um banco e importamos o referido script.

- Selecionar o banco criado
- Clicar em Importar



Clique em Procurar e indique o script SQL.

Veja que ele acusa o tamanho máximo do script a ser importado. No meu caso 5.120 KB, mas este valor não é o padrão. O padrão é 2MB e isso pode ser alterado no php.ini.

Role a tela e clique em Executar

Lembrando que caso a tabela já exista com o mesmo nome no banco a importação falhará. No meu caso estou importando um script com uma tabela clientes também. Então removi a tabela que criei.

Acontece com frequência que o script tem comandos para criar um banco e usá-lo:
create database nome;
use nome;

Eu particularmente gosto de excluir estas linhas e importar para um banco que criei.

Após importar, clique num nome de tabela à esquerda que será aberta a janela Visualizar listando os registros e com muitas informações.

Remover uma tabela

- Selecione o banco à esquerda
- Na linha com o nome da tabela clique em Eliminar

Também podemos excluir uma tabela pelo menu SQL
drop table nome;

Remover um registro

- Selecione o banco
- Selecione a tabela
- Na linha do registro clique em Remover

Esta foi apenas uma introdução pelos recursos que mais uso, mas veja que existe muito mais.

3 - Gerenciamento de bancos

DML e DDL

O que diferencia as siglas é a letra do meio.

Como, por exemplo:

DML = "M" do meio vem de Manipulação.

DDL = "D" do meio vem de Definição.

DCL = "C" do meio vem de Controle.

DTL = "T" do meio vem de Transação.

DQL = "Q" do meio vem de "Consulta".

Pense assim, a linguagem SQL é uma só, porém ela é dividida em tipos de acordo com a funcionalidade dos comandos.

Os tipos da linguagem SQL são:

DDL - Data Definition Language - Linguagem de Definição de Dados.

São os comandos que interagem com os objetos do banco.

São comandos DDL : CREATE, ALTER e DROP

DML - Data Manipulation Language - Linguagem de Manipulação de Dados.

São os comandos que interagem com os dados dentro das tabelas.

São comandos DML : INSERT, DELETE e UPDATE

DQL - Data Query Language - Linguagem de Consulta de dados.

São os comandos de consulta.

São comandos DQL : SELECT (é o comando de consulta)

Aqui cabe um parenteses. Em alguns livros o SELECT fica na DML em outros tem esse grupo próprio.

DTL - Data Transaction Language - Linguagem de Transação de Dados.

São os comandos para controle de transação.

São comandos DTL : BEGIN TRANSACTION, COMMIT E ROLLBACK

DCL - Data Control Language - Linguagem de Controle de Dados.

São os comandos para controlar a parte de segurança do banco de dados.

São comandos DCL : GRANT, REVOKE E DENY.

Autor original

Luiz Fernando

Fonte - <https://www.devmedia.com.br/forum/o-que-e-dcl/564671>

<https://pt.stackoverflow.com/questions/262867/o-que-s%C3%A3o-as-siglas-ddl-dml-dql-dtl-e-dcl>

3.1 – DDL

Formado pelos comandos que alteram a estrutura do banco.

3.1.1 - Criando um banco de dados

Irei abordar usando comandos do SQL, pois criar gráficamente apenas clicando em botões não tem muito o que ensinar, é bem intuitivo.

Criar um banco usando a console SQL do gerenciador de banco de dados ou mesmo o seu terminal/console

Convencionou-se escrever o nome dos comandos SQL em maiúsculas, mas podemos escrever sem atentar para o case.

Considerando que você está em um gerenciador de bancos de dados como o phpMyAdmin, o adminer ou o MySQL Workbench, apenas acesse a aba SQL

Criar um banco de dados

Forma simplificada

```
CREATE DATABASE nome;
```

Sintaxe completa:

```
CREATE DATABASE [IF NOT EXISTS] nome  
[CHARACTER SET charset_name]  
[COLLATE collation_name];
```

Remover um banco existente

Forma simplificada

```
DROP DATABASE nome;
```

Sintaxe completa:

```
DROP DATABASE [IF EXISTS] nome;
```

Se estiver usando a console ou terminal do SGBD, no caso o MySQL ou seu fork MariaDb

Caso use no desktop sem senha

```
mysql -uroot
```

Se tiver senha

```
mysql -uroot -p
```

Agora poderá executar os mesmos comandos acima para criar ou remover o banco.

Criar dois bancos

E para cada um criar um usuário com todos os poderes mas somente sobre o seu banco

Importante. O usuário deve ter este formato:

'nome'@'host'

Laravel

```
mysql -uroot -p
create database acldb CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'aclus'@'localhost' IDENTIFIED BY 'senhaforte';
grant all privileges on acldb.* to 'aclus'@'localhost';
FLUSH PRIVILEGES;
```

Joomla

```
create database portaldb CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'portalus'@'localhost' IDENTIFIED BY 'senhaforte';
grant all privileges on portaldb.* to 'portalus'@'localhost';
FLUSH PRIVILEGES;
\q
```

Criar Tabela

```
CREATE TABLE nome(
    campo1 tipo PK,
    campo2 tipo not null
);
```

Adicionar um campo para a tabela produtos

```
alter table produtos add column teste int;
```

Remover o campo teste da tabela produtos

```
alter table produtos drop column teste;
```

Alterar o engine da tabela produtos para innodb

```
ALTER TABLE produtos ENGINE = InnoDB;
```

Resetar o valor inicial do id

```
ALTER TABLE produtos AUTO_INCREMENT = 13;
```

Alterar o charset da tabela produtos

```
ALTER TABLE produtos CHARACTER SET = utf8;
```

Mudar o tipo do campo

```
ALTER TABLE produtos MODIFY quantidade BIGINT NOT NULL;
```

Renomear um campo

```
ALTER TABLE produtos change quantidade quant int;
```

Remover chave estrangeira

```
ALTER TABLE produtos DROP FOREIGN KEY fk_nome;
```

Referência sobre SQL

<https://www.w3schools.com/sql/default.asp>

```
CREATE TABLE IF NOT EXISTS products (  
    productID INT UNSIGNED NOT NULL AUTO_INCREMENT,  
    productCode CHAR(3) NOT NULL DEFAULT "",  
    name VARCHAR(30) NOT NULL DEFAULT "",  
    quantity INT UNSIGNED NOT NULL DEFAULT 0,  
    price DECIMAL(7,2) NOT NULL DEFAULT 99999.99,  
    PRIMARY KEY (productID)  
);
```

```
ALTER TABLE products DROP FOREIGN KEY products_ibfk_1;  
SHOW CREATE TABLE products;
```

3.2 – DML

Comandos que manipulam os dados de tabelas

Gerenciamento dos bancos do MySQL via console

Gerenciar de forma semelhante também através de outros gerenciadores como o phpmyadmin e o adminer usando a aba SQL.

```
CREATE TABLE IF NOT EXISTS products (  
    productID INT UNSIGNED NOT NULL AUTO_INCREMENT,  
    productCode CHAR(3) NOT NULL DEFAULT "",  
    name VARCHAR(30) NOT NULL DEFAULT "",  
    quantity INT UNSIGNED NOT NULL DEFAULT 0,  
    price DECIMAL(7,2) NOT NULL DEFAULT 99999.99,  
    PRIMARY KEY (productID)  
);
```

```
INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);
```

-- Insert multiple rows in one command

-- Inserting NULL to the auto_increment column results in max_value + 1

```
INSERT INTO products VALUES  
    (NULL, 'PEN', 'Pen Blue', 8000, 1.25),  
    (NULL, 'PEN', 'Pen Black', 2000, 1.25);
```

-- Insert value to selected columns

-- Missing value for the auto_increment column also results in max_value + 1

```
INSERT INTO products (productCode, name, quantity, price) VALUES  
    ('PEC', 'Pencil 2B', 10000, 0.48),  
    ('PEC', 'Pencil 2H', 8000, 0.49);
```

- Missing columns get their default values

```
mysql> INSERT INTO products (productCode, name) VALUES ('PEC', 'Pencil HB');
```

-- 2nd column (productCode) is defined to be NOT NULL

```
mysql> INSERT INTO products values (NULL, NULL, NULL, NULL, NULL);
```

Usando select sem tabela

SELECT 1+1;

SELECT NOW();

SELECT 1+1, NOW();

SELECT name, quantity FROM products WHERE quantity <= 2000;

SELECT name, price FROM products WHERE name LIKE 'PENCIL%';

SELECT name, price FROM products WHERE name LIKE 'P__ %';

*SELECT * FROM products WHERE quantity >= 5000 AND price < 1.24 AND name LIKE 'Pen %';*

*SELECT * FROM products WHERE NOT (quantity >= 5000 AND name LIKE 'Pen %');*

*SELECT * FROM products WHERE name IN ('Pen Red', 'Pen Black');*

*SELECT * FROM products
WHERE (price BETWEEN 1.0 AND 2.0) AND (quantity BETWEEN 1000 AND 2000);*

*SELECT * FROM products WHERE productCode IS NULL;*

*SELECT * FROM products WHERE productCode = NULL;*

*SELECT ... FROM tableName
WHERE criteria
ORDER BY columnA ASC|DESC, columnB ASC|DESC, ...*

*SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC;*

*SELECT * FROM products WHERE name LIKE 'Pen %' ORDER BY price DESC, quantity;*

*SELECT * FROM products ORDER BY RAND();*

*SELECT * FROM products ORDER BY price LIMIT 2;*

*SELECT productID AS ID, productCode AS Code,
name AS Description, price AS `Unit Price` -- Define aliases to be used as display names
FROM products
ORDER BY ID; -- Use alias ID as reference*

SELECT CONCAT(productCode, ' - ', name) AS `Product Description`, price FROM products;

SELECT price FROM products;

SELECT DISTINCT price AS `Distinct Price` FROM products;

SELECT DISTINCT price, name FROM products;

*SELECT * FROM products ORDER BY productCode, productID;*

*SELECT * FROM products GROUP BY productCode;*

SELECT COUNT() AS `Count` FROM products;*

SELECT productCode, COUNT() FROM products GROUP BY productCode;*

```
SELECT MAX(price), MIN(price), AVG(price), STD(price), SUM(quantity)
FROM products;
-- Without GROUP BY - All rows
```

```
SELECT productCode, MAX(price) AS 'Highest Price', MIN(price) AS 'Lowest Price'
FROM products
GROUP BY productCode;
```

```
SELECT productCode, MAX(price), MIN(price),
    CAST(AVG(price) AS DECIMAL(7,2)) AS 'Average',
    CAST(STD(price) AS DECIMAL(7,2)) AS 'Std Dev',
    SUM(quantity)
FROM products
GROUP BY productCode;
-- Use CAST(... AS ...) function to format floating-point numbers
```

```
SELECT
    productCode,
    MAX(price),
    MIN(price),
    CAST(AVG(price) AS DECIMAL(7,2)) AS 'Average',
    SUM(quantity)
FROM products
GROUP BY productCode
WITH ROLLUP;    -- Apply aggregate functions to all groups
```

```
UPDATE products SET price = price * 1.1;
SELECT * FROM products;
```

```
UPDATE products SET quantity = quantity - 100 WHERE name = 'Pen Red';
SELECT * FROM products WHERE name = 'Pen Red';
```

```
UPDATE products SET quantity = quantity + 50, price = 1.23 WHERE name = 'Pen Red';
SELECT * FROM products WHERE name = 'Pen Red';
```

```
DELETE FROM products WHERE name LIKE 'Pencil%';
SELECT * FROM products;
```

```
DELETE FROM products;
SELECT * FROM products;
```

```
SELECT products.name, price, suppliers.name
FROM products
    JOIN suppliers ON products.supplierID = suppliers.supplierID
WHERE price < 0.6;
```

```
SELECT products.name, price, suppliers.name
FROM products, suppliers
WHERE products.supplierID = suppliers.supplierID
    AND price < 0.6;
```

```
SELECT products.name AS 'Product Name', price, suppliers.name AS 'Supplier Name'
FROM products
    JOIN suppliers ON products.supplierID = suppliers.supplierID
WHERE price < 0.6;
```

```
SELECT p.name AS 'Product Name', p.price, s.name AS 'Supplier Name'
FROM products AS p
    JOIN suppliers AS s ON p.supplierID = s.supplierID
WHERE p.price < 0.6;
```

```
ALTER TABLE products DROP FOREIGN KEY products_ibfk_1;
SHOW CREATE TABLE products;
```

Subconsulta

```
SELECT suppliers.name from suppliers
WHERE suppliers.supplierID
NOT IN (SELECT DISTINCT supplierID from products_suppliers);
```

```
SELECT * FROM patients
WHERE lastVisitDate BETWEEN '2012-09-15' AND CURDATE()
ORDER BY lastVisitDate;
```

```
SELECT * FROM patients
WHERE YEAR(dateOfBirth) = 2011
ORDER BY MONTH(dateOfBirth), DAY(dateOfBirth);
```

```
SELECT * FROM patients
WHERE MONTH(dateOfBirth) = MONTH(CURDATE())
AND DAY(dateOfBirth) = DAY(CURDATE());
```

```
SELECT name, dateOfBirth, TIMESTAMPDIFF(YEAR, dateOfBirth, CURDATE()) AS age
FROM patients
ORDER BY age, dateOfBirth;
```

```
SELECT name, lastVisitDate FROM patients
WHERE TIMESTAMPDIFF(DAY, lastVisitDate, CURDATE()) > 60;
```

```
SELECT name, lastVisitDate FROM patients
WHERE TO_DAYS(CURDATE()) - TO_DAYS(lastVisitDate) > 60;
```

```
SELECT * FROM patients
WHERE dateOfBirth > DATE_SUB(CURDATE(), INTERVAL 18 YEAR);
```

```
UPDATE patients
SET nextVisitDate = DATE_ADD(CURDATE(), INTERVAL 6 MONTH)
WHERE name = 'Ali';
```

```
select now(), curdate(), curtime();
```

```
SELECT DAYNAME(NOW()), MONTHNAME(NOW()), DAYOFWEEK(NOW()), DAYOFYEAR(NOW());
```

```
SELECT DATE_ADD('2012-01-31', INTERVAL 5 DAY);
```

```
SELECT DATE_SUB('2012-01-31', INTERVAL 2 MONTH);
```

```
SELECT DATEDIFF('2012-02-01', '2012-01-28');
```

```
SELECT TIMESTAMPDIFF(DAY, '2012-02-01', '2012-01-28');
```

```
SELECT TO_DAYS('2012-01-31');
```

```
SELECT FROM_DAYS(734899);
```

```
SELECT DATE_FORMAT('2012-01-01', '%W %D %M %Y');
Sunday 1st January 2012
-- %W: Weekday name
-- %D: Day with suffix
-- %M: Month name
-- %Y: 4-digit year
-- The format specifiers are case-sensitive
```

```
SELECT DATE_FORMAT('2011-12-31 23:59:30', '%W %D %M %Y %r');
```

```
CREATE TABLE IF NOT EXISTS `datetime_arena` (
```

```

`description` VARCHAR(50) DEFAULT NULL,
`cDateTime` DATETIME DEFAULT '1000-01-01 00:00:00',
`cDate` DATE DEFAULT '1000-01-01 ',
`cTime` TIME DEFAULT '00:00:00',
`cYear` YEAR DEFAULT '0000',
`cYear2` YEAR(2) DEFAULT '0000',
`cTimeStamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

```

```
DESCRIBE `datetime_arena`;
```

```

INSERT INTO `datetime_arena`
(`description`, `cDateTime`, `cDate`, `cTime`, `cYear`, `cYear2`)
VALUES
('Manual Entry', '2001-01-01 23:59:59', '2002-02-02', '12:30:30', '2004', '05');

```

```
mysql> SELECT * FROM `datetime_arena` WHERE description='Manual Entry';
```

```
UPDATE `datetime_arena` SET `cYear2`='99' WHERE description='Manual Entry';
```

```
mysql> SELECT * FROM `datetime_arena` WHERE description='Manual Entry';
```

```

INSERT INTO `datetime_arena`
(`description`, `cDateTime`, `cDate`, `cTime`, `cYear`, `cYear2`)
VALUES
('Built-in Functions', now(), curdate(), curtime(), now(), now());

```

```
mysql> SELECT * FROM `datetime_arena` WHERE description='Built-in Functions';
```

```

INSERT INTO `datetime_arena`
(`description`, `cDateTime`, `cDate`, `cTime`, `cYear`, `cYear2`)
VALUES
('Error Input', '2001-13-31 23:59:59', '2002-13-31', '12:61:61', '99999', '999');

```

```
mysql> SELECT * FROM `datetime_arena` WHERE description='Error Input';
```

```
SELECT `cDate`, `cDate` + INTERVAL 30 DAY, `cDate` + INTERVAL 1 MONTH FROM `datetime_arena`;
```

```

CREATE TABLE torneos (
nome varchar(30),
vitorias real,
melhor real,
tamanho real
);

```

```

INSERT INTO torneos (nome, vitorias, melhor, tamanho)
VALUES ('Dolly', '7', '245', '8.5'),
('Etta', '4', '283', '9'),
('Irma', '9', '266', '7'),
('Barbara', '2', '197', '7.5'),
('Gladys', '13', '273', '8');

```

```

CREATE TABLE refeicoes (
nome varchar(30),
data_nascimento date,
entrada varchar(30),
acompanhamento varchar(30),
sobremesa varchar(30)
);

```

```

INSERT INTO refeicoes (nome, data_nascimento, entrada, acompanhamento, sobremesa)
VALUES ('Dolly', '1946-01-19', 'steak', 'salad', 'cake'),

```



```
('Etta', '1938-01-25', 'chicken', 'fries', 'ice cream'),
('Irma', '1941-02-18', 'tofu', 'fries', 'cake'),
('Barbara', '1948-12-25', 'tofu', 'salad', 'ice cream'),
('Gladys', '1944-05-28', 'steak', 'fries', 'ice cream');
```

```
SELECT nome, vitorias FROM torneos
WHERE vitorias > (
SELECT vitorias FROM torneos WHERE nome = 'Barbara'
);
```

```
SELECT nome, tamanho FROM torneos AS t
WHERE vitorias > (
SELECT AVG(vitorias) FROM torneos WHERE tamanho = t.tamanho
);
```

```
SELECT nome, entrada, acompanhamento, sobremesa
FROM refeicoes
WHERE nome = (SELECT nome FROM torneos
WHERE vitorias = (SELECT MAX(vitorias) FROM torneos));
```

```
SELECT
`name`,
`age`
FROM
`people`
WHERE
`age` > 22
LIMIT
1
```

Listar tamanho de todos os bd em MBs

```
SELECT table_schema "DB Name",
Round(Sum(data_length + index_length) / 1024 / 1024, 1) "DB Size in MB"
FROM information_schema.tables
GROUP BY table_schema;
```

O banco information_schema traz as informações sobre os metadados.

Checar se registro não existe antes de inserir

```
INSERT INTO Table1 (column1, column2)
SELECT ?, ? FROM dual
WHERE NOT EXISTS (SELECT * FROM Table1 WHERE column1 = ? AND column2 = ?)
```

Referência

<https://www.ntu.edu.sg/home/ehchua/programming/sql/SQLMisc.html>
https://www.ntu.edu.sg/home/ehchua/programming/sql/MySQL_Beginner.html

3.2.1 – Like

```
SELECT
employeeNumber,
lastName,
firstName
FROM
employees
WHERE
```

```
firstName LIKE 'a%';
```

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastName LIKE '%on';
```

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastname LIKE '%on%';
```

Para encontrar employers cujos primeiros nomes começam com T , terminam com m, e contêm qualquer caractere único entre eles, por exemplo, Tom , Tim, utiliza-se o wildcard (_) de sublinhado para construir o padrão como se segue:

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    firstname LIKE 'T_m';
```

```
SELECT
    employeeNumber,
    lastName,
    firstName
FROM
    employees
WHERE
    lastName NOT LIKE 'B%';
```

Referências

<https://www.mysqltutorial.org/mysql-like/>

3.2.2 – LIMIT

MySQL LIMIT

Summary: in this tutorial, you will learn how to use MySQL LIMIT clause to constrain the number of rows returned by a query.

Introduction to MySQL LIMIT clause

The LIMIT clause is used in the SELECT statement to constrain the number of rows to return. The LIMIT clause accepts one or two arguments. The values of both arguments must be zero or positive integers.

The following illustrates the LIMIT clause syntax with two arguments:

```
SELECT
    select_list
FROM
    table_name
LIMIT [offset,] row_count;
Code language: SQL (Structured Query Language) (sql)
```

In this syntax:

The offset specifies the offset of the first row to return. The offset of the first row is 0, not 1.
The row_count specifies the maximum number of rows to return.

The following picture illustrates the LIMIT clause:

When you use the LIMIT clause with one argument, MySQL will use this argument to determine the maximum number of rows to return from the first row of the result set.

Therefore, these two clauses are equivalent:

LIMIT row_count;

And

LIMIT 0 , row_count;

In addition to the above syntax, MySQL provides the following alternative LIMIT clause syntax:

LIMIT row_count OFFSET offset

The LIMIT and ORDER BY clauses

By default, the SELECT statement returns rows in an unspecified order. When you add the LIMIT clause to the SELECT statement, the returned rows are unpredictable.

Therefore, to ensure the LIMIT clause returns an expected output, you should always use it with an ORDER BY clause like this:

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    sort_expression
LIMIT offset, row_count;
```

The following picture illustrates the evaluation order of the LIMIT clause in the SELECT statement:

LIMIT

MySQL LIMIT clause examples

We'll use the customers table from the sample database for demonstration.

1) Using MySQL LIMIT to get the highest or lowest rows

This statement uses the LIMIT clause to get the top five customers who have the highest credit:

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY creditLimit DESC
LIMIT 5;
```

Try It Out

mysql limit get highest values

In this example:

First, the ORDER BY clause sorts the customers by credits in high to low.

Then, the LIMIT clause returns the first 5 rows.

Similarly, this example uses the LIMIT clause to find five customers who have the lowest credits:

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY creditLimit
LIMIT 5;
```

Try It Out

mysql limit get lowest values

In this example:

First, the ORDER BY clause sorts the customers by credits in low to high.

Then, the LIMIT clause returns the first 5 rows.

Because there are more than 5 customers that have credits zero, the result of the query above may lead to an inconsistent result.

To fix this issue, you need to add more columns to the ORDER BY clause to constrain the row in unique order:

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY
    creditLimit,
    customerNumber
LIMIT 5;
```

Try It Out

2) Using MySQL LIMIT clause for pagination

When you display data on the screen, you often want to divide rows into pages, where each page contains a limited number of rows like 10 or 20.

To calculate the number of pages, you take the total rows divided by the number of rows per page. For fetching rows of a specific page, you can use the LIMIT clause.

This query uses the COUNT(*) aggregate function to get the total rows from the customers table:

```
SELECT
  COUNT(*)
FROM
  customers;
```

```
+-----+
| COUNT(*) |
+-----+
|    122   |
+-----+
```

1 row in set (0.00 sec)

Code language: JavaScript (javascript)

Suppose that each page has 10 rows; to display 122 customers, you have 13 pages. The last 13th page contains two rows only.

This query uses the LIMIT clause to get rows of page 1 which contains the first 10 customers sorted by the customer name:

```
SELECT
  customerNumber,
  customerName
FROM
  customers
ORDER BY customerName
LIMIT 10;
```

Try It Out

MySQL LIMIT for pagination example

This query uses the LIMIT clause to get the rows of the second page that include rows 11 – 20:

```
SELECT
  customerNumber,
  customerName
FROM
  customers
ORDER BY customerName
LIMIT 10, 10;
```

Try It Out

In this example, the clause LIMIT 10, 10 returns 10 rows for the row 11 – 20.

3) Using MySQL LIMIT to get the nth highest or lowest value

To get the nth highest or lowest value, you use the following LIMIT clause:

```
SELECT select_list
FROM table_name
ORDER BY sort_expression
LIMIT n-1, 1;
```

The clause LIMIT n-1, 1 returns 1 row starting at the row n.

For example, the following finds the customer who has the second-highest credit:

```
SELECT
    customerName,
    creditLimit
FROM
    customers
ORDER BY
    creditLimit DESC
LIMIT 1,1;
```

Try It Out

MySQL LIMIT find nth highest row example

Let's double-check the result. This query returns all customers sorted by credits from high to low:

```
SELECT
    customerName,
    creditLimit
FROM
    customers
ORDER BY
    creditLimit DESC;
```

Try It Out

As you can see clearly from the output, the result was correct as expected.

Note that this technique works when there are no two customers who have the same credit limits. To get a more accurate result, you should use the DENSE_RANK() window function.

MySQL LIMIT & DISTINCT clauses

If you use the LIMIT clause with the DISTINCT clause, MySQL immediately stops searching when it finds the number of unique rows specified in the LIMIT clause.

The example uses the LIMIT clause with the DISTINCT clause to return the first five unique states in the customers table:

```
SELECT DISTINCT
    state
FROM
    customers
WHERE
    state IS NOT NULL
LIMIT 5;
```

Try It Out

MySQL DISTINCT with LIMIT clause

Summary

Use the MySQL LIMIT clause to constrain the number of rows returned by the SELECT statement.

<https://www.mysqltutorial.org/mysql-limit.aspx>

4 - Tipos de dados

Tipos de Dados no MySQL

Abaixo temos uma lista dos tipos de dados mais comuns e suas descrições em MySQL:

DECIMAL(M,D) – Ponto decimal com M dígitos no total (precisão) e D casas decimais (escala); o padrão é 10,0; M vai até 65 e D até 30.

FLOAT(M,D) – Ponto flutuante com precisão M e escala D; o padrão é 10,2; D vai até 24.

CHAR(M) – String que ocupa tamanho fixo entre 0 e 255 caracteres

BOOL / BOOLEAN – Valores binários 0 / 1; Na verdade, é um alias para o tipo TINYINT(1)

VARCHAR(M) – String de tamanho variável, até 65535 caracteres.

BLOB / MEDIUMBLOB/ TINYBLOB – Campo com tamanho máximo de 65535 caracteres binários; ‘Binary Large Objects’, são usados para armazenar grandes quantidades de dados, como imagens.

MEDIUMTEXT – Permite armazenar até 16.777.215 caracteres.

LONGTEXT – Permite armazenar até 4.294.967.295 caracteres.

DATE – Uma data de 01/01/1000 a 31/12/9999, no formato YYYY-MM-DD

DATETIME – Uma combinação de data e hora de 01/01/1000 00:00:00 a 31/12/9999 23:59:59, no formato YYYY-MM-DD HH:MM:SS

TIME – Hora apenas, no formato HH:MM:SS

YEAR(M) – Ano nos formatos de 2 ou 4 dígitos; Se forem 2 (YEAR(2)), ano vai de 1970 a 2069; para 4 (YEAR(4)), vai de 1901 a 2155. O padrão é 4.

<http://www.bosontreinamentos.com.br/mysql/mysql-tipos-de-dados-comuns-09/>
<https://elias.praciano.com/2014/01/mysql-tipos-de-dados/>

Alguns tipos especiais

unidade enum('KG', 'M', 'M3', 'G') DEFAULT 'KG' NOT NULL;

O enum traz uma lista de possíveis valores e é interessante usar um tipo de campo select para listar os mesmos.

DECIMAL(M,D)

PRECISION DECIMAL (12,2);

DATETIME - Entre 1000-01-01 00:00:00 e 9999-12-31 23:59:59

data datetime

data '1973-12-30 15:30:00';

YEAR - pode armazenar o ano com 2 ou 4 dígitos

Entre 1970 e 2069

YEAR(2) ou YEAR

Valor default para time

time timestamp DEFAULT NOW());

5 - Funções nativas do MySQL

```
CREATE TABLE `precos` (  
  `col1` INT( 3 ) NOT NULL ,  
  `col2` INT( 3 ) NOT NULL  
) ENGINE = MYISAM;
```

```
INSERT INTO `precos` (`col1`, `col2`)  
VALUES ('2', '4'), ('4', '6'), ('5', '9');
```

Adição

```
SELECT(col1 + col2) as Resultado FROM precos
```

```
SELECT (SUM(col1) + SUM(col2)) as Resultado FROM precos
```

Subtração

```
SELECT(col1 - col2) as Resultado FROM precos
```

Multiplicação

O operador utilizado para multiplicação é o (* asterisco).

```
SELECT(col1 * col2) as Resultado FROM precos
```

Divisão

O operador utilizado para divisão é a (/ barra).

```
SELECT(col1 / col2) as Resultado FROM precos
```

Multiplicação

O operador utilizado para multiplicação é o (* asterisco).

```
SELECT(col1 * col2) as Resultado FROM precos
```

Módulo ou MOD

Esta função retorna o restante de dois numeros. Exemplo: 21 / 5 (vinte e um dividido por 5), o resto seria 1

```
SELECT MOD(21, 5) AS Resultado
```

<https://blog.fabianobento.com.br/2010/08/funcoes-basicas-mysql-iniciantes/>

MySQL Functions

MySQL has many built-in functions.

This reference contains string, numeric, date, and some advanced functions in MySQL.

MySQL String Functions

Function	Description
ASCII	Returns the ASCII value for the specific character
CHAR_LENGTH	Returns the length of a string (in characters)
CHARACTER_LENGTH	Returns the length of a string (in characters)
CONCAT	Adds two or more expressions together
CONCAT_WS	Adds two or more expressions together with a separator
FIELD	Returns the index position of a value in a list of values
FIND_IN_SET	Returns the position of a string within a list of strings
FORMAT	Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places
INSERT	Inserts a string within a string at the specified position and for a certain number of characters
INSTR	Returns the position of the first occurrence of a string in another string
LCASE	Converts a string to lower-case
LEFT	Extracts a number of characters from a string (starting from left)
LENGTH	Returns the length of a string (in bytes)
LOCATE	Returns the position of the first occurrence of a substring in a string
LOWER	Converts a string to lower-case
LPAD	Left-pads a string with another string, to a certain length
LTRIM	Removes leading spaces from a string
MID	Extracts a substring from a string (starting at any position)
POSITION	Returns the position of the first occurrence of a substring in a string
REPEAT	Repeats a string as many times as specified
REPLACE	Replaces all occurrences of a substring within a string, with a new substring
REVERSE	Reverses a string and returns the result
RIGHT	Extracts a number of characters from a string (starting from right)
RPAD	Right-pads a string with another string, to a certain length
RTRIM	Removes trailing spaces from a string
SPACE	Returns a string of the specified number of space characters
STRCMP	Compares two strings
SUBSTR	Extracts a substring from a string (starting at any position)
SUBSTRING	Extracts a substring from a string (starting at any position)
SUBSTRING_INDEX	Returns a substring of a string before a specified number of delimiter occurs
TRIM	Removes leading and trailing spaces from a string
UCASE	Converts a string to upper-case
UPPER	Converts a string to upper-case

MySQL Numeric Functions

Function	Description
ABS	Returns the absolute value of a number
ACOS	Returns the arc cosine of a number
ASIN	Returns the arc sine of a number
ATAN	Returns the arc tangent of one or two numbers

ATAN2	Returns the arc tangent of two numbers
AVG	Returns the average value of an expression
CEIL	Returns the smallest integer value that is >= to a number
CEILING	Returns the smallest integer value that is >= to a number
COS	Returns the cosine of a number
COT	Returns the cotangent of a number
COUNT	Returns the number of records returned by a select query
DEGREES	Converts a value in radians to degrees
DIV	Used for integer division
EXP	Returns e raised to the power of a specified number
FLOOR	Returns the largest integer value that is <= to a number
GREATEST	Returns the greatest value of the list of arguments
LEAST	Returns the smallest value of the list of arguments
LN	Returns the natural logarithm of a number
LOG	Returns the natural logarithm of a number, or the logarithm of a number to a specified base
LOG10	Returns the natural logarithm of a number to base 10
LOG2	Returns the natural logarithm of a number to base 2
MAX	Returns the maximum value in a set of values
MIN	Returns the minimum value in a set of values
MOD	Returns the remainder of a number divided by another number
PI	Returns the value of PI
POW	Returns the value of a number raised to the power of another number
POWER	Returns the value of a number raised to the power of another number
RADIANS	Converts a degree value into radians
RAND	Returns a random number
ROUND	Rounds a number to a specified number of decimal places
SIGN	Returns the sign of a number
SIN	Returns the sine of a number
SQRT	Returns the square root of a number
SUM	Calculates the sum of a set of values
TAN	Returns the tangent of a number
TRUNCATE	Truncates a number to the specified number of decimal places

MySQL Date Functions

Function	Description
ADDDATE	Adds a time/date interval to a date and then returns the date
ADDTIME	Adds a time interval to a time/datetime and then returns the time/datetime
CURDATE	Returns the current date
CURRENT_DATE	Returns the current date
CURRENT_TIME	Returns the current time
CURRENT_TIMESTAMP	Returns the current date and time
CURTIME	Returns the current time
DATE	Extracts the date part from a datetime expression
DATEDIFF	Returns the number of days between two date values
DATE_ADD	Adds a time/date interval to a date and then returns the date
DATE_FORMAT	Formats a date
DATE_SUB	Subtracts a time/date interval from a date and then returns the date
DAY	Returns the day of the month for a given date
DAYNAME	Returns the weekday name for a given date
DAYOFMONTH	Returns the day of the month for a given date
DAYOFWEEK	Returns the weekday index for a given date
DAYOFYEAR	Returns the day of the year for a given date

EXTRACT	Extracts a part from a given date
FROM_DAYS	Returns a date from a numeric datevalue
HOUR	Returns the hour part for a given date
LAST_DAY	Extracts the last day of the month for a given date
LOCALTIME	Returns the current date and time
LOCALTIMESTAMP	Returns the current date and time
MAKEDATE	Creates and returns a date based on a year and a number of days value
MAKETIME	Creates and returns a time based on an hour, minute, and second value
MICROSECOND	Returns the microsecond part of a time/datetime
MINUTE	Returns the minute part of a time/datetime
MONTH	Returns the month part for a given date
MONTHNAME	Returns the name of the month for a given date
NOW	Returns the current date and time
PERIOD_ADD	Adds a specified number of months to a period
PERIOD_DIFF	Returns the difference between two periods
QUARTER	Returns the quarter of the year for a given date value
SECOND	Returns the seconds part of a time/datetime
SEC_TO_TIME	Returns a time value based on the specified seconds
STR_TO_DATE	Returns a date based on a string and a format
SUBDATE	Subtracts a time/date interval from a date and then returns the date
SUBTIME	Subtracts a time interval from a datetime and then returns the time/datetime
SYSDATE	Returns the current date and time
TIME	Extracts the time part from a given time/datetime
TIME_FORMAT	Formats a time by a specified format
TIME_TO_SEC	Converts a time value into seconds
TIMEDIFF	Returns the difference between two time/datetime expressions
TIMESTAMP	Returns a datetime value based on a date or datetime value
TO_DAYS	Returns the number of days between a date and date "0000-00-00"
WEEK	Returns the week number for a given date
WEEKDAY	Returns the weekday number for a given date
WEEKOFYEAR	Returns the week number for a given date
YEAR	Returns the year part for a given date
YEARWEEK	Returns the year and week number for a given date

MySQL Advanced Functions

Function	Description
BIN	Returns a binary representation of a number
BINARY	Converts a value to a binary string
CASE	Goes through conditions and return a value when the first condition is met
CAST	Converts a value (of any type) into a specified datatype
COALESCE	Returns the first non-null value in a list
CONNECTION_ID	Returns the unique connection ID for the current connection
CONV	Converts a number from one numeric base system to another
CONVERT	Converts a value into the specified datatype or character set
CURRENT_USER	Returns the user name and host name for the MySQL account that the server used to authenticate the current client
DATABASE	Returns the name of the current database
IF	Returns a value if a condition is TRUE, or another value if a condition is FALSE
IFNULL	Return a specified value if the expression is NULL, otherwise return the expression
ISNULL	Returns 1 or 0 depending on whether an expression is NULL
LAST_INSERT_ID	Returns the AUTO_INCREMENT id of the last row that has been inserted or updated in a table

NULLIF Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned

SESSION_USER Returns the current MySQL user name and host name

SYSTEM_USER Returns the current MySQL user name and host name

USER Returns the current MySQL user name and host name

VERSION Returns the current version of the MySQL database

https://www.w3schools.com/mysql/mysql_ref_functions.asp

6 - Relacionamentos entre tabelas (introdução)

Tabelas como pedidos e clientes, pedidos e produtos, grupos e usuários e outras similares devem ser relacionadas com a constraint foreign key, para garantir a integridade dos dados.

Exemplos:

Sem relacionamentos uma tabela de pedidos permite cadastrar um pedido para um cliente que não existe, mas quando bem relacionadas somente cadastrar pedido para cliente já cadastrado. Assim como não permitirá remover um cliente que tenha pedidos em seu nome.

Foreign key

pedidos e produtos com relacionamento um para vários

Dica: Ao importar scripts de tabelas relacionadas sempre importe primeiro a tabela primária, aquela que não tem chave estrangeira, neste caso, produtos

```
CREATE TABLE produtos (  
    id int primary key auto_increment,  
    nome varchar(50) not null,  
    quantidade int  
);
```

```
CREATE TABLE pedidos (  
    id int primary key auto_increment,  
    data date not null,  
    quantidade int,  
    produto_id int not null,  
    CONSTRAINT `fk-produto` FOREIGN KEY (`produto_id`) REFERENCES `produtos` (`id`) ON DELETE CASCADE  
ON UPDATE CASCADE  
);
```

7 - Usuários e Privilégios

Por conta da segurança e da eficiência do aplicativo, cada aplicativo deve ter um usuário e este usuário deve ter apenas os privilégios necessários para usar o aplicativo e nunca todos os privilégios.

EVITAR A CRIAÇÃO DE USUÁRIO com tantos poderes como os abaixo

```
mysql -u root -p
GRANT ALL PRIVILEGES ON *.* TO admin@"%"
IDENTIFIED BY 'senha' WITH GRANT OPTION;
```

Liberando apenas para 192.168.0.102 (web)

```
mysql -u root -p
create database portal_db;
GRANT ALL PRIVILEGES ON portal_db.* TO portal_us@192.168.0.102 IDENTIFIED BY 'senha' WITH GRANT
OPTION;
\q
```

/etc/init.d/mysql restart

Privilégios:

. - Privilégio global. Todos os bancos (*) e todas as tabelas (.*)

db.* - Todas as tabelas do banco db

db.tabela1 - Somente a tabela tabela1 do banco db

Acesso com o uso do coringa (%):

Exemplos:

... TO remoto@"%.mysqlbrasil.com.br"

... TO remoto@"200.236.13.%"

... TO " @"%.mysqlbrasil.com.br"

```
mysql -u root -p
```

```
CREATE DATABASE familia CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE USER 'us_familia'@'localhost' IDENTIFIED BY 'senhaforte';
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE TEMPORARY TABLES,
LOCK TABLES ON familia.* TO 'us_familia'@'localhost' IDENTIFIED BY 'yourpassword';
FLUSH PRIVILEGES;
\q
```

Outros privilégios

ALL PRIVILEGES- como vimos anteriormente, isso daria a um usuário do MySQL todo o acesso a uma determinada base de dados (ou se nenhuma base de dados for selecionada, todo o sistema)

CREATE- permite criar novas tabelas ou bases de dados

DROP- permite deletar tabelas ou bases de dados

DELETE- permite deletar linhas das tabelas

INSERT- permite inserir linhas nas tabelas

SELECT- permite utilizar o comando Select para ler bases de dados

UPDATE- permite atualizar linhas das tabelas

GRANT OPTION- permite conceder ou revogar privilégios de outros usuários


```
mysql -u root -p
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER, CREATE TEMPORARY
TABLES, LOCK TABLES ON joomla.* TO 'yourusername'@'localhost' IDENTIFIED BY 'yourpassword';
```

Criação de um usuário restrito

mysql_secure_installation

A maior restrição é conceder somente o privilégio de SELECT

```
mysql -uroot -p
```

```
CREATE USER 'laravel_us'@'localhost' IDENTIFIED BY 'senhaforte';
CREATE DATABASE `laravel_db` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
GRANT ALL PRIVILEGES ON `laravel_db`.* TO `laravel_us`@'localhost' IDENTIFIED BY 'senhaforte';
FLUSH PRIVILEGES;
```

```
CREATE USER 'admin_us'@'localhost' IDENTIFIED BY 'senhaforte';
CREATE DATABASE `admin_db` CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
GRANT ALL PRIVILEGES ON `admin_db`.* TO `admin_us`@'localhost' IDENTIFIED BY 'senhaforte';
FLUSH PRIVILEGES;
```

Após instalar use somente SELECT ou mais alguns privilégios (apenas uma sugestão e para alguns casos, aplicativos demo por exemplo)

```
REVOKE ALL ON `laravel_db`.* TO `laravel_us`@'localhost' IDENTIFIED BY 'senhaforte';
GRANT SELECT ON `laravel_db`.* TO `laravel_us`@'localhost' IDENTIFIED BY 'senhaforte';
FLUSH PRIVILEGES;
```

```
GRANT INSERT, SELECT, DELETE, UPDATE ON `admin_db`.* TO `admin_us`@'localhost' IDENTIFIED BY
'zmxn1029A@';
```

```
REVOKE ALL ON `admin_db`.* TO `admin_us`@'localhost' IDENTIFIED BY 'senhaforte';
GRANT SELECT ON `admin_db`.* TO `admin_us`@'localhost' IDENTIFIED BY 'senhaforte';
FLUSH PRIVILEGES;
```

Os privilégios abaixo são suficientes para o CMS Joomla

```
GRANT CREATE, INSERT, UPDATE, SELECT, DELETE, DROP, INDEX, ALTER, LOCK TABLES, CREATE
TEMPORARY TABLES, GRANT ON `joomla_db`.* TO `joomla_us`@'localhost' IDENTIFIED BY 'senhaforte';
```

8 – Views (introdução)

Uma view é uma consulta que podemos salvar para chamar depois.

```
CREATE VIEW supplier_view
AS
SELECT suppliers.name as `Supplier Name`, products.name as `Product Name`
FROM products
JOIN suppliers ON products.productID = products_suppliers.productID
JOIN products_suppliers ON suppliers.supplierID = products_suppliers.supplierID;
```

```
CREATE VIEW patient_view
AS
SELECT
    patientID AS ID,
    name AS Name,
    dateOfBirth AS DOB,
    TIMESTAMPDIFF(YEAR, dateOfBirth, NOW()) AS Age
FROM patients
ORDER BY Age, DOB;
```

```
SELECT * FROM patient_view WHERE Name LIKE 'A%'
```

```
SELECT * FROM patient_view WHERE age >= 18;
```

9 – Dicas

Valores default para campos datetime

```
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
```

Muito utilizados em frameworks

Utilização de EXISTS

Caso precise retornar em uma consulta registros de uma tabela que satisfaçam uma determinada condição segundo referências de uma segunda tabela, ao invés de utilizar uma subconsulta na cláusula WHERE para um operador IN prefira a utilização de EXISTS:

```
SELECT *  
FROM MINHA_TABELA M  
WHERE EXISTS (SELECT *  
              FROM TABELA_LOG L  
              WHERE L.ID_MINHA_TABELA = M.ID  
              AND TO_CHAR(L.DATA, 'YYYY') = '2017');
```

Na maior parte dos cenários, esta forma tem um desempenho muito superior, em diversos bancos dados, do que a utilização da cláusula IN com subconsultas:

```
SELECT *  
FROM MINHA_TABELA M  
WHERE M.ID IN (SELECT L.ID_MINHA_TABELA  
              FROM TABELA_LOG L  
              WHERE TO_CHAR(L.DATA, 'YYYY') = '2017');
```

Não utilize HAVING para filtrar dados

Caso necessite filtrar dados em um agrupamento de informações, prefira sempre realizar esta operação na cláusula WHERE ao invés do HAVING, por questões de performance, a não ser que seja necessário realizar algum filtro utilizando realmente as operações de agregação:

--NÃO RECOMENDÁVEL

```
SELECT NOME, TIPO  
FROM MINHA_TABELA A  
GROUP BY NOME, TIPO  
HAVING TIPO = 2
```

--RECOMENDÁVEL

```
SELECT NOME, TIPO  
FROM MINHA_TABELA A  
WHERE TIPO=2  
GROUP BY NOME, TIPO
```

Utilize procedures e views

Quando não utilizamos procedures e views, toda vez que executamos uma instrução SQL é necessário que o SGBD analise se a sintaxe do comando esta correta, se os objetos referenciados realmente existentes, dentre outras análises igualmente necessárias.

Quando o código a ser executado encontra-se em uma procedure ou view, o banco de dados não precisa fazer estas verificações e validações, pois as mesmas já foram feitas ao ser criar as procedures e views. Portanto, com o banco de dados poupando este trabalho, logicamente a performance das execuções de SQL enviados pela aplicação aumenta consideravelmente em sistemas críticos.

Tipos são extremamente importantes

Se preocupe sempre com os tipos das colunas das tabelas do sistema que você está criando para verificar se os mesmos correspondem fielmente ao tipo de informação que será armazenada. Por exemplo, se a coluna irá armazenar uma data, crie um campo do tipo DATE. Se vai armazenar um número inteiro, crie uma coluna do tipo INTEGER e por aí vai. Isto parece óbvio, mas é muito comum encontrarmos este tipo de situação.

Esta boa prática vai dar segurança por não permitir que seja inserida informação com tipo não compatível ou inconsistente, e vai melhorar a performance de consultas futuramente por não haver necessidade de se fazer conversões de tipo.

Traga no SELECT somente as colunas necessárias

Esta é a dica mais clichê de todas: evite utilizar SELECT * FROM. É muito comum fazermos consultas com JOINS em diversas tabelas. Especificar no SELECT somente as colunas que vai realmente utilizar melhorará é uma boa prática quase obrigatória para nós desenvolvedoras. Um outro benefício que também considero importante é facilitar a leitura do SQL em manutenções também.

Só utilize ORDER BY e DISTINCT se for realmente necessário

Muitas vezes, para determinadas funcionalidades do sistema, ordenação seja algo que não importe ou não haja necessidade, devendo ser evitado. Às vezes queremos, por exemplo, apenas listar conteúdo em tela e resolvemos por conta própria, sem haver especificações explícitas pra isto, fazer ordenação por data, registro mais recente e por aí vai, sendo que às vezes isto não agregará valor ao usuário final na aplicação. É estranho algum retorno de consulta sem ordenação? Pode até parecer, mas deve ser utilizado conscientemente por questões de performance, assim como o DISTINCT.

Utilização de índices em colunas muito acessadas

Caso seja identificado que é necessária a criação de algum índice que vise melhorar a performance das consultas à base de dados de uma aplicação, procure fazer as seguintes perguntas para determinar o critério de criação, exatamente nesta ordem:

Qual coluna é acessada ou requisitada com mais frequência, sendo chave-primária ou não?

Será que não é conveniente a modificação ou remodelagem da estrutura para fins de performance, considerando a criticidade desta minha consulta?

Índices em colunas muito presentes em WHERE, JOIN, ORDER BY e TOP

Uma boa dica para verificar se seria conveniente a criação de um índice em determinada coluna é verificar a frequência de utilização delas em cláusulas WHERE, JOIN, ORDER BY e TOP. Esta é sempre uma boa pista de índices que poderiam ser criados.

Mas, como dito no tópico anterior, a criação de índices sempre deve ser analisada e aplicada com muito cuidado.

Não deixe as chaves estrangeiras para depois

Esta dica é quase que tão óbvia e trivial quanto a não utilização de SELECT * FROM. Mas, é muito comum vermos sistemas criados utilizando tabelas sem os devidos relacionamentos no banco de dados. Então, nunca deixe pra depois a criação das devidas referências de chave-primária e estrangeira. Crie-as no exato momento da criação da própria tabela.

Dedicação de tempo à modelagem

Como dito anteriormente, muitas vezes o banco de dados é a “alma” do sistema. Vale a pena investir tempo no correto planejamento e modelagem da base de dados, refletindo na estrutura a ser criada de cada tabela, coluna, relacionamentos e muitos outros aspectos.

Investir nesta etapa vale muito a pena.

Nunca execute DELETE ou UPDATE sem um WHERE.

<https://www.linkedin.com/in/alexandremalavasi/>

10 – Metadados

Metadados são dados sobre os dados, ou seja, informações não sobre os dados, mas sobre as tabelas, campos, tipos de dados, etc.

Você que utiliza MySQL Server ou MariaDB e precisa obter informações como: se o banco existe, se a tabela existe, se o campo existe, se a view existe ou até mesmo se a procedure existe para fazer suas atualizações em tempo de execução, este vídeo é para você !

Verificando se a base de dados (schema) pdv existe:

```
show databases like 'pdv';
```

Verificando se a tabela produto existe no banco pdv:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'pdv'
AND TABLE_TYPE = 'BASE TABLE'
AND table_name = 'produto';
```

Verificando se o campo DescProd existe na tabela produto no banco PDV

```
select * from information_schema.COLUMNS
where TABLE_SCHEMA = 'pdv'
and TABLE_NAME = 'produto'
and COLUMN_NAME = 'DescProd'
```

Verificando se a view vw_venda existe na base pdv:

```
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'pdv'
AND TABLE_TYPE = ''
AND table_name = 'vw_venda';
```

Verificando se a procedure pr_ucAcesso existe na base pdv:

```
SHOW PROCEDURE STATUS
where db = 'pdv' and name = 'pr_ucAcesso'
```

<https://infocotidiano.com.br/category/dicas-mysql>

É possível obter o tipo de campo de uma **consulta** MySQL , da mesma maneira que você pode obtê-lo de uma tabela com o SHOW COLUMNS comando? Como em uma tabela derivada,

```
SELECT x -- presumedMetaFn(x) -- returns "int"
FROM (
    SELECT 1 AS x
    UNION SELECT 2
) AS t;
```

acessar metadados no mysql

Algumas vezes é necessario acessar os metadados para pegar o nome, char set, tipo ou tamanho de alguma objeto do banco, no mysql isso é bem simples. Esse tutorial será bem pratico para mais informações acesse: <http://dev.mysql.com/doc/refman/5.5/en/information-schema.html>

listar todas a tabelas daquela database:

```
SELECT * FROM information_schema.tables WHERE table_schema = 'nome_da_database'
```

exibe a estrutura de uma tabela.

```
SELECT * FROM information_schema.columns WHERE table_name = 'nome_da_tabela'
```

Todas as views

```
SELECT * FROM information_schema.views
```

Outra maneira:

todas as databases

```
show databases
```

<https://perdeuinfo.wordpress.com/2012/08/29/acessar-metadados-no-mysql/>

Download da Apostila MySQL

<https://github.com/ribafs/apostilas>

Seção de arquivos do grupo PHP Brasil do Facebook

<https://www.facebook.com/download/953643422221880/MySQLApostila.pdf>