SQL com My SQL: Gerenciando as tabelas do banco de dados

Tipos de Dados

A entidade de um banco é a tabela, e ela é composta por colunas que possuem diversos tipos (categorias) e seus valores devem respeitar o tipo estabelecido na criação da tabela.

Vamos ver agora os principais tipos de dados no My SQL começando com os numéricos.

Tipo	Valor em Bytes	Menor Valor (Com Sinal) - Signed	Menor Valor (Sem snal) - Unsigned	Maior Valor (Com sinal) - Signed	Maor valor (Sem sinal) - Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2xE63	0	2xE63-1	2xE64 - 1

Propriedade UNSIGNED:

Não permite sinal no número. Por isso o conjunto de valores válidos aumentam.

Temos dois tipos quando se trata de dados numéricos, inteiro e decimal. No caso dos números inteiros, há diversos tipos que podem ser criados no My SQL e estão relacionados com o espaço necessário para armazenar determinado número no banco.

Quanto maior o espaço, maior o conjunto de números possíveis a serem armazenados.

Observando a tabela na coluna ***Tipo***, temos cinco categorias de números inteiros, ***TINYINT, SMALLINT, MEDIUMINT, INT e BIGINT***. No campo ao lado ***Valor em Bytes*** mostra o número de bytes gastos para armazenar um número de determinado tipo, como por exemplo, o ***BIGINT*** é necessário gastar 8 bytes para armazenar, já o ***TINYINT*** gasta 1 byte.

Quanto maior a quantidade de bytes que posso armazenar, maior o espaço de números com os quais posso trabalhar. Então, por exemplo, o *TINYINT* fica entre *-128* e *127*, se for criado um número dessa categoria e enviado para armazenar o número *300*, será retornado um erro, visto que esse tipo não suporta.

Existe uma propriedade no My SQL chamada de ***UNSIGNED***. Ele é um atributo ara números inteiros no MySQL utilizado para a definição de números positivos (sem sinal), visto que o sinal de menos (-) ocupa espaço de armazenamento.

Então, se informar para o MySQL que uma coluna específica é do tipo inteiro e sem sinal, o conjunto de números para armazenar nesse tipo de campo é maior, visto que não é preciso armazenar no computador os sinais e até mesmo de números positivos, já que não é reservado um espaço para arquivar o sinal.

Exemplificando, se o *TINYINT* for do tipo *UNSIGNED*, consigo guardar de *0* a *255* e se for com sinal do *-128* a *127*, perceba haver uma diferença de espaços com o ganho do sinal.

Agora vamos falar de tipos numéricos decimais.

Ponto Flutuante:

FLOAT – Precisão simples (4 bytes)
DOUBLE: Precisão dupla (8 bytes)

Exemplo: Se declaramos um campo FLOAT(7,4) se incluímos o número 999,00009 o valor armazenado será 999,0001

No caso dos decimais temos dois tipos de: ***Precisão fixa e Ponto Flutuante***. Significa que que será realizado um arredondamento quando o número de casas decimais for superior ao número permitido pelo banco de dados.

No ponto flutuante também temos dois tipos de valores decimais, o ***FLOAT***, que trabalha com uma precisão simples de 4 bytes e o ***DOUBLE***, com precisão dupla de 8 bytes. A diferença entre ambos é o tamanho de espaço de armazenamento.

O *DOUBLE* é mais usado para quando queremos ter números mais exatos nos cálculos, uma vez que ele faz arredondamentos mais precisos com números com mais casas decimais. Quando declaramos um número *FLOAT* ou *DOUBLE*, é possível especificar o número de dÍgitos e casas decimais.

Quando esse tipo é declarado, podemos especificar da seguinte forma ***FLOAT(6,2)*** significa que esse número do tipo *FLOAT* terá seis dígitos e duas casas decimais. É possível mais de quatro casas decimais, como por exemplo ***999,00009*** este com cinco casas. O My SQL atendendo a especificação, irá arredondar esse número para quatro casas decimais, armazenando o ***999,0001***.

O próximo tipo será o decimal fixo.

Fixos:

DECIMAL ou NUMERIC

Tamanho: Até 65 dígitos

Especificamos o número de dígitos e o número de casas decimais.

Exemplo: Se declaramos um campo DECIMAL(5,2) iremos poder armazenar valores somente entre -999,99 e 999,99.

Número máximo de dígitos são 65.

A definição é a quantidade máxima de dígitos que o número pode ter, que no caso é 65 dígitos. Isso inclui números inteiros e decimais. Eu posso ter um número inteiro e 64 casas decimais ou eu posso ter 64 números inteiros e uma casa decimal. O limite máximo são 65 dígitos.

Um exemplo, temos ***DECIMAL(5,2)*** o número é decimal com 5 dígitos e 2 casas decimais, só poderá ser armazenado números entre ***-999,99 e 999,99***, tal como é específico no tamanho. O valor exato ao tamanho será guardado no banco.

O próximo tipo será o ***BIT***.

Único:

BIT

Tamanho: Até 64 Bits

Exemplo: BIT(1) – Pode ser 1 ou 0

BIT(2) – Pode ser 01,10,00,11

O ***BIT*** armazena valores até 64 bytes. Se um campo ***BIT(1)***, é possível guardar ***1 ou 0***.

Se for ***BIT(2)*** é possível armazenar ***01,10,00 e 11***. Agora, se tiver um número de até *64 BITS*, tenho o tamanho de 64 caracteres que podem ser gravado de *1* ou *0* que representa um número binário.

O *BIT* é bastante utilizado em campos lógicos sendo o ***1 definido como TRUE e 0 definido como FALSE***.

Vamos falar mais sobre campos numéricos.

Atributos dos campos numéricos:

SIGNED ou UNSIGNED – Vai possuir ou não sinal no número

ZEROFILL – Preenche com Zeros os espaços:
Exemplo: INT(4). Se armazenarmos o valor 5 será gravado 0005.

AUTO_INCREMENT – Sequencia auto incrementada.
Exemplo: 1,2,3,4,5 .....

Erros de OUT FO RANGE:

Vão ocorrer quando os valores estourarem os limites.

Já foi mencionado antes o ***SIGNED ou INSIGNED***, que representam se haverá número negativo ou não, mas vamos explicar sobre o ***ZEROFILL***. Este preenche com zeros o que estiver faltando do número. Um exemplo, um campo ***INT(4)*** e pedir para solicitar o valor *5* será gravado o valor ***0005***.

Outro atributo é o ***AUTO_INCREMENT ou AUTO_INCREMENTO***. É uma propriedade do próprio banco de dados que gera uma sequência numérica automaticamente. Se tiver uma tabela vazia com um campo desse tipo e for inserido um valor nessa coluna será atribuído o ***valor 1***, se incluirmos mais um valor será gerado o ***valor 2*** e assim por diante.

Essa sequência se inicia no *1* ou no *0*, definido no próprio *AUTO_INCREMENT*, bem como, o valor do incremento que quero aplicar, podendo ser *0, 5, 10, 15, 20* ao invés de *1, 2, 3, 4*.

O ***OUT_OF RANGE*** é um erro que acontece quando tentamos gravar na base de dados um valor que está fora do espaço permitido.

Agora vamos falar sobre campos de ***Data e Hora***.

Data e Hora:

- DATE – 1000-01-01 até 9999-12-31.
- DATETIME – 1000-01-01 00:00:00 até 9999-12-31 23:59:59
- TIMESTAMP – 1970-01-01 00:00:01 UTC até 2038-01-19 UTC
- TIME – -838:59:59 e 839:59:59
- YEAR – 1901 – 2155 (Pode ser expresso no formato 2 ou 4 dígitos)

Temos cinco tipos o ***DATE, DATETIME, TIMESTAMP, TIME e YEAR.*** O ***DATE*** armazena um dia no formato ***ano, mês e dia ou YEAR-MONTH-DAY***. Vai de ***1000-01-01 até 9999-12-31.***

Já o ***DATETIME*** além de guardar a data, ele também guarda a hora, isso é importante quando tempos campos do tipo ***LOG***, que possui o horário em que alguém fez alguma ação específica no sistema.

O ***TIMESTAMP*** é bem semelhante ao ***DATETIME***, mas ele possuí duas características que o tornam diferente, uma delas é que o *TIMESTAMP* tem um tamanho (range) menor, de ***1970-01-01 até 2038-01-19***. Outra característica é que o *TIMESTAMP* possuí fuso horário. Por essa razão o tamanho é menor, para armazenar mais informações.

Por mais que esse range de datas pareça pequeno, vale lembrar que utilizamos fuso horário para sistemas, por exemplo, de agendas para marcar reuniões em empresas que tiver funcionários(as) em diversos países. Por isso, o range até 2038 não é tão pequeno quanto parece, visto que ninguém irá marcar uma reunião de hoje até o ano de 2038.

O ***TIME*** armazena somente o horário e tem um range, de *-838:59:59 a 839:59:59*. Normalmente, usamos só para gravar uma hora no relógio que vai de meia-noite às onze e cinquenta e nove da noite do dia seguinte. Por isso, não é preciso ter 838 horas gravadas.

No campo ***YEAR*** é guardado somente o ano de *1901 a 2155*. Podendo ser de duas ou quatro casas decimais, mas, normalmente utilizamos o *DATE* com uma data de primeiro de janeiro do ano que quero armazenar.

Vamos falar agora sobre ***STRINGS***

STRINGS:

- CHAR – Cadeia de caracteres com valor fixo (de 0 a 255).
- VARCHAR – Cadeia de caracteres com valor variado (de 0 a 255)

CHAR(4) – “aa” – “ aa”

VARCHAR(4) – “aa” – “aa”

STRINGS são cadeias de caracteres, os textos. Existem categorias, começando pelo ***CHAR e VARHCAR*** ambas possuem o limite de 255 caracteres, a única diferença entre os dois são os espaços.

Um exemplo, tenho um campo *CHAR(4)* e armazeno as letras “*aa”* no banco de dados será guardado dois espaços vazios, já que no campo foi estabelecido o caractere de tamanho quatro, ficaria "*aa*". Esse tipo ocupa espaço em disco já que grava os espaços vazios e isso pode ser desnecessário.

Caso fosse armazenado no *VARCHAR(4)* o *“aa”* será guardado somente dois caracteres sem gastar espaço a mais.

STRINGS:

- BINARY – Cadeia de caracteres com valor fixo (de 0 a 255). Expressos em Binário.
- VARBINAY – Cadeia de caracteres com valor variado (de 0 a 255). Expressos em Binário.

Temos também as *STRINGS* do tipo ***BINARY e VARBINARY***. Ambos possuem o mesmo conceito do *CHAR e VARCHAR*, um armazena espaços e o outro não. A diferença é que o armazenamento não é em caractere e sim em bytes, ou seja, o tamanho máximo é expresso em binário.

STRINGS:

- BLOB – Binário longo. Podemos ter:

- TINYBLOB
- BLOB
- MEDIUMBLOB
- LONGBLOB

- TEXT – Texto longo

- TINYTEXT
- TEXT
- MEDIUMTEXT
- LONGTEXT

Existem mais tipos de *STRINGS* como o ***TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT***.

O *BLOB* é binário, então, um *LONGBLOB* é possível guardar um grande binário no banco. Por exemplo, posso gravar bytes de uma foto ou bytes de um arquivo de word. Já o *TEXT* vai ser usado somente para armazenar textos.

STRINGS:

ENUM

- Permite armazenar uma lista pré-definida de valores.

- Exemplo:

Size ENUM('x-small', 'small', 'medium', 'large', 'x-large')

O ***ENUM*** permite guardar valores pré-definidos, como o exemplo mostrado na imagem. Os textos no *ENUM* têm que ficar entre parênteses.

Atributos dos campos String:

SET e COLLATE – Que tipo de conjunto de caracteres serão suportados.

Já foi mencionado no primeiro relatório que são os tipos ***SET e COLLATE***. Estão relacionados com as cadeias de caracteres que serão usados para armazenar o texto.

Caso queira guardar texto em alguma língua diferente é preciso selecionar na definição de campo o *SET e o COLLATE* especificando a língua utilizada, assim, será inserido no campo uma ***tabela ASCII*** maior ou menor dependendo do tipo que você está usando.

Criando Tabelas

Foi me passado que tenho que criar três tabelas ***Produto, Vendedor e Cliente***. Vou começar com a tabela cliente e nela vou utilizar a linguagem SQL. Como já mostrei aqui o comando para criar uma tabela é o ***CREATE TABLE***.

O que tem que ser armazenado na tabela *Cliente*:

- CPF
- Nome
- Data de Nascimente
- Idade
- Sexo/Gênero
- Limite de Crédito
- Volume mínimo de Compra
- Se é a primeira compra do cliente
- Endereço

Veja como ficou.

```
CREATE TABLE TblCliente
(
CPF VARCHAR(11) NOT NULL,
Nome_Cliente VARCHAR(90) NOT NULL,
Endereco_1 VARCHAR(150) NOT NULL,
Endereco_2 VARCHAR(150),
Bairro VARCHAR(50) NOT NULL,
Cidade VARCHAR(50) NOT NULL,
Estado VARCHAR(50) NOT NULL,
CEP VARCHAR(8) NOT NULL,
Idade SMALLINT NOT NULL,
Sexo VARCHAR(1) NOT NULL,
Limite_de_Credito FLOAT NOT NULL,
Volume_de_Compra FLOAT NOT NULL,
Primeira_Compra BIT(1) NOT NULL
);
```

Como você pode ver, resolvi deixar o CPF como *VARCHAR* já que um CPF pode ter pontos(.), traço(-), e alguns podem começar com zero. O Endereço foi quebrado em Bairro, Cidade, Estado e CEP.

O Limite de Crédito e Volume de Compra são valores então decidi coloca-los como *FLOAT*. Seguindo a orientação do instrutor eu não preciso me preocupar com o formato do valor, do número, ou do texto no momento de armazená-lo no banco de dados, mas sim no momento de exibir em um relatório, em um programa.

E aí eu tenho, não somente no SQL, mas também nas linguagens de programação com as quais vai trabalhar em conjunto com o banco de dados, de exibir a informação do jeito como ela tem que aparecer. Isto é, posso gravar um número com dez casas decimais, mas no momento em que eu for exibir, forço a exibir só com duas casas.

Agora o último campo o Primeira Compra resolvi colocar em *BIT*, o um(1) vai significar que é a primeira compra do cliente na empresa, já o zero(0) significa que o cliente já comprou na empresa.

Agora para a tabela ***Vendedor*** me foi passado os seguintes atributos:

- Matrícula
- Nome
- Percentual de Comissão

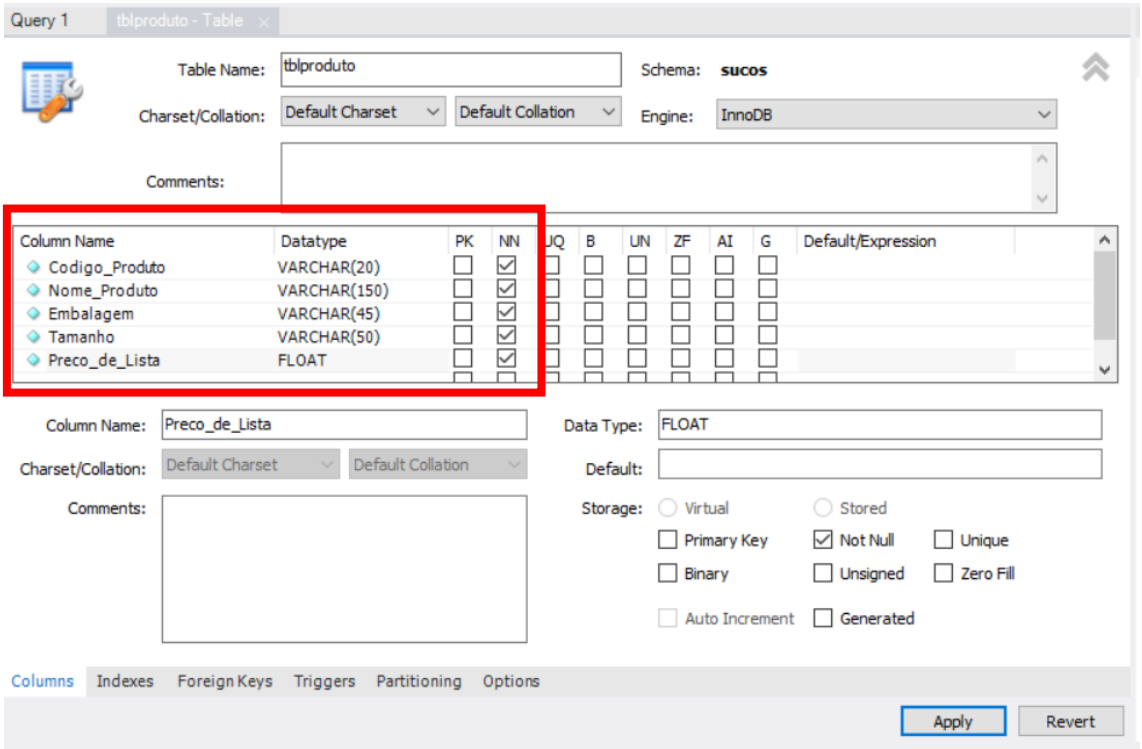
Veja como ficou no SQL

```
CREATE TABLE TblVendedor
(
Matricula VARCHAR(5) NOT NULL,
Nome_Vendedor VARCHAR(100) NOT NULL,
Percentual_Comissao FLOAT NOT NULL
);
```

Agora a última tabela, a tabela ***Produto*** e ela irá armazenar os seguintes dados:

- Código
- Nome
- Embalagem
- Tamanho
- Sabor
- Preço de Lista

Para criar essa tabela eu usei o assistente do My SQL observe a imagem abaixo.

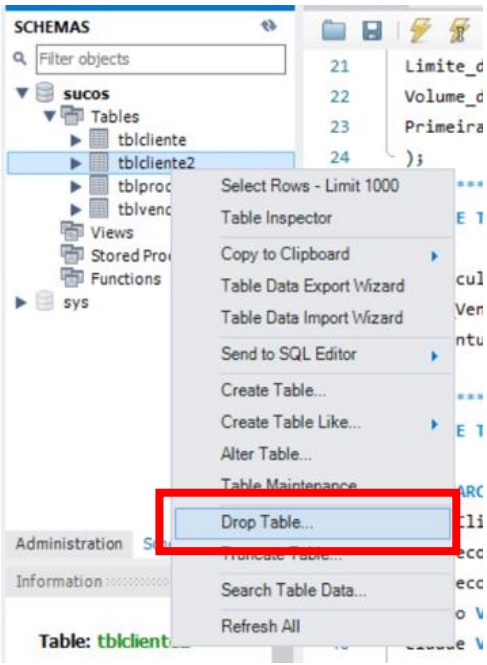


Depois de clicar em ***Apply***, irá aparecer uma outra aba com o código em SQL.

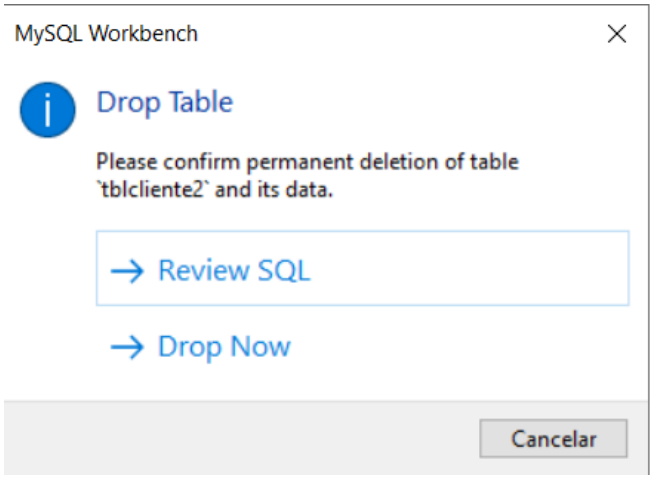
Depois é só clicar em *Apply* novamente que a tabela será criada.

Apagar Tabela

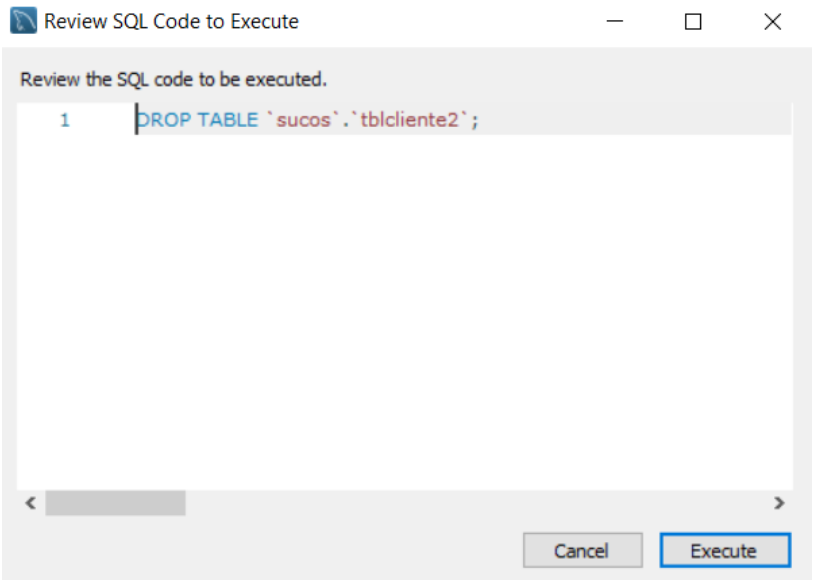
Vou mostrar dois métodos para apagar uma tabela no My SQL. Uma é pelo comando ***DROP TABLE table_name***. Outra é utilizar o assistente.



No assistente clique com o botão direito na tabela e vá em ***Drop Table***.



Depois irá aparecer uma aba e ela mostra duas opções, na primeira ***Review SQL*** ela irá te mostrar o código SQL para excluir a tabela.



Agora se você clicar em ***Drop Now***, a tabela irá ser eliminada. Tome cuidado pois esse comando é poderoso e se usado da maneira errada, você irá prejudicar e muito o seu banco.