BANCO DE DADOS

AULA 3

Prof. Ricardo Sonaglio Albano
Profª Silvie Guedes Albano

CONVERSA INICIAL

Nesta etapa, iremos continuar nossos estudos sobre a Linguagem de Consulta Estruturada (*Structured Query Language* – SQL), que começamos a estudar anteriormente. Abordaremos os tipos de dados, suas aplicações e conheceremos os principais comandos dessa linguagem. Também trataremos das restrições (*constraints*) e da declaração de cada tipo de chave em suas respectivas tabelas, explicando sobre as particularidades de cada uma.

Estudaremos a aplicação e o controle das colunas de autoincremento que são utilizadas para diferenciar linhas em uma tabela do Banco de Dados.

Vamos ingressar no desenvolvimento de um Banco de Dados, explicando passo a passo todo o processo, sempre com a aplicação de exemplos práticos e implementando o conhecimento adquirido em nosso estudo de caso.

TEMA 1 – SQL *DATA TYPES*

Vale lembrar que, anteriormente, definimos que o MySQL será o Sistema Gerenciado de Banco de Dados (SGBD) utilizado neste estudo.

Todo objeto que tem a função de armazenamento de dados sempre terá um tipo de dado associado ao mesmo, pré-definindo um único padrão de dado que o objeto poderá receber. Os objetos que têm tipos de dados associados são:

- Colunas em tabelas e exibições;
- Parâmetros em procedimentos;
- Variáveis;
- Funções que retornam um ou mais valores de dados de um tipo de dado específico.

O MySQL suporta uma diversidade de tipos de dados que podem ser assumidos por uma coluna. Esses tipos de dados podem ser agrupados em três categorias:

1. Tipo numérico.
2. Tipo cadeia de caracteres ou valor *string*.
3. Tipo valor temporal.

A atribuição de um tipo de dado a um objeto define as características de uma coluna, isto é:

- O tipo de dado contido pelo objeto;
- O comprimento ou tamanho do valor a ser armazenado (*bits* ou *bytes*);
- A precisão e a escala do número (apenas dados numéricos), em que:
  - Precisão – Refere-se ao total de dígitos em um número;
  - Escala – É o número de dígitos à direita da casa decimal.

Por exemplo, o número 578,91 tem uma precisão de 5 dígitos e uma escala de 2 (dois dígitos após a vírgula).

O comprimento de um tipo de dado numérico é o número de *bytes* usado para armazenar o valor de forma física (arquivo de dados). Por exemplo, um tipo de dados inteiro, contendo 10 dígitos, será armazenado em 4 *bytes* e não aceitará a inclusão de casas decimais. Dessa forma, o dado informado apresenta uma precisão de 10 dígitos, um comprimento de 4 *bytes* e uma escala de 0, pois não possui casas decimais.

Perceba que, no exemplo, estamos nos referindo a apenas um valor. Imagine então uma linha com diversas colunas, em que o tamanho armazenado no arquivo de dados seria o somatório em *bytes* de cada tipo de dado pertencente a cada coluna na linha. Assim, estimaremos o tamanho de uma tabela e, consequentemente, o volume que o Banco de Dados irá ocupar. Com o passar do tempo, calcularmos a taxa de crescimento.

Recorde que o espaço de armazenamento gera custo. Portanto, a definição de cada tipo de dado de forma correta é importante para o uso eficiente desse espaço. Essa percepção é uma das atividades executadas pelos administradores de Banco de Dados.

1.1 Tipo Numérico

Podem assumir os valores inteiros, ponto flutuante (sem definição de limite na precisão e escala) ou ponto fixo (limitação na precisão e escala).

Os tipos de dados considerados inteiros são: ***bit***, ***tinyint***, ***smallint***, ***int***, ***mediumint*** e ***bigint***. Em termos de espaço físico (armazenamento no arquivo de dados em disco), os tipos numéricos ocupam um comprimento de 1 a 8 *bytes*, exceto no caso do tipo *bit* que apresenta um armazenamento diferenciado, abrangendo um comprimento de 0 (zero) a 64 *bits* por valor armazenado.

No caso dos números de ponto flutuante, ou seja, que não possuem definição do número de dígitos, tanto no que se refere ao número inteiro

(precisão) quanto para as casas decimais após a vírgula (escala), os mesmos representados pelos tipos ***float*** e ***double*** que, no que se refere ao espaço físico, requerem de 4 e 8 *bytes*, respectivamente, para cada lado do valor (precisão e escala).

Finalmente, os dados numéricos de ponto fixo, com definição do máximo de dígitos presentes, tanto no número inteiro (precisão) quanto nas casas decimais (escala), são representados pelo tipo de dado ***decimal***, o qual necessita de 4 *bytes* de armazenamento para a precisão e o mesmo tamanho para a escala.

1.2 Tipo Cadeia de Caracteres ou Valor *String*

Os valores *string* são subdivididos em duas categorias:

1. Valores binários – *Binary*, *varbinary* e *blob* (16 *bits*, mais 2 de controle).
2. Valores não binários – Representados pelos tipos de dados ***char***, ***nchar***, ***varchar***, ***nvarchar*** e ***text***.

No que se refere ao comprimento, os valores não binários, geralmente, ocupam o equivalente ao número de caracteres definidos na coluna.

Aqui, devemos ressaltar uma característica presente no tipo de dados *varchar*, que apenas ocupa o tamanho realmente utilizado acrescido de um caractere de controle para identificar o fim da cadeia de caracteres, ou seja, é o mais econômico, pois desconsidera o espaço ocioso na cadeia de caracteres para situações em que a coluna fica mais vazia do que cheia.

Os tipos *nchar* e *nvarchar* seguem as mesmas características dos seus pares, mas possuem a capacidade de armazenar caracteres Unicode. Por exemplo, a coluna nome do cliente, com comprimento de 100 caracteres, pode ser definida de duas forma, isto é, “nomeCliente varchar(100)” ou “nomeCliente char(100)”. Vejamos as diferenças: se tivéssemos que armazenar, por exemplo, o nome “Zanana Silva”, seriam 12 caracteres; a coluna do tipo *char* irá ocupar os 100 caracteres definidos, já no tipo *varchar* irá ocupar 13 caracteres (12 do nome mais 1 para identificar o final da cadeia de caracteres).

Considere outra situação, em que se deseja armazenar a coluna sigla do estado com comprimento de 2 caracteres. Vejamos: “siglaEstado varchar(02)” ou “siglaEstado char(02)”. Se tivéssemos que armazenar, por exemplo, a sigla “PR” na coluna do tipo *char*, seriam ocupados os 2 caracteres definidos. Já no

tipo *varchar* seriam ocupados 3 caracteres (2 da sigla mais 1 para identificar o final da cadeia de caracteres).

Perceba que para colunas que a cadeia de caracteres irá ficar preenchida, o tipo *char* ocupará menos espaço. Já para as colunas que ficarão mais vazias do que preenchidas, o tipo *varchar* é mais adequado.

1.3 Tipo Valor Temporal

Utilizados no armazenamento de dados do tipo data e hora, isto é, dados ou valores referentes ao tempo (o que justifica seu nome).

Em termos de comprimento, eles requerem um tamanho de armazenamento de 1 a 8 *bytes*.

Os valores temporais são identificados pelos tipos ***date***, ***time***, ***datetime***, ***timestamp*** e ***year***, sendo:

- *Date* – Utilizado exclusivamente para datas sem a declaração de hora. Por exemplo, data de nascimento ou data de vencimento de um boleto. Formato padrão: YYYY-MM-DD (ano-mês-dia).
- *Time* – Apenas para hora. Formato padrão: hh:mm:ss (hora:minuto:segundo).
- *Datetime* – Utilizado para valores de data e hora, baseando-se no calendário e hora do relógio. Não considera o fuso horário. Por exemplo, o prazo de entrega da declaração do Imposto de Renda é até às 23h00min de um determinado dia, sendo de responsabilidade do usuário saber se está ou não, por exemplo, no horário de verão, fuso horário do local de entrega. Formato padrão: YYYY-MM-DD hh:mm:ss.
- *Timestamp* – Semelhante ao *datetime*, porém baseado no fuso horário. É um número sequencial começando em 01/01/1970 00:00 de Londres, ou seja, no horário do Brasil seria 21:00 do dia 21/12/1969. O *timestamp* pode variar de acordo com a tabela abaixo.

Tamanho	Formato
14	AnoMesDiaHoraMinutoSegundo aaaammddhhmmss
12	AnoMesDiaHoraMinutoSegundo aammddhhmmss
8	AnoMesDia aaaammdd
6	AnoMesDia aammdd
4	AnoMes aamm
2	Ano aa

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

- *Year* – Armazena apenas o ano, podendo utilizar 2 ou 4 dígitos.

Esses são os tipos de dados mais comuns, ou seja, os mais utilizados. Contudo, vale salientar que há variações de tipos de dados de um SGBD para outro. Sempre é bom consultar a documentação específica de cada SGBD para conhecer os tipos de dados suportados.

TEMA 2 – SQL NA PRÁTICA

A partir deste momento, estudaremos os principais comandos SQL, incluindo exemplos para que você possa acompanhar de forma prática e clara. Lembrando que todos os comandos que serão apresentados referem-se ao SGBD MySQL.

2.1 Comandos Úteis para Iniciar

Reunimos alguns comandos úteis que facilitaram a utilização dos Bancos de Dados. São eles:

Apresentando os Bancos de Dados existentes:

Sintaxe: `show databases;`

Onde:

show	Comando de apresentação.
databases	Bancos de Dados existentes.
;	Ponto e vírgula, determina o final do comando (esse sinal não é obrigatório nas interfaces gráficas).

Acessando um Banco de Dados (conexão):

Sintaxe: `use bdados;`

Onde:

use	Comando de acesso a um Banco de Dados existente.
bdados	Identificação do nome do Banco de Dados a ser acessado.
;	Ponto-e-vírgula, determina o final do comando.
Exemplo:	**`use delivery;`**

Descobrindo o Banco de Dados que você está conectado:

Sintaxe:	**`select database();`**
Onde:	
select	Comando de seleção/consulta.
database()	Banco de Dados conectado no momento.
;	Ponto-e-vírgula, determina o final do comando.

Apresentando as tabelas existentes dentro de um Banco de Dados:

É importante ressaltar que para executar esse comando é necessário estar dentro do Banco de Dados desejado, utilizando o comando ***use***.

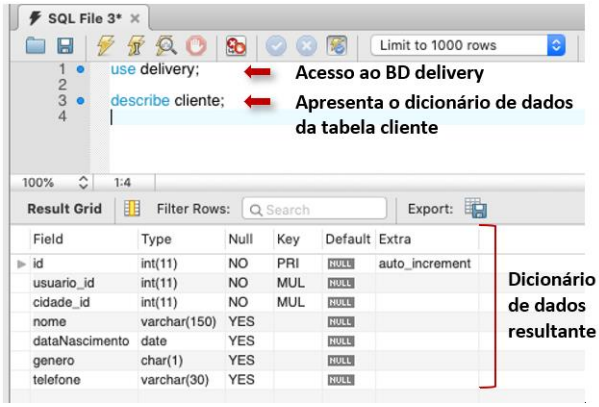
Sintaxe:	**`show tables;`**
Onde:	
show	Comando de apresentação.
tables	Tabelas pertencentes ao Banco de Dados.
;	Ponto-e-vírgula, determina o final do comando.

Apresentando o dicionário de dados de uma tabela:

O dicionário de dados contém a estrutura lógica da tabela, ou seja, o nome da coluna, o tipo de dado, a chave, entre outros.

Sintaxe:	**`describe tabela;`**
Onde:	
describe	Comando de descrição.
tabela	Nome da tabela desejada.
;	Ponto-e-vírgula, determina o final do comando.
Exemplo:	describe cliente;

Figura 1 – Dicionário de dados da tabela Cliente (ferramenta MySQL Workbench)



Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

2.2 Principais Comandos

Dando continuidade ao nosso estudo, abordaremos os comandos mais utilizados no dia a dia, juntamente com sua sintaxe e aplicação na prática.

O primeiro passo é criarmos os objetos necessários, ou seja, iniciando com Banco de Dados e as tabelas. O comando que iremos utilizar chama-se ***create***.

Vale salientar que esse mesmo comando pode ser usado para a criação de usuários, *views*, entre outros objetos.

Criando um Banco de Dados (*create database*):

Sintaxe:	`create database bdados;`
Onde:	
create	Comando de criação.
database	Declaração que será criado um Banco de Dados.
bdados	Nome do Banco de Dados. Vale salientar que o nome escolhido não deve pertencer a um Banco de Dados existente, nem ser uma palavra reservada.
;	Ponto-e-vírgula, determina o final do comando.
Exemplo:	`create database delivery;`

Dica: Podemos incrementar a *query*, referente ao *create*, com o objetivo de realizar uma pré-verificação da existência de um Banco de Dados com o mesmo nome. Para isso, basta utilizar o comando *if not exists* (se não existir).

Sintaxe: `create database [if not exists] bdados;`

Essa *query* somente criará o Banco de Dados se o mesmo não existir.

Criando tabelas (*create table*):

Agora que temos o nosso primeiro Banco de Dados criado, o próximo passo é criar as tabelas (relações) pertencentes ao Banco de Dados. Para isso, usaremos o mesmo comando *create*, declarando o nome da tabela juntamente com a definição de todas as colunas.

Sintaxe:
```
create table ntabela (
        coluna1 tipodado(tam) restrições, ...,
        colunan tipodado(tam) restrições);
```

Onde:

create	Comando de criação.
table	Declaração que será criada uma tabela.
ntabela	Nome da tabela.
(...)	Parênteses que contém o grupo de colunas da tabela, juntamente com a definição de sua estrutura.
coluna1...n	Declaração de um nome para a coluna.
tipodado(tam)	Tipo de dado correspondente a coluna e ao tamanho.
restrições	Definição de chaves, *not null*, *unique*, *default* e *check*.
,	Vírgula, separador de colunas.
;	Ponto e vírgula, determina o final do comando.

Exemplo (versão simples):
```
create table usuario (
        id integer,
        email varchar(100),
        senha varchar(20));
```

Analisando a *query* apresentada, percebemos que ainda existem alguns detalhes que não foram definidos, como por exemplo, a chave primária. Essa será uma tarefa abordada no Tópico 3 desta etapa.

Até o presente momento aprendemos como criar um Banco de Dados e suas tabelas. Nosso próximo passo é estudarmos como realizar a inclusão de dados nas mesmas.

Incluindo dados na tabela (*insert*):

O comando de inclusão de dados em uma tabela denomina-se *insert*. Os valores de cada coluna na linha também devem pertencer ao mesmo domínio definido na criação da tabela.

Sintaxe:
```
insert into ntabela (
```

```
coluna1, ..., colunan) values
(valor1, ..., valorn);
```

Onde:

insert into	Inclusão dos dados acompanhado da cláusula ***into***.
ntabela	Nome da tabela que receberá a inclusão.
(...)	Parênteses que contêm o grupo de colunas da tabela.
coluna1...n	Declaração do nome de cada coluna.
values	Cláusula que inicia o bloco de declaração dos valores.
(...)	Parênteses que contêm os valores das colunas.
valor1...n	Definição do valor (conforme domínio) de cada coluna.
;	Ponto e vírgula, determina o final do comando.

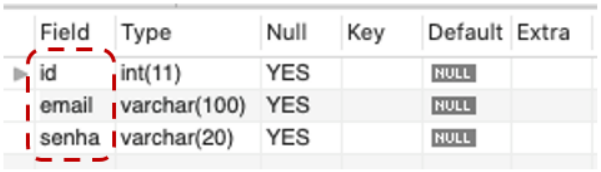
No que se refere aos valores, é necessária uma observação, isto é, com exceção dos valores numéricos, os demais serão declarados dentro de aspas simples.

O *insert* permite a escrita de uma *query* mais simples, omitindo o nome das colunas. Porém, apenas se todos os dados forem declarados na mesma sequência que foram definidos na criação da tabela. Na dúvida, use o *describe*.

Sintaxe:

```
insert into ntabela values
(valor1, ..., valorn);
```

Figura 2 – Dicionário de dados da tabela Usuário (ferramenta MySQL Workbench)



Field	Type	Null	Key	Default	Extra
id	int(11)	YES		NULL	
email	varchar(100)	YES		NULL	
senha	varchar(20)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Exemplo (tabela Usuário):

```
insert into usuario (id, email, senha) values
(1, 'maria.lopes@email.com', 'M1256779@');
```

Perceba que os dados inseridos seguem a mesma ordem de declaração da tabela Usuário, ou seja, informando primeiro o valor da coluna 'id' (*integer*), após o valor da coluna 'email' (*varchar*) e, por fim, o valor correspondente à coluna 'senha' (*varchar*).

O comando *insert* também permite a inserção de dados em colunas específicas de uma tabela, desde que sejam respeitadas as restrições de cada coluna. Exemplo:

```
insert into usuario (id, senha) -- Exceto a coluna email
     values (2, 'JJ886645@');
```

Agora que a nossa tabela possui dados, isso nos leva a outra questão: Como visualizar os dados armazenados na tabela?

Consultando dados na tabela (*select*):

O comando *select* é o responsável pela consulta dos dados na tabela, retornando o resultado conforme estabelecido no comando.

Sintaxe:	`select * from ntabela;`
Onde:	
select	Comando de consulta de dados em uma ou mais tabelas.
*	Indica que a consulta se refere a todos os dados da tabela.
from	Cláusula que precede o nome da tabela e significa “de”.
ntabela	Nome da tabela que recebe a solicitação de consulta.
;	Ponto e vírgula determina o final do comando.
Exemplo:	`select * from usuario;`

Figura 3 – *Select* na tabela Usuário (ferramenta MySQL Workbench)

id	email	senha
1	maria.lopes@email.com	M1256779@
2	NULL	JJ886645@

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Perceba que na segunda linha, a coluna ‘email’ não possui nenhum valor (*null*, vazio), pois o valor dessa coluna não foi declarado no comando *insert*.

O mesmo comando *select* possibilita solicitar diversos tipos de consultas, como por exemplo, consultas em determinadas colunas, ordenar o resultado em ordem crescente ou decrescente, realizar filtros, entre outras.

Para consultar os dados de colunas específicas de uma tabela, basta informar os respectivos nomes. Por exemplo, retornar somente os dados das colunas ‘email’ e ‘senha’:

```
select email, senha from usuario;
```

Para apresentar os dados em uma determinada ordem (ascendente ou descendente), utilizamos a expressão ***order by***. Por exemplo, retornar uma

consulta abrangendo apenas o e-mail e a senha do usuário, ordenando por ordem alfabética (A-Z) da coluna ‘email’:

```
select email, senha from usuario order by email;
/*Se houver algum valor NULO dentro da coluna que será
ordenada, o mesmo será apresentado antes que os demais.*/
```

Figura 4 – *Select* e *order by* na tabela Usuário (ferramenta MySQL Workbench)

email	senha
NULL	JJ886645@
maria.lopes@email.com	M1256779@

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Também é possível apresentar as linhas em ordem descendente (DESC). Inclusive, o padrão é ascendente (ASC), logo, não precisa ser declarado.

Vale salientar que a cláusula *order by* é a última clausula da instrução *select*, ou seja, sempre estará no final da instrução. Exemplo:

```
select * from usuario order by id desc;
-- ordenará por ordem descendente.
```

Para realizar um filtro no *select*, utiliza-se a cláusula *where*, a qual permite a definição de condições para realizar a consulta. Por exemplo, qual o usuário que não possui e-mail registrado?

```
select * from usuario where email is null;
```

Na cláusula *where*, também podemos aplicar os operadores lógicos, relacionais e booleanos. Perceba que no exemplo estamos procurando o valor nulo na coluna ‘email’. Nesse caso, não usamos o operador = (igual), mas a cláusula *is*.

Apesar de estarmos construindo nossos objetos, talvez você se depare com a necessidade de eliminar algum objeto (Banco de Dados, tabelas, usuários, entre outros). Esse é um procedimento que exige cautela na decisão, pois a exclusão pode ser irreversível. A seguir trataremos desse assunto.

Excluindo objetos em SQL (*drop*):

O comando *drop* permite a exclusão de objetos, seja um Banco de Dados, tabelas, usuários, *views*, entre outros. Sempre tenha muita atenção ao utilizar comandos de eliminação.

Excluindo uma tabela:

Sintaxe: **`drop table ntabela;`**

Onde:

drop — Comando de exclusão.

table — Declaração que será a exclusão uma tabela.

ntabela — Nome da tabela a ser excluída. Vale salientar que a mesma deve existir.

; — Ponto e vírgula, determina o final do comando.

Exemplo: **`drop table estado;`**

Após a execução dessa linha de comando, a referida tabela (Estado) estará eliminada, juntamente com todos os seus dados.

Excluindo um Banco de Dados:

Sintaxe: **`drop database bdados;`**

Onde:

drop — Comando de exclusão.

database — Declaração que será excluído um Banco de Dados.

bdados — Nome do Banco de Dados a ser excluído. Obviamente, o Banco de Dados deverá existir para ser excluído.

; — Ponto e vírgula, determina o final do comando.

Exemplo: **`drop database delivery;`**

Após a execução da *query*, o Banco de Dados estará completamente excluído e isso incluirá todos os objetos (tabelas, visões, usuários, procedimentos, entre outros) pertencentes a ele e, consequentemente, todos os dados armazenados.

Importante: para excluir um Banco de Dados, o mesmo não poderá estar em uso, isto é, você não pode estar acessando o Banco de Dados que deseja excluir.

TEMA 3 – SQL PK, FK e UK

Como aprendemos anteriormente, as chaves fazem parte das restrições de integridade de um Banco de Dados. As restrições são classificadas como:

- Chave primária (PK);
- Chave estrangeira (FK);
- Chaves compostas;
- *Unique*;

- *Not null*;
- *Check*;
- *Default*.

Neste tópico, iremos estudar apenas as chaves, as demais restrições serão abordadas ainda nesse material de estudo.

Declarando uma chave primária (PK) em uma tabela:

Primeiramente devemos ressaltar que quando criamos uma restrição de chave primária, todas as colunas incluídas na chave primária devem possuir a restrição ***not null*** declarada, impedindo que a chave primária possa armazenar valores nulos. Também existe a opção de inclusão do autoincremento na chave primária, mas tal conceito será empregado posteriormente.

Sintaxe:
```
coluna tipodado restrição,
    primary key(coluna)
```

Onde:

coluna	Declaração da coluna.
tipodado	Definição do tipo de dado da coluna.
restrição	Declarações das restrições (exemplo, *not null*).
primary key	Atribuindo a restrição *Primary Key* (PK).
(coluna)	Nome da coluna que se tornará *primary key*.

Exemplo:
```
create table estado (
    id integer not null,
    descricao varchar(100),
    sigla char(02),
    primary key(id));
```

Dicionário de dados resultante:

Figura 5 – Chave primária da tabela Estado (ferramenta MySQL Workbench)

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
descricao	varchar(100)	YES		NULL	
sigla	char(2)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Perceba que a coluna ‘Key’ apresenta a informação ‘PRI’, identificando que o ‘id’ é a chave primária da tabela.

Declarando chaves estrangeiras (FK) em uma tabela:

Sintaxe:

```
coluna tipodado restrição,
    foreign key(coluna)
    references tabOrigem (colOrigem)
    comportamento
```

Onde:

coluna	Nome da coluna que faz relação com outra tabela.
tipodado	Definição do tipo de dado da coluna.
restrição	Declarações das restrições (exemplo, *not null*).
foreign key	Atribuição na coluna da restrição chave estrangeira.
(coluna)	Nome da coluna que atuará como chave estrangeira.
references	Cláusula que declara a referência a outra tabela.
tabOrigem	Nome da tabela que possui a coluna como chave primária. Como sabemos, uma chave estrangeira faz referência a uma chave primária de outra tabela que se relaciona.
(colOrigem)	Coluna (chave primária) que está sendo referenciada como chave estrangeira na tabela destino.
comportamento	*On delete no action on update no action*.

Em uma tabela com chave estrangeira, deve-se aplicar as restrições de integridade, nesse caso, as restrições “*on delete*” (exclusão) e “*on update*” (modificação), que recebem o comportamento “*no action*” (sem ação). Esses comportamentos referem-se à ação que será executada pelo Banco de Dados quando o usuário tentar excluir ou alterar o valor de uma chave primária que está sendo utilizada em outra tabela como chave estrangeira. Geralmente, não é possível alterar ou excluir um valor que possua uma referência com outra tabela, exceto em cenários especiais, utilizando *cascade*, mas esse é um tema para estudarmos mais à frente.

Exemplo:

```
create table cidade (
id integer not null,
estado_id integer not null,
descricao varchar(150),
primary key(id),
foreign key(estado_id) references estado(id)
on delete no action on update no action);
```

Dicionários de dados:

Figura 6 – Chave estrangeira, tabelas Estado e Cidade (ferramenta MySQL Workbench)

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Avaliando o dicionário de dados da tabela Cidade, verificamos que a coluna ‘Key’ apresenta a informação ‘MUL’, identificando que o ‘estado_id’ tem a função de chave estrangeira na tabela, correspondendo ao ‘id’ (chave primária) da tabela Estado.

Logo a seguir utilizaremos o comando *insert* na tabela Cidade para exemplificar o comportamento da relação.

Figura 7 – *Select* nas tabelas Estado e Cidade (ferramenta MySQL Workbench)

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Comandos:

```
insert into cidade values (1, 3, 'Curitiba');
insert into cidade values (2, 1, 'Porto Alegre');
insert into cidade values (3, 4, 'Brasília');
```

Figura 8 – Erro no *insert* da tabela Cidade (ferramenta MySQL Workbench)

	Action	Response
✔ 1	insert into cidade values (1, 3, 'Curitiba')	1 row(s) affected
✔ 2	insert into cidade values (2, 1, 'Porto Alegre')	1 row(s) affected
✖ 3	insert into cidade values (3, 4, 'Brasília')	Error Code: 1452. Cannot add or update a child row: a foreign key...

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Ao executar os três comandos *insert*, as duas primeiras linhas serão incluídas com sucesso na tabela Cidade, mas a terceira linha resulta em um erro. O motivo é a tentativa de inserir o valor 4 na coluna 'estado_id' (tabela Cidade), valor que não existe na tabela Estado. Pelo princípio de integridade dos dados, não é possível inserir um valor na tabela filho (Cidade) que não exista na tabela pai (Estado).

Figura 9 – *Select* resultante das tabelas Estado e Cidade (ferramenta MySQL Workbench)

Tabela Estado

id	descricao	sigla
1	Rio Grande do Sul	RS
2	Santa Catarina	SC
3	Paraná	PR
NULL	NULL	NULL

Tabela Cidade

id	estado_id	descricao
1	3	Curitiba
2	1	Porto Alegre
NULL	NULL	NULL

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Declaração de chaves únicas (*Unique Key* – UK) em uma tabela:

Assim como a chave primária, a chave única (*unique key*) baseia-se no conceito de unicidade, ou seja, garantindo a exclusividade de cada linha em uma tabela. A diferença entre as duas está na aplicação, isto é, só é permitido uma única PK na tabela (mesmo em chaves compostas), mas é possível declarar diversas *unique* em uma mesma tabela.

Sintaxe: `coluna tipodado(tamanho) unique`

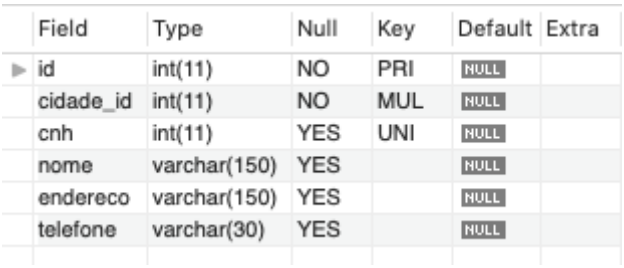
Exemplo:

```
create table entregador (
        id integer not null,
        cidade_id integer not null,
        cnh integer unique,
        nome varchar(150),
        endereco varchar(150),
        telefone varchar(30),
```

```
primary key(id),
foreign key(cidade_id) references cidade(id)
  on delete no action on update no action);
```

Dicionário de dados resultante:

Figura 10 – *Unique* na tabela Entregador (ferramenta MySQL Workbench)



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
cidade_id	int(11)	NO	MUL	NULL	
cnh	int(11)	YES	UNI	NULL	
nome	varchar(150)	YES		NULL	
endereco	varchar(150)	YES		NULL	
telefone	varchar(30)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Perceba que a coluna ‘Key’ apresenta a informação ‘UNI’, identificando ‘cnh’ como restrição única na tabela.

Testando a *unique*:

```
insert into entregador values (1, 1, 1100992211,
  'Wilquison', 'Rua da flores, 100', '(99)-99887-6655');
insert into entregador values (2, 1, 1100992211,
  'Zanana', 'Avenida 7, 1000', '(41)-99453-2190');
```

O SGBD insere com sucesso a primeira linha e, ao executar o segundo *insert*, detecta que o ‘cnh’ informado já existe e, dessa forma, apresenta um erro.

Figura 11 – Erro de restrição *Unique* (ferramenta MySQL Workbench)

	Action	Response
1	insert into entregador values (1, 1, 1100992211, 'Wilquison', 'Rua da flores, 100', '(99)-99887-6655')	1 row(s) affected
2	insert into entregador values (2, 1, 1100992211, 'Zanana', 'Avenida 7, 1000', '(41)-99453-2190')	Error Code: 1062. Duplicate entry '1100992211' for key 'cnh'

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Declaração de chaves compostas em uma tabela:

Em SQL, uma chave composta é caracterizada pela união de várias chaves primárias, ou seja, a declaração de uma PK composta de várias colunas.

Sintaxe:

```
constraint nomeRestricao
    primary key (coluna1, ..., colunan)
```

Onde:

constraint — Declaração da cláusula.

nomeRestricao — Definição de um nome que irá identificar a cláusula criada

para posterior manipulação. É um identificador único.

primary key — Identifica as colunas que receberão a restrição PK.

(coluna1...n) — Nome de todas as colunas que compõem a chave.

Exemplo:

```
create table pedidoProduto (
    pedido_id integer not null,
    produto_id integer not null,
    quantidadeProduto integer,
    valorUnitario decimal,
    desconto decimal,
    primary key(pedido_id, produto_id),
    foreign key(pedido_id) references pedido(id)
      on delete no action on update no action,
    foreign key(produto_id) references produto(id)
      on delete no action on update no action);
```

Dicionário de dados resultante:

Figura 12 – Chave composta da tabela pedidoProduto (ferramenta MySQL Workbench)

Field	Type	Null	Key	Default	Extra
pedido_id	int(11)	NO	PRI	NULL	
produto_id	int(11)	NO	PRI	NULL	
quantidadeProduto	int(11)	YES		NULL	
valorUnitario	decimal(10,0)	YES		NULL	
desconto	decimal(10,0)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

No dicionário de dados da tabela pedidoProduto perceba a indicação de duas colunas (pedido_id e produto_id) como chaves primárias (PRI), o que caracteriza uma chave composta.

TEMA 4 – SQL *CONSTRAINTS*

Sabemos que toda restrição é uma regra com a qual os dados devem estar em conformidade, ou seja, a restrição é utilizada para limitar e padronizar o recebimento de um dado, evitando, assim, que dados inválidos sejam inseridos no banco. As restrições dividem-se em:

1. Restrição em nível de coluna – Fazendo referência a uma única coluna da tabela.
2. Restrição em nível de tabela – Referenciam uma ou mais colunas da tabela, especificando as colunas aos quais se aplicam.

Já discutimos sobre as chaves, a seguir prosseguiremos com a declaração das demais restrições (*not null*, *check* e *default*).

Restrição *not null*:

Sempre que a restrição *not null* for associada a uma coluna, impedirá que a coluna aceite um valor nulo (vazio).

Sintaxe:
```
coluna tipo de dado(tamanho) not null
```

Exemplo:
```
create table usuario (
    id integer not null,
    email varchar(100) not null,
    senha varchar(20) not null,
    primary key(id));
```

Testando o *not null*:

```
insert into usuario (id, senha) values (2, 'JJ886645@');
```

Figura 13 – Erro *not null* na tabela Usuário (ferramenta MySQL Workbench)

	T	Action	Response
⊗	1	2 insert into usuario (id, senha) values (2, 'JJ886645@')	Error Code: 1364. Field 'email' doesn't have a default value

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Observe que a execução do comando *insert* retornou um erro, isso porque não foi declarado um valor para a coluna ‘email’, que é de preenchimento obrigatório (*not null*).

Restrição *check*:

No Sistema Gerenciador de Banco de Dados (SGBD) MySQL, a restrição *check* é ignorada, ou seja, não é implementada. Contudo, é importante que você saiba que essa restrição existe e é implementada em outros SGBDs.

A restrição *check* específica regras para os valores da coluna, validando a mesma. É usada para limitar a entrada de dados. Por exemplo, o estado civil (**S**olteiro, **C**asado, **V**iúvo, entre outros) e o gênero (**M**asculino e **F**eminino) de uma pessoa.

Sintaxe:
```
coluna tipodado(tamanho)
check(coluna in('valor1',...,'valorn'))
```

Dentro da restrição *check* é necessário informar a coluna, que receberá a validação, a cláusula *in* e os valores correspondentes (todos entre parênteses).

Exemplo: **`genero char(01) check(genero in('M', 'F'))`**

Restrição *default*:

Definição de um valor padrão associado a uma coluna. Caso não seja informado um valor para a coluna, o valor padrão será adicionado.

Sintaxe: **`coluna tipodado(tamanho) default valorpadrao`**

A declaração *default* é bastante simples, apenas sendo necessária sua referência e ao lado o valor padrão que a coluna irá assumir (caso não seja fornecido).

Exemplo:
```
create table cliente (
            id integer not null,
            nome varchar(150),
            dataNascimento date,
            genero char(01),
            estadoCivil char(01) default 'S',
            primary key(id));
```

Testando o *default*:

```
insert into cliente (id, nome, dataNascimento, genero)
  values (1,'Zanana','1999-12-31','F');
```

Figura 14 – *Select* do *default* na tabela Cliente (ferramenta MySQL Workbench)

	id	nome	dataNascimen...	genero	estadoCivil
▶	1	Zanana	1999-12-31	F	S

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

No comando *insert,* não foi declarado o valor da coluna 'estadoCivil', dessa forma, o valor *default* 'S' é atribuído a coluna.

É importante ressaltar que podemos aplicar várias restrições em uma mesma coluna. Por exemplo:

```
genero char(01) check(genero in('M','F')) default 'M'
```

Importante: as restrições não precisam ser especificadas apenas no momento da criação de uma tabela (*create table*), podem também serem aplicadas em tabelas existentes utilizando o comando *alter table* que estudaremos na sequência.

TEMA 5 – ALTERAÇÕES, *AUTO INCREMENT*, ENTRE OUTROS

Neste tópico, estudaremos como realizar modificações na estrutura de tabelas existentes e incluir propriedades especiais as colunas.

5.1 *Alter Table*

Comando que permite ao usuário alterar uma tabela, ou seja, realizar modificações nos elementos existentes dentro de uma tabela específica.

A utilização desse comando permite que, em casos em que ocorrer a especificação incorreta ou incompleta de uma coluna, não seja necessário a exclusão de toda a tabela e, consequentemente, a perda dos dados nela armazenados.

A sintaxe do *alter table* sempre é acompanhada por um dos seguintes comandos, que determinará a ação a ser realizada:

- *Add* – Adiciona uma coluna em uma tabela existente;
- *Change* – Renomeia e altera as definições de uma coluna existente (nome, tipo da coluna, entre outros);
- *Modify* – Modifica as definições de uma coluna existente, mas não permite alterar o seu nome;
- *Alter* – Altera uma coluna existente em uma tabela;
- *Drop* – Exclui uma coluna pertencente a uma tabela.

Adicionando uma coluna em uma tabela:

Sintaxe:
```
alter table ntabela
    add ncoluna tipodado(tamanho);
```

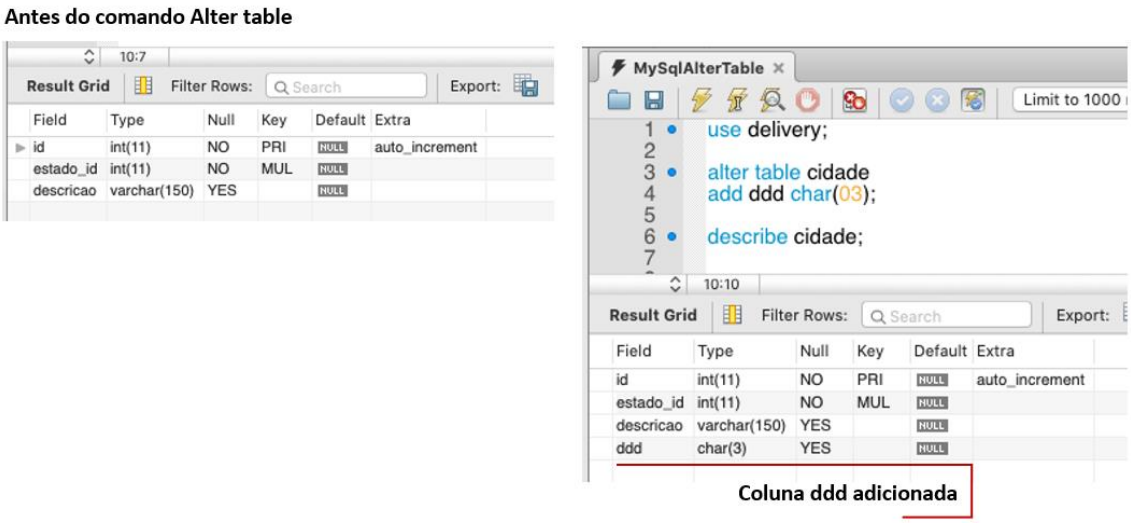
Onde:

alter table	Comando de alteração de tabela.
ntabela	Nome da tabela (existente) que sofrerá a alteração.
add	Comando de adição de uma coluna.
ncoluna	Identificação do nome da nova coluna.
tipodado	Especificação do tipo de dado que a nova coluna irá assumir.
(tamanho)	Definição do tamanho.
;	Ponto e vírgula, determina o final do comando.

Exemplo:
```
alter table cidade add ddd char(3);
```

Execute a *query* e avalie o dicionário de dados da tabela Cidade utilizando o comando correspondente: **`describe cidade;`**

Figura 15 – Comando *alter table* na tabela Cidade (ferramenta MySQL Workbench)



Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

É importante ressaltar que na inclusão de novas colunas não possível adicionar a restrição *not null* em uma tabela existente que já possua dados armazenados, uma vez que as linhas existentes não poderiam fornecer valores a nova coluna e, obrigatoriamente, não pode permanecer vazia.

Alterando a estrutura de uma coluna:

Sintaxe:
```
alter table ntabela
    modify colunaatual tipodado(tam);
```

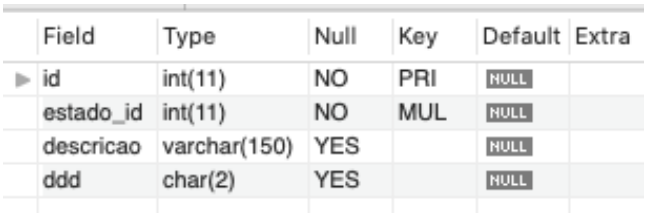
Onde:

alter table	Comando de alteração de tabela.
ntabela	Nome da tabela (existente) cuja coluna sofrerá a alteração.
modify	Comando que altera a estrutura de uma coluna pré-definida.
colunaatual	Nome da coluna que sofrerá a alteração.
tipodado(tam)	Tipo de dado, juntamente com seu tamanho.
;	Ponto e vírgula, determina o final do comando.

Exemplo: **`alter table cidade modify ddd char(02);`**

Após a execução da *query*, verifique o dicionário de dados da tabela Cidade utilizando o comando correspondente: **`describe cidade;`**

Figura 16 – *Alter table modify* na tabela Cidade (ferramenta MySQL Workbench)



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
estado_id	int(11)	NO	MUL	NULL	
descricao	varchar(150)	YES		NULL	
ddd	char(2)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Renomeando uma coluna:

Sintaxe:
```
alter table ntabela
  change colunaatual colunanova tipo(tamanho);
```

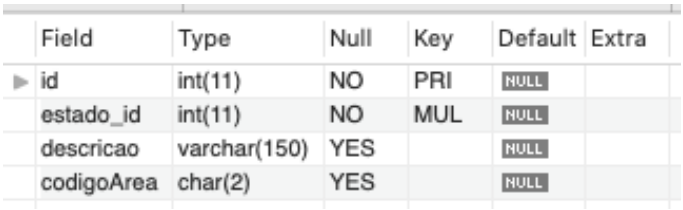
Onde:

alter table	Comando de alteração de tabela.
ntabela	Nome da tabela onde se encontra a coluna.
change	Comando que irá renomear o nome da coluna.
colunaatual	Coluna (existente) que sofrerá a alteração de nome.
colunanova	Novo nome da coluna.
;	Ponto e vírgula, determina o final do comando.

Exemplo:
```
alter table cidade
  change ddd codigoArea char(02);
```

Dicionário de dados resultante da tabela Cidade:

Figura 17 – Alterando o nome da coluna na tabela Cidade (ferramenta MySQL Workbench)



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
estado_id	int(11)	NO	MUL	NULL	
descricao	varchar(150)	YES		NULL	
codigoArea	char(2)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Eliminando uma coluna de uma tabela:

Sintaxe:
```
alter table ntabela drop column ncoluna;
```

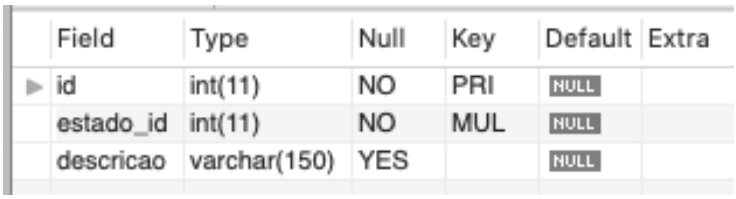
Onde:

alter table	Comando de alteração de tabela.
ntabela	Nome da tabela (existente) cuja coluna será eliminada.
drop column	Comando de exclusão de coluna.
ncoluna	Nome da coluna que será excluída.
;	Ponto e vírgula, determina o final do comando.

Exemplo: **`alter table cidade drop codigoArea;`**

Dicionário de dados resultante da tabela Cidade:

Figura 18 – Excluindo uma coluna na tabela Cidade (ferramenta MySQL Workbench)



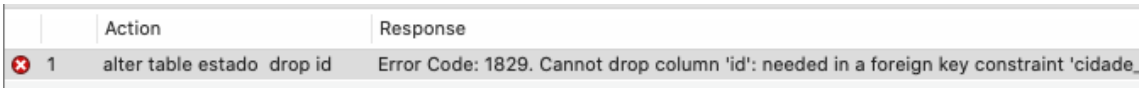
Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	
estado_id	int(11)	NO	MUL	NULL	
descricao	varchar(150)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

É importante ressaltar que não é possível a exclusão de uma coluna que é chave primária e que está sendo referenciada como chave estrangeira em outras tabelas. Para melhor entendimento, vamos exemplificar esse caso realizando uma tentativa de exclusão na tabela Estado, coluna ‘id’.

Exemplo: **`alter table estado drop id;`**

Figura 19 – Erro de exclusão de chave primaria (ferramenta MySQL Workbench)



	Action	Response
1	alter table estado drop id	Error Code: 1829. Cannot drop column 'id': needed in a foreign key constraint 'cidade_

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Adicionando uma chave estrangeira em uma tabela:

Sintaxe:
```
alter table ntabela add constraint nRestricao
      foreign key ncoluna
      references ntabela(ncoluna);
```

Onde:

alter table	Comando de alteração de tabela.
ntabela	Nome da tabela (existente) cuja coluna terá uma restrição.
add constraint	Comando de inclusão de restrição na coluna.

nRestricao	Nome da restrição que será incluída.
foreign key	Restrição de chave estrangeira.
ncoluna	Nome da coluna que é a chave estrangeira.
references	Cláusula do comando.
ntabela	Nome da tabela que contém a chave primária.
ncoluna	Nome da coluna que é chave primária na ntabela.
;	Ponto e vírgula, determina o final do comando.

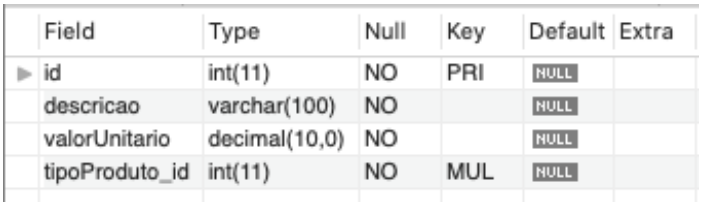
Exemplo: Criar um relacionamento entre as tabelas Produto e TipoProduto, sabendo que na tabela Produto não existe a coluna que representará a ligação entre as duas tabelas. Dessa forma, antes de criarmos a chave estrangeira, devemos adicionar a coluna 'tipoProduto_id' na tabela Produto.

```
alter table produto
   add tipoProduto_id integer not null;
```

Agora que a coluna foi adicionada na tabela Produto, vamos adicionar a restrição de chave estrangeira.

```
alter table produto add constraint fk_prodTipo
   foreign key (tipoProduto_id)
   references tipoProduto(id);
```

Figura 20 – Adicionando uma chave estrangeira na tabela Produto (ferramenta MySQL Workbench)



	Field	Type	Null	Key	Default	Extra
▶	id	int(11)	NO	PRI	NULL	
	descricao	varchar(100)	NO		NULL	
	valorUnitario	decimal(10,0)	NO		NULL	
	tipoProduto_id	int(11)	NO	MUL	NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Renomeando uma tabela:

Sintaxe:

```
alter table tabelaantiga
      rename to tabelanova;
```

Onde:

alter table	Comando de alteração de tabela.
tabelaantiga	Nome da tabela (existente) que será renomeada.
rename to	Comando de alteração.

tabelanova	Novo nome da tabela.
;	Ponto e vírgula, determina o final do comando.

Exemplo:

```
alter table cidade rename to cidades;
```

5.2 Propriedades Especiais

No caso dos tipos de dados numéricos, podemos inserir algumas propriedades para melhorar a integridade dos dados. Destacam-se: *unsigned*, *zerofill* e *auto_increment*, sendo:

1. *Unsigned* – Essa propriedade não permite o recebimento de valores negativos (inferior a zero) para colunas do tipo de dado inteiro. Uma coluna que não possua essa propriedade declarada assumirá por padrão ***signed***, permitindo a atribuição de valores negativos.
2. *Zerofill* – Insere o autopreenchimento com zeros nos espaços ociosos, ou seja, que não estão sendo utilizados. É importante ressaltar que a declaração *zerofill* em uma coluna, automaticamente o torna ***unsigned*** (apenas valores positivos).
3. *Auto_increment* – Propriedade bastante utilizada em chaves primárias, cuja característica é gerar automaticamente um número inteiro e incrementá-lo a partir da inclusão de uma nova linha, não havendo a inclusão manual desse valor na coluna. A coluna que recebe essa propriedade, necessariamente, deverá ser declarada com o tipo de dado inteiro (*integer*), possuir a restrição *not null* e a definição de *primary key* ou *unique*. Na tabela, inclusive, só poderá haver uma coluna utilizando essa propriedade. Vale salientar que uma coluna com *auto_increment* é sempre ***unsigned*** por padrão.

   Exemplo de aplicação do *unsigned* e *zerofill*:

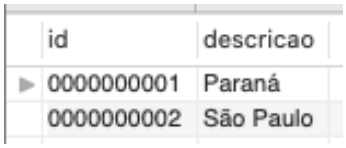
```
create table estado (
     id integer unsigned zerofill,
     descricao varchar(100));

insert into estado values (1, 'Paraná');
insert into estado values (2, 'São Paulo');
```

```
select * from estado;
```

Perceba que, ao executar o *select* na tabela Estado, a coluna ‘id’ ficou preenchida automaticamente com zeros à esquerda dos números 1 e 2. Essa é a função da propriedade *zerofill*.

Figura 21 – Exemplo da propriedade *zerofill* (ferramenta MySQL Workbench)



id	descricao
0000000001	Paraná
0000000002	São Paulo

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

E se tentarmos inserir uma linha com valor negativo na coluna ‘id’?

```
insert into estado values (-1, 'Exemplo Unsigned');
```

Figura 22 – Erro no uso da propriedade *unsigned* (ferramenta MySQL Workbench)

	Action	Response
1	insert into estado...	Error Code: 1264. Out of range value for column 'id' at row 1

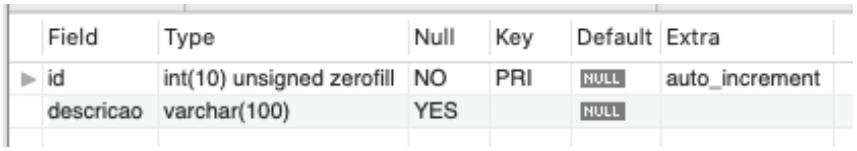
Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Exemplo de *auto-increment*: vamos criar a tabela Estado com a coluna ‘id’ com a propriedade *auto-increment*, tornando a coluna dinâmica, ou seja, sempre que uma nova linha for adicionada o valor de ‘id’ será incrementado em 1 sequencialmente.

```
create table estado (
     id integer unsigned zerofill
     primary key auto_increment,
     descricao varchar(100));
```

Apresentando o dicionário de dados da tabela Estado, verifique que a coluna ‘id’ está declarada como *auto_increment*.

Figura 23 – Exemplo da propriedade *auto_increment* (ferramenta MySQL Workbench)



Field	Type	Null	Key	Default	Extra
id	int(10) unsigned zerofill	NO	PRI	NULL	auto_increment
descricao	varchar(100)	YES		NULL	

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

Para testar o *auto-increment*, vamos inserir dados na tabela. Por exemplo:

```
insert into estado (descricao) values ('Paraná');
insert into estado (descricao) values ('São Paulo');
select * from estado;
```

Perceba que ao inserir os estados na tabela não foi necessário informar o 'id' de cada estado. Esse valor será inserido automaticamente pela propriedade *auto_increment*.

Figura 24 – Exemplo da propriedade *auto_increment* (ferramenta MySQL Workbench)

id	descricao
0000000001	Paraná
0000000002	São Paulo
NULL	NULL

Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

5.3 Detalhando uma *Query* ou *script*

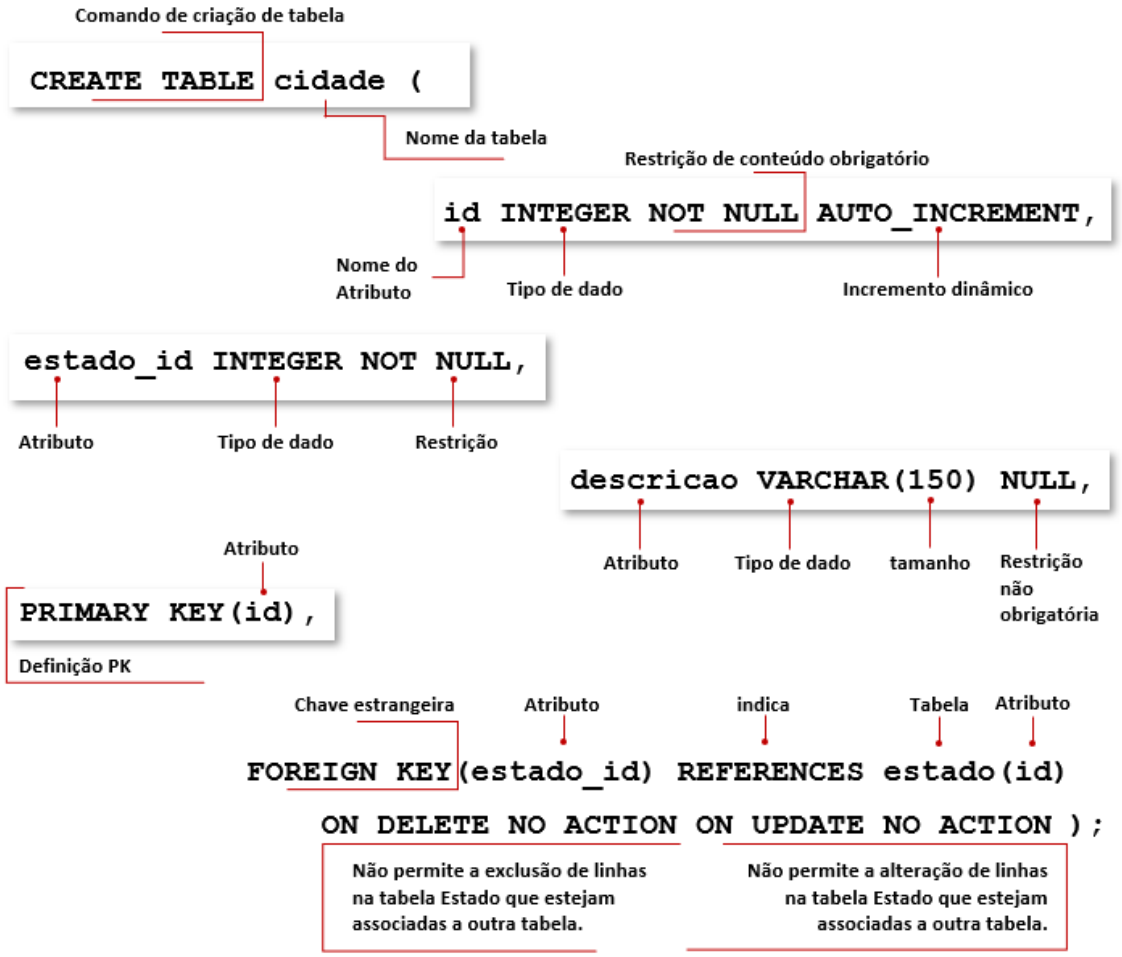
Ao longo desta etapa, você teve acesso a um grande volume de comandos que devem ser assimilados. Dessa forma, para ajudá-lo a compreender melhor os componentes de uma *query*, elegemos o comando *create table* e iremos detalhá-lo para você.

Query ou *script*:

```
CREATE TABLE cidade (
  id INTEGER NOT NULL AUTO_INCREMENT,
  estado_id INTEGER NOT NULL,
  descricao VARCHAR(150) NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(estado_id) REFERENCES estado(id)
```

```
ON DELETE NO ACTION ON UPDATE NO ACTION);
```

Figura 25 – Detalhando uma *query* ou *script*



Fonte: Elaborado por Ricardo e Silvie Albano, 2022.

FINALIZANDO

Nesta etapa, conhecemos os tipos de dados e suas características, comentamos também sobre a influência que eles exercem sobre o crescimento de um Banco de Dados.

Ingressamos no mundo do SQL, onde, através da teoria aliada a exemplos práticos, começamos a dar os nossos primeiros passos na linguagem. Criando um Banco de Dados, declarando as tabelas necessárias e realizando a manutenção dos dados existentes nas mesmas, executando comandos de inclusão, alteração, exclusão e consulta aos dados. Também aprendemos a realizar modificações na estrutura das tabelas, alterando as características de cada coluna, inserindo novas colunas e até mesmo excluindo colunas desnecessárias.

Abordamos com se dá a definição de chaves nas tabelas e de que forma isso ocorre nas relações entre as tabelas, discutindo sobre a importância do uso de restrições que têm por objetivo manter a integridade dos dados.

Recomendamos que você dedique um tempo de qualidade para assimilar os conhecimentos apresentados, revisando todos os conceitos cuidadosamente e colocando-os em prática, criando suas próprias *queries*.

Lembre-se, a palavra de ordem é praticar!

REFERÊNCIAS

BEIGHLEY, L. **Use a Cabeça SQL**. Rio de Janeiro: Alta Books, 2010.

HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Bookman, 2009.

MYSQL DOCUMENTATION. Disponível em: <https://dev.mysql.com/doc>. Acesso em: 5 jan. 2023.

SETZER, V. W. **Banco de Dados:** Conceitos, modelos, gerenciadores, projeto lógico, projeto físico. 3. ed. São Paulo: Edgard Blücher, 1986.