**MySQL**

**Comando CREATE DATABASE**

Podemos criar um novo banco de dados com o comando a seguir:

**CREATE DATABASE [IF NOT EXISTS] nome\_BD;**

Onde nome\_BD é o nome que queremos dar ao banco de dados. O elemento **IF NOT EXISTS** é opcional. Ele previne o erro de tentar criar um banco de dados que já existe no servidor. Não é possível ter dois bancos de dados com o mesmo nome.

**Comando USE**

O comando USE instrui o SGBDR a utilizar o banco de dados especificado para rodar os comandos.

Sintaxe:

**USE nome\_banco\_de\_dados;**

**Comando DROP DATABASE**

**Excluir Banco de Dados**

Podemos excluir um banco de dados existente com o comando DROP DATABASE;

**DROP DATABASE [IF EXISTS] nome\_BD;**

**Comando CREATE TABLE**

Para criar tabelas em um banco de dados, usamos a declaração **CREATE TABLE**. Veja a sintaxe abaixo:

**CREATE TABLE [IF NOT EXISTS] nome\_tabela(**

**nome-coluna tipo\_dados constraints**

**nome-coluna tipo\_dados constraints**

**nome-coluna tipo\_dados constraints**

**......**

**);**

Note que precisamos especificar, além do nome da tabela, os nomes das colunas que a comporão e também seus respectivos tipos de dados, além das eventuais constraints.

### Tipos de Dados no MySQL

A tabela abaixo resume os tipos de dados mais comuns no MySQL, que podem ser usados na criação de tabelas, para estabelecer o tipo de cada coluna:

|  |  |
| --- | --- |
| **Tipo** | **Descrição** |
| INT | Inteiros entre -2,147,483,648 e 2,147,483,647 |
| TINYINT | Números inteiros de -128 a 127 |
| SMALLINT | Números inteiros de -32768 a 32767 |
| MEDIUMINT | Números inteiros de -8388608 a 8388607 |
| BIGINT | Números entre -9,223,372,036,854,775,808 e 9,223,372,036,854,775,807 |
| 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, com até 65535 M 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. |

## Constraints (Restrições)

* As Restrições são regras aplicadas nas colunas de uma tabela.
* São usadas para limitar os tipos de dados que são inseridos.
* Podem ser especificadas no momento de criação da tabela (CREATE) ou após a

tabela ter sido criada (ALTER)

As principais constraints são as seguintes:

* NOT NULL
* PRIMARY KEY
* UNIQUE
* FOREIGN KEY
* DEFAULT

**NOT NULL**

* A constraint NOT NULL impõe a uma coluna a NÃO aceitar valores NULL.
* Ou seja, a constraint NOT NULL obriga um campo a sempre possuir um valor.
* Deste modo, não é possível inserir um registro (ou atualizar) sem entrar com um valor neste campo.

**PRIMARY KEY**

* A restrição PRIMARY KEY (Chave Primária) identifica de forma única cada registro em uma tabela de banco de dados.
* As Chaves Primárias devem sempre conter valores únicos.
* Uma coluna de chave primária não pode conter valores NULL
* Cada tabela deve ter uma chave primária e apenas uma chave primária.

**UNIQUE**

* A restrição UNIQUE identifica de forma única cada registro em uma tabela de um banco de dados.
* As constraints UNIQUE e PRIMARY KEY garantem a unicidade em uma coluna ou conjunto de colunas.
* Uma constraint PRIMARY KEY automaticamente possui uma restrição UNIQUE definida, portanto não é necessário especificar essa constraint neste caso.
* É possível termos várias constraints UNIQUE em uma mesma tabela, mas apenas uma Chave Primária por tabela (lembrando que uma PK pode ser composta, ou seja, constituída por mais de uma coluna – mas ainda assim, será uma única chave primária).

**FOREIGN KEY**

* Uma FOREIGN KEY (Chave Estrangeira) em uma tabela é um campo que aponta para uma chave primária em outra tabela.
* Desta forma, é usada para criar os relacionamentos entre as tabelas no banco de dados.

**DEFAULT**

* A restrição DEFAULT é usada para inserir um valor padrão especificado em uma coluna.
* O valor padrão será adicionado a todos os novos registros caso nenhum outro valor seja especificado na hora de inserir dados.

**Auto Incremento em colunas (AUTO\_INCREMENT)**

* O auto incremento permite que um número único seja gerado quando um novo registro é inserido em uma tabela.
* No MYSQL trata-se da palavra chave AUTO\_INCREMENT, cujo valor inicial padrão é 1, e se incrementa de 1 em 1.
* Ao inserir valores na tabela, não é necessário especificar o valor para a coluna de auto-incremento.
* Só é permitido usar uma coluna de auto incremento por tabela, geralmente do tipo inteiro.
* Necessita também da constraint NOT NULL (configurado automaticamente)

**Comando ALTER TABLE**

É possível alterar a estrutura de uma tabela no MySQL após ter sido criada, acrescentando ou excluindo atributos (campos)

Usamos para isso o comando **ALTER TABLE**

### Excluir colunas: ALTER TABLE – DROP

Sintaxe:

**ALTER TABLE nome-tabela**

**DROP COLUMN nome-coluna;**

### Adicionar colunas: ALTER TABLE – ADD

Sintaxe:

**ALTER TABLE nome-tabela**

**ADD  nome-coluna tipo\_dados constraints;**

### Alterar colunas de uma tabela: ALTER TABLE-MODIFY COLUMN

Podemos alterar colunas, por exemplo mudando o tipo de dados:

Sintaxe:

**ALTER TABLE nome-tabela**

**MODIFY COLUMN nome-coluna novo\_tipo\_dados;**

#### Alterar constraints

Podemos também alterar uma coluna, adicionando constraints, como uma chave primária.

Sintaxe:

**ALTER TABLE nome-tabela**

**ADD PRIMARY KEY nome-coluna;**

### ****Excluir Constraints****

Pode-se excluir uma chave primária (ou constraint):

Sintaxe:

**ALTER TABLE nome-tabela**

**DROP PRIMARY KEY;**

Pode-se excluir uma chave estrangeira também:

**ALTER TABLE nome-tabela**

**DROP FOREIGN KEY;**

### DROP TABLE – Excluir tabela

Podemos excluir uma tabela com o comando DROP TABLE

Sintaxe:

**DROP TABLE *nome\_tabela;***

## Comando INSERT INTO

Uma das operações mais importantes em um banco de dados é a inserção de registros (dados). Fazemos isso com o uso do comando **INSERT INTO**.

Sintaxe:

**INSERT INTO nome-tabela [coluna1, coluna2,...]**

**VALUES (valor1, valor2,...);**

De acordo com a sintaxe apresentada, devemos especificar a tabela e quais colunas dessa tabela receberão os dados, e em seguida, logo após a palavra-chave **VALUES**, especificamos os dados que serão inseridos na mesma ordem em que as colunas foram especificadas.

Caso haja uma coluna com auto incremento, ela não deve ser incluída na lista de colunas do comando, pois seus dados serão gerados e inseridos automaticamente pelo MySQL quando um novo registro for adicionado.

## Comando UPDATE – Modificar Registros em Tabela

Este comando possibilita modificar registros em uma tabela do MySQL. É muito comum alterar o valor de um registro, por exemplo alterando o endereço de um cliente em um banco de dados, ou o telefone, ou o preço de um produto.

Sintaxe:

**UPDATE nome-*tabela***

**SET nome-coluna = novo\_valor\_armazenado**

**WHERE nome-coluna = valor\_filtro;**

**Observação importante:** Caso não seja usada a cláusula WHERE para filtrar os registros, todos os dados da coluna serão alterados!

### Comando SELECT

Podemos dizer que a tarefa mais importante que um banco de dados MySQL realiza, do ponto de vista do usuário, é permitir a realização de consultas aos dados armazenados. O comando básico que utilizamos para realizar as consultas é o comando SELECT.

Sintaxe:

**SELECT nome-coluna(s) FROM nome-tabela;**

Para ver todas as colunas podemos utilizar o \* no lugar das **nome-coluna(s)**. Então serão mostradas todas as colunas e seus respectivos dados.

### Cláusula WHERE

A cláusula **WHERE** permite filtrar registros nos resultados de uma consulta, de modo a trazer apenas as informações desejadas (e não o conteúdo completo das colunas).

**Sintaxe:**

**SELECT nome-colunas**

**FROMnome- tabela**

**WHERE nome-coluna = valor;**

**Cláusula ORDER BY**

A palavra-chave **ORDER BY** é usada para ordenar o conjunto-resultado de registros em um consulta SQL, tanto de forma ascendente quanto descendente.

**Sintaxe:**

**SELECT nome-colunas FROM nome-tabela**

**ORDER BY nome-coluna\_a\_ordenar;**

Podemos configurar a ordenação como ascendente (crescente) ou descendente (decrescente) com o uso das palavras ASC ou DESC:

* **ASC** – Ordem ascendente
* **DESC** – Ordem descendente (inversa)

### Ordem de Classificação por Tipo de Dados

A ordem padrão de classificação do ORDER BY (ASC) para os diversos tipos de dados é a seguinte:

* Valores numéricos são exibidos com os menores valores primeiro (do menor para o maior).
* Valores de data são mostrados com os valores menores primeiro, o que significa as datas mais antigas (do mais antigo para o mais recente).
* Valores de caracteres são exibidos em ordem alfabética.
* Quando houver valores nulos (NULL), eles serão mostrados por último (em sequências descendentes, com DESC, são mostrados primeiro

**Operadores AND, OR e NOT**

* Os operadores AND, OR e NOT são usados para filtrar registros baseados em mais de uma condição.
* O operador AND mostra um registro se ambas as condições forem verdadeiras.
* O operador OR mostra um registro se pelo menos uma das condições for verdadeira.
* O operador NOT é a negação de uma expressão (inverte seu estado lógico).

### Cláusula AS - ALIAS

Pode-se dar um nome diferente (amigável) a uma coluna ou tabela ao retornar o resultado de uma consulta, de modo que seja mais fácil ou intuitivo entender os dados retornados. Para isso, usamos a claúsula AS.

**Sintaxe:**

**SELECT nome-coluna**

**AS alias\_ nome-coluna**

**FROM nome-tabela AS alias\_nome-tabela;**

Podemos também aplicar um alias a uma coluna sem a necessidade de usar a palavra AS, bastando para isso inserir o alias desejado logo após o nome da coluna, sem separação por vírgulas.

**Sintaxe:**

**SELECT nome-coluna alias\_nome-coluna**

**FROM nome-tabela alias\_nome-tabela;**

Para aplicar alias em mais de uma coluna, basta acrescentá-las normalmente, separando-as por vírgulas, e incluindo os alias logo após o nome de cada coluna.

Além disso, podemos criar alias usando palavras compostas, incluindo espaços, bastando para isso envolver o alias entre aspas.

### Cláusula inner join

A cláusula JOIN é usada para combinar dados provenientes de duas ou mais tabelas do banco de dados, baseado em um relacionamento entre colunas destas tabelas.

Um INNER JOIN (ou simplesmente JOIN) permite obter registros com dados provenientes de duas ou mais tabelas relacionadas do banco de dados no MySQL. A sintaxe básica de um INNER JOIN em uma consulta é:

**SELECT nome-colunas**  
**FROM nome-tabela1**  
**INNER JOIN nome-tabela2**  
**WHERE nome-tabela1. nome-coluna= nome-tabela2. nome-coluna;**

Onde, **nome-tabela1. nome-coluna** é o nome da primeira tabela concatenado com um ponto e com o nome da coluna chave primária ou estrangeira da tabela, e **nome-tabela2. nome-coluna** é o nome da segunda tabela concatenado com um ponto e com a chave estrangeira ou primária dessa tabela que se relaciona com a chave da primeira tabela.

**Bibliografia**

Instalando o MYSQL

<https://www.youtube.com/watch?v=qWjtkdst7-g>

Curso de MYSQL

[**http://www.bosontreinamentos.com.br/curso-completo-de-mysql/**](http://www.bosontreinamentos.com.br/curso-completo-de-mysql/)