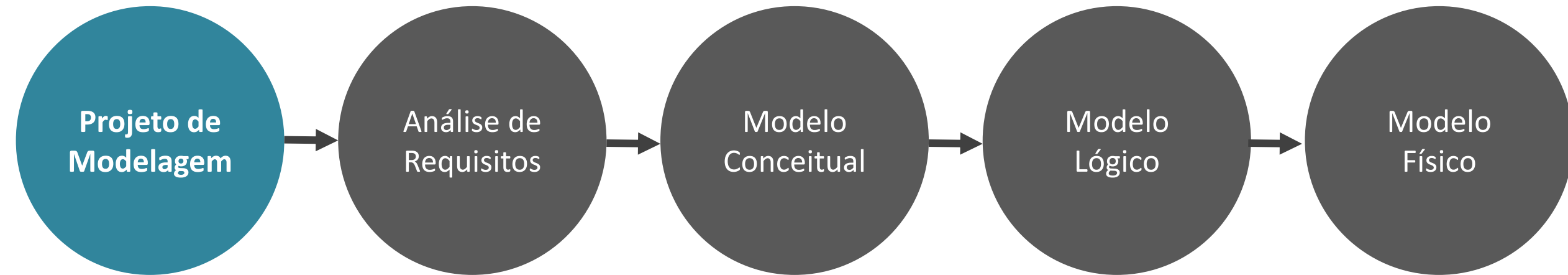


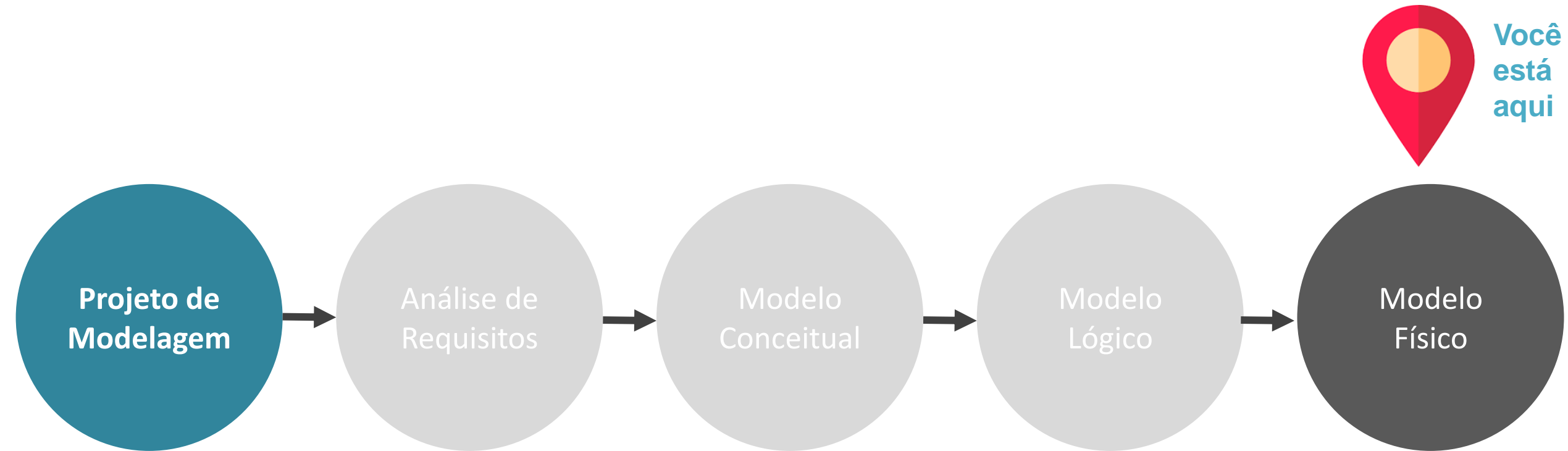
PROJETO DE BANCO DE DADOS

Modelo Físico e Projeto Prático

Quais são as etapas de um Projeto de Modelagem?



Quais são as etapas de um Projeto de Modelagem?



Nesta aula vamos falar sobre a terceira delas, o **Modelo Físico**!

Introdução

Uma vez que o **analista** observa o mundo real, ele faz uma representação do observado através dos três modelos: **conceitual, lógico e físico**.

Relembrando, o **modelo conceitual** é aquele onde vamos definir as entidades e os relacionamentos, por meio de diagramas.

Já o **modelo lógico** é o momento em que definiremos como serão as tabelas.

Por fim, o **modelo físico** é a etapa de criação das tabelas dentro do software de banco de dados.

Linguagem SQL e Sistemas de Bancos de Dados (SGBD)

Será no Modelo Físico que vamos implementar o projeto de modelagem através da linguagem SQL.

Sabemos que para utilizar a linguagem, precisamos escolher um programa de banco de dados, mais conhecido como Sistema de Gerenciamento de Banco de Dados (SGBD).



Implementando o Banco de Dados

Um **modelo de dados físico** descreve a implementação específica do banco de dados, com base no modelo lógico. Ele oferece uma abstração do banco de dados e ajuda a gerar um esquema com detalhes mais ricos, tais como o tipo dos atributos e os nomes das tabelas e dos campos das tabelas.

TIPOS DE DADOS



Para criarmos o modelo físico do nosso banco de dados, precisamos entender os **tipos de dados**. Cada atributo de uma entidade é responsável por guardar uma **informação** de um determinado registro, que pode ser uma data, um valor, uma observação, um código, etc. Por isso, no diagrama anterior, cada atributo tem um tipo de dado, ou seja, uma indicação do tipo de informação que será registrada. Os tipos de dados mais comuns são os seguintes:

Tipos numéricos.

- **Int:** Abrange o armazenamento de números inteiros. É comumente utilizado no campo de código das entidades, por se tratar de códigos únicos que identificam o registro da entidade.
- **Decimal ou Numeric:** Representa o armazenamento de números decimais, com precisão fixa, especificada através da notação decimal(p, e), em que p representa a precisão, enquanto e representa a escala. Por exemplo, decimal(3, 2) abrange os números com 3 dígitos antes da vírgula e 2 dígitos após a vírgula.
- **Float ou Double:** São utilizados para valores numéricos. Porém, a notação dos valores é diferente, pois são armazenados dados com até 17 dígitos no total. As notações para utilização desses tipos são float(t, d) e double(t, d), em que t representa a quantidade total de dígitos e d representa a quantidade de casas decimais.

Tipos de data e hora

- **Date:** Armazena informações de data, tais como ano, mês e dia.
- **Time:** Armazena informações de horário, tais como hora, minuto e segundo.
- **Datetime:** Armazena uma combinação de data e hora, no formato YYYY-MM-DD hh:mm:ss

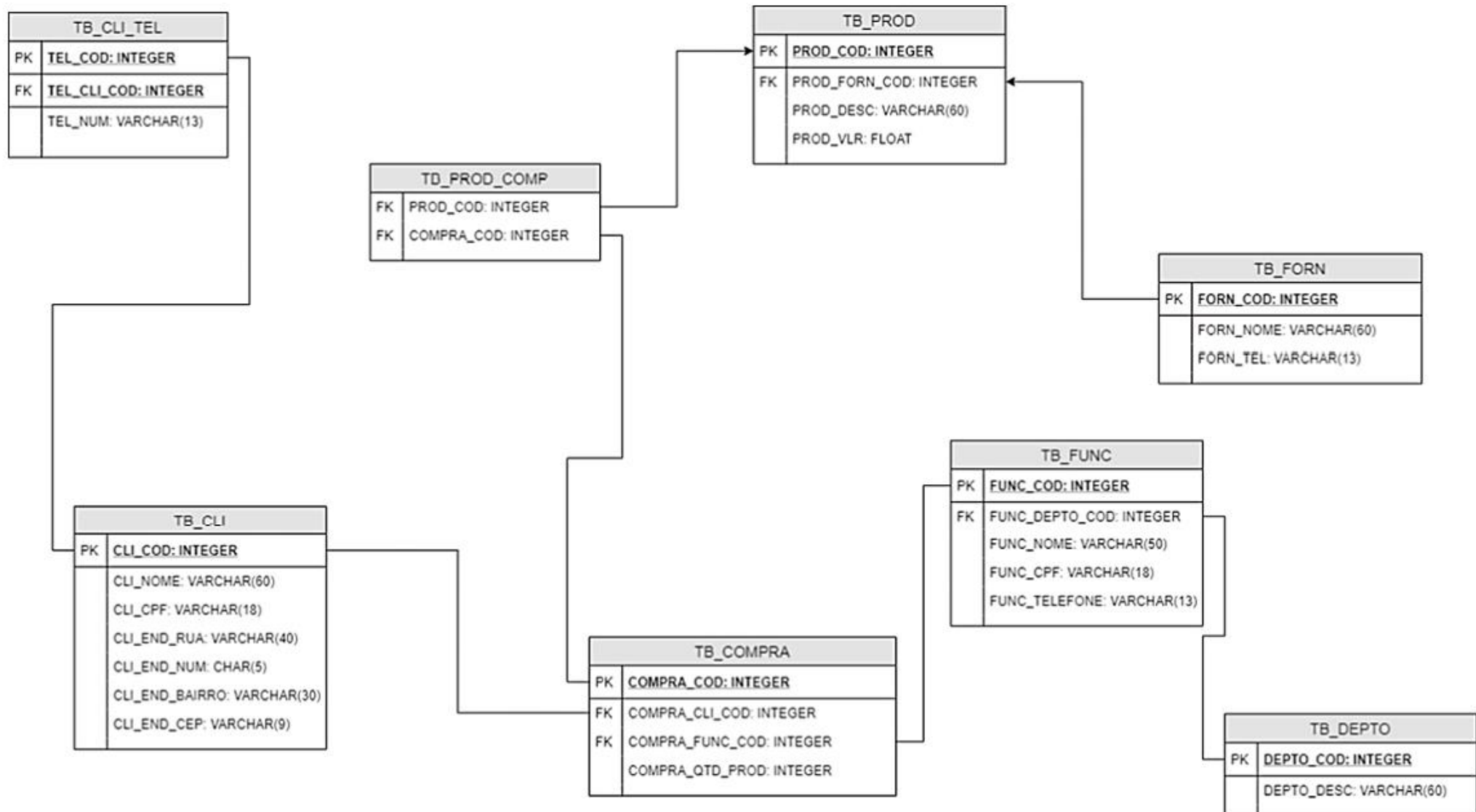
Tipos com caracteres

- **Char:** Armazena uma quantidade fixa de caracteres diversos, tais como letras, números e caracteres especiais, limitada a até 255 caracteres. A notação é `char(n)`, sendo `n` a quantidade de caracteres.
- **Varchar:** Assim como Char, este tipo de dado também armazena uma quantidade fixa de caracteres diversos, tais como letras, números e caracteres especiais, limitada a até 65.535 caracteres. A notação é `varchar(n)`, sendo `n` a quantidade de caracteres.

Diagrama do Modelo Físico

- Agora que você já conhece os tipos de dados, crie um novo **diagrama** no draw.io e implemente o **modelo físico** do nosso banco de dados, com base nas informações do modelo lógico da aula anterior. Seu diagrama deve ficar semelhante ao seguinte:

Diagrama do Modelo Físico



IMPLEMENTAÇÃO DO SGBD

Agora que já criamos o **modelo físico**, é preciso que o SGBD seja definido, para que a implementação do banco de dados seja feita através da linguagem de consulta estruturada, conhecida também como SQL (**do inglês, Structured Query Language**). O SGBD utilizado para a implementação do nosso estudo de caso será o **MySQL**. Assim, é necessário que o MySQL já esteja devidamente instalado em seu computador.

IMPLEMENTAÇÃO DO SGBD



Criando um novo banco de dados no modo interativo

O MySQL permite que o banco de dados seja criado e manuseado de duas formas:

IMPLEMENTAÇÃO DO SGBD

- **Interativa:** Feita através do MySQL Workbench, que faz parte da instalação realizada inicialmente. Essa ferramenta oferece diversas funcionalidades, desde a modelagem do banco de dados até sua criação e manutenção.
- **Não interativa:** Aqui, o banco de dados deve ser acessado via linha de comando do próprio MySQL. Trata-se de uma instância do banco de dados instalado na máquina que costuma ser encontrada na pasta *bin* da instalação ou no menu iniciar.

Structured Query Language (SQL)

Essa linguagem pode ser dividida em agrupamentos de comandos que permitem **criar** ou **alterar** a estrutura do banco de dados, ou ainda **inserir**, **alterar**, **deletar** e **recuperar dados** do próprio banco de dados. Esses agrupamentos se chamam **DDL**, **DML** e **DCL** (respectivamente, Data Definition Language, Data Manipulation Language e Data Control Language).

DDL SQL (Criação e alteração da estrutura)

DDL é a sigla de **Data Definition Language** (em português, linguagem de definição de dados) e designa os comandos utilizados para criar e modificar a estrutura do banco de dados ou de seus objetos, tais como tabelas e chaves. Confira a seguir os principais comandos desse grupo.

DDL SQL (Criação e alteração da estrutura)

DDL é a sigla de **Data Definition Language** (em português, linguagem de definição de dados) e designa os comandos utilizados para criar e modificar a estrutura do banco de dados ou de seus objetos, tais como tabelas e chaves. Confira a seguir os principais comandos desse grupo.

DDL SQL (Criação e alteração da estrutura)

Comando CREATE

Este comando é utilizado para criar desde o banco de dados até os objetos que compõem sua estrutura, como tabelas, procedimentos, campos, etc.

Comando DROP

Deleta objetos da estrutura do banco de dados.

DDL SQL (Criação e alteração da estrutura)

Comando ALTER

Altera campos, chaves, tipos, etc. de itens existentes no banco de dados. Para adicionar um novo campo, a sintaxe é a seguinte:

DDL SQL (Criação e alteração da estrutura)

Comando TRUNCATE

Remove todos os dados e o espaço alocado em uma tabela.

```
TRUNCATE `nome_do_banco_de_dados`.`tb_nome_tabela`;
```

DDL SQL (Criação e alteração da estrutura)

Comando RENAME

Renomeia um objeto existente no banco de dados.

Criação do banco e das tabelas por meio de um script

Tendo entendido como usar os comandos **DDL**, agora podemos implementar um **script** que cria o banco de dados e todas as tabelas do banco de dados. Siga os passos abaixo:

1. Dê um **start** no servidor **Apache** e no SGBD **MySQL**.
2. Abrir o **MySQL Workbench** e criar uma conexão com o servidor.
3. Configurar a janela do Workbench e codificar os scripts

Criando o Banco de Dados estudocaso_vendas

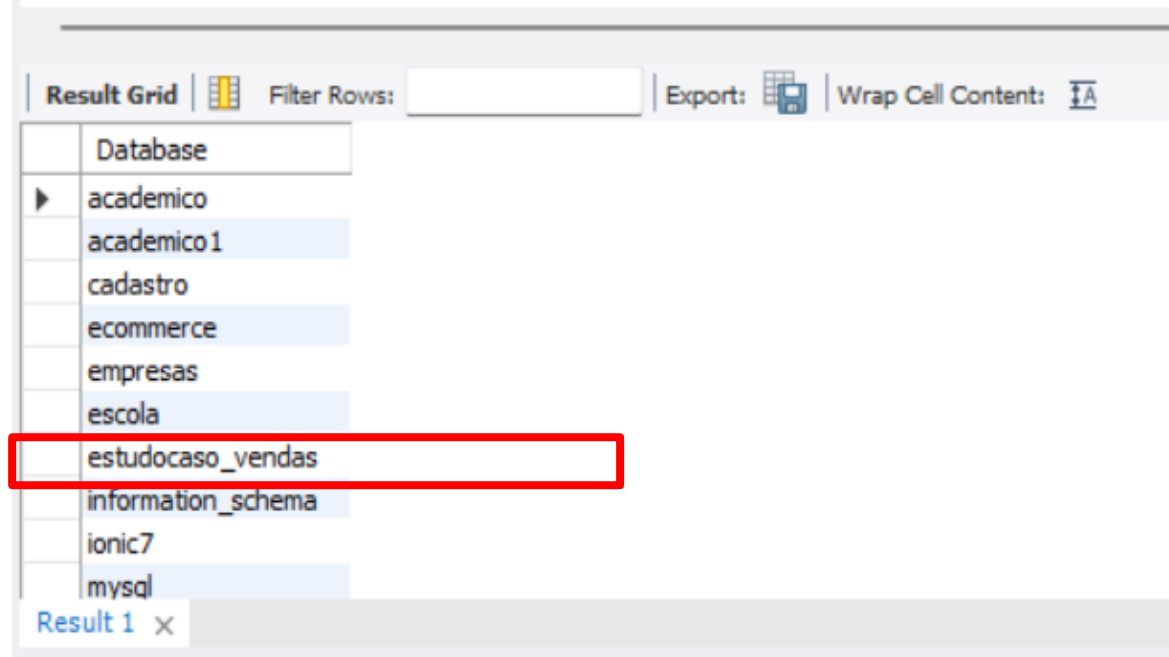
```
1  # Criando o Banco de Dados
2 • create database estudocaso_vendas
3   default character set utf8
4   default collate utf8_general_ci;
5
```


Criando o Banco de Dados estudocaso_vendas

Mostrando o Banco de Dados Criado

6

7 • `show databases;`



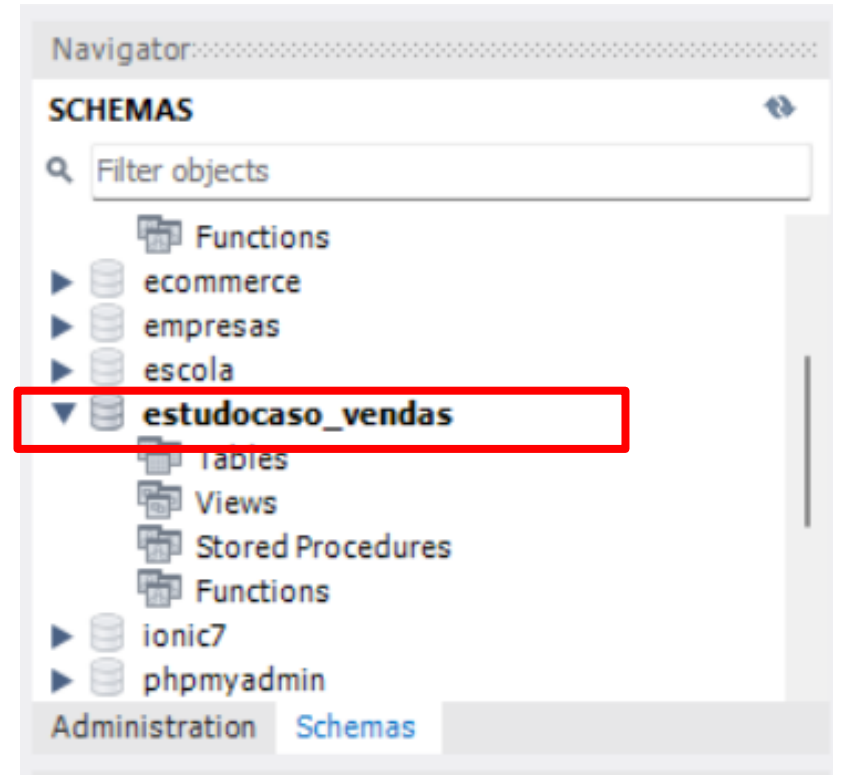
The screenshot shows a database management interface with a toolbar at the top containing 'Result Grid', 'Filter Rows', 'Export', and 'Wrap Cell Content'. Below the toolbar is a table of databases. The database 'estudocaso_vendas' is highlighted with a red rectangular box. At the bottom left, there is a tab labeled 'Result 1' with a close button 'x'.

Database
academico
academico1
cadastro
ecommerce
empresas
escola
estudocaso_vendas
information_schema
ionic7
mysql

Criando o Banco de Dados estudocaso_vendas

Colocando o Banco de Dados em Uso

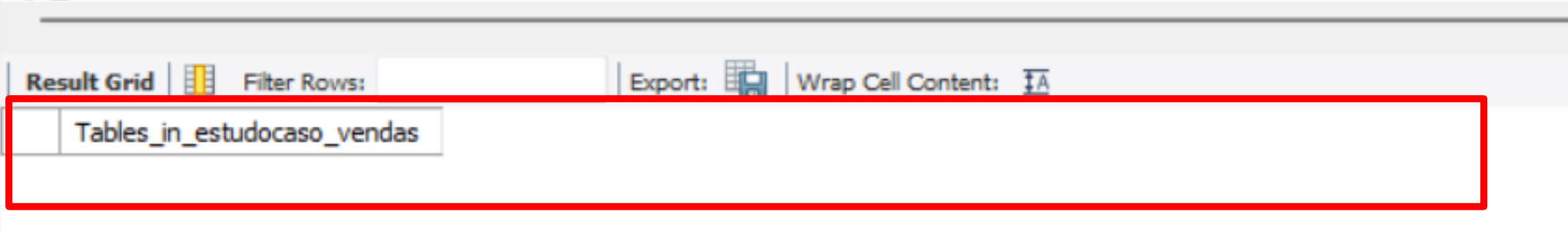
```
# Colocando o Banco de Dados em uso  
use estudocaso_vendas;
```



Criando o Banco de Dados estudocaso_vendas

Verificando se existe tabelas no Banco de Dados.

```
12  #Verificando se tem tabela no banco de dados
13 •  show tables;
14
--
```



Criando a tabela *Cliente*

```
16  # Criando a tabela cliente
17 • create table if not exists cliente (
18     codigo int not null primary key,
19     nome varchar(100) not null,
20     cpf varchar(11) not null unique,
21     end_rua varchar(60),
22     end_num char(5),
23     end_bairro varchar(40),
24     end_cep varchar(9)
25 ) default charset=utf8;
26
```

Criando a tabela *Cliente*

Verificando a estrutura da tabela cliente. (describe ou desc)

26

27 • desc cliente;

28

Result Grid						
		Filter Rows:			Export:	Wrap Cel
	Field	Type	Null	Key	Default	Extra
►	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

*Excluindo a tabela **Cliente***

Excluindo uma Tabela

```
drop table if exists cliente;
```

Excluindo o Banco de Dados estudocaso_vendas

Excluindo o Banco de Dados

```
drop database estudocaso_vendas;
```

Criando o Banco de Dados estudocaso_vendas

Criando o Banco de Dados novamente

```
1  # Criando o Banco de Dados
2 • create database estudocaso_vendas
3   default character set utf8
4   default collate utf8_general_ci;
5
```


*Criando a tabela **Cliente** novamente*

Criando a tabela cliente

```
⊖ create table if not exists cliente (  
  codigo int not null primary key,  
  nome varchar(100) not null,  
  cpf varchar(11) not null unique,  
  end_rua varchar(60),  
  end_num varchar(5),  
  end_bairro varchar(40),  
  end_cep varchar(9)  
)default charset=utf8;
```

Criando a tabela *Fornecedor*

```
# Criando a tabela fornecedor
• create table if not exists fornecedor (
  codigo int not null primary key,
  nome varchar(100) not null,
  telefone varchar(13)
)default charset=utf8;
```

Criando a tabela Departamento

Criando a tabela departamento

```
create table if not exists departamento (  
    codigo int not null primary key,  
    descricao varchar(100) not null  
); default charset=utf8;
```

Criando a tabela *Produto*

Criando a tabela produto

```
create table if not exists produto (  
    codigo int not null primary key,  
    cod_forn int not null,  
    descricao varchar(100) not null,  
    valor decimal(5,2),  
    foreign key(cod_forn) references fornecedor(codigo)  
); default charset=utf8;
```

Criando a tabela *Funcionario*

```
# Criando a tabela funcionario
```

- ```
create table if not exists funcionario (
 codigo int not null primary key,
 cod_depto int not null,
 nome varchar(100) not null,
 cpf varchar(11) not null unique,
 foreign key(cod_depto) references departamento(codigo)
)default charset=utf8;
```

# Criando a tabela Compra

```
Criando a tabela compra
```

- ```
create table if not exists compra (  
  codigo int not null primary key,  
  cod_cli int not null,  
  cod_func int not null,  
  qtd_prod int not null,  
  foreign key(cod_cli) references cliente(codigo),  
  foreign key(cod_func) references funcionario(codigo)  
); default charset=utf8;
```

Criando a tabela *Prod_Compra*

Criando a tabela prod_compra

- `create table if not exists prod_compra (
 cod_prod int not null,
 cod_compra int not null,
 foreign key(cod_prod) references produto(codigo),
 foreign key(cod_compra) references compra(codigo)
)default charset=utf8;`

*Criando a tabela **Cliente_Telefone***

Criando a tabela cliente_telefone

```
create table if not exists cliente_telefone (  
  codigo int not null primary key,  
  cod_cli int not null,  
  numero varchar(13) not null,  
  foreign key(cod_cli) references cliente(codigo)  
)default charset=utf8;
```




Concluímos até o momento a criação do Banco de Dados e suas Tabelas.

O próximo passo será inserir dados nas tabelas.

DML SQL (Inserção e manuseio de dados)

A próxima parte importante do SQL é a **DML**, sigla que vem de **Data Manipulation Language** (ou linguagem de manipulação de dados, em português), que agrupa comandos utilizados para **manipular** e **inserir** dados no banco de dados. Os principais comandos de manipulação são:

DML SQL (Inserção e manuseio de dados)

Comando INSERT

Utilizado para inserir um ou mais dados nas tabelas do banco de dados, por meio da seguinte sintaxe:

```
INSERT INTO `nome_banco_dados`.`tabela` (campo1, campo2)  
VALUES ('valor1', 'valor2');
```

DML SQL (Inserção e manuseio de dados)

Comando SELECT

Utilizado para recuperar dados nas tabelas do banco de dados, por meio da seguinte sintaxe:

```
SELECT campo1, campo2  
FROM `nome_banco_dados`.`tabela`
```

DML SQL (Inserção e manuseio de dados)

Comando UPDATE

Utilizado para atualizar os dados na tabela, por meio da seguinte sintaxe:

```
UPDATE `nome_banco_dados`.`tabela`  
SET campo1 = "Novo Valor"
```

DML SQL (Inserção e manuseio de dados)

Comando DELETE

Usado para deletar dados da tabela, por meio da seguinte sintaxe:

```
DELETE FROM `nome_banco_dados`.`tabela`
```

POPULANDO TABELAS

Após aprendermos os comandos do grupo **DML**, podemos agora **inserir dados**, o que também é chamado de **popular tabelas**. A princípio, vamos inserir os dados nas tabelas de **clientes**, **departamentos** e **fornecedores**, que possuem **relacionamentos um para um**, já que não será possível atribuímos um dado relacionado a essas tabelas se elas ainda não tiverem dados

ADICIONANDO DADOS NA TABELA *CLIENTE*

#Inserindo dados na tabela CIENTE

```
INSERT INTO cliente(codigo, nome, cpf, end_rua, end_num, end_bairro, end_cep)
VALUES ('1', 'André', '11111111111', 'Rua A', '99', 'Bairro A', '69111111' );
```

```
select *from cliente;
```


ADICIONANDO DADOS NA TABELA *CLIENTE*

```
INSERT INTO cliente(codigo, nome, cpf, end_rua, end_num, end_bairro, end_cep)  
VALUES ('2', 'Bruna', '22222222222', 'Rua B', '199', 'Bairro B', '69222222' );
```

```
INSERT INTO cliente(codigo, nome, cpf, end_rua, end_num, end_bairro, end_cep)  
VALUES ('3', 'Caio', '33333333333', 'Rua C', '299', 'Bairro C', '69333333' );
```


ADICIONANDO DADOS NA TABELA FORNECEDOR

#Inserindo dados na tabela FORNECEDOR

```
INSERT INTO fornecedor(codigo, nome, telefone)  
VALUES ('1', 'Fornecedor Um', '92933442201');
```

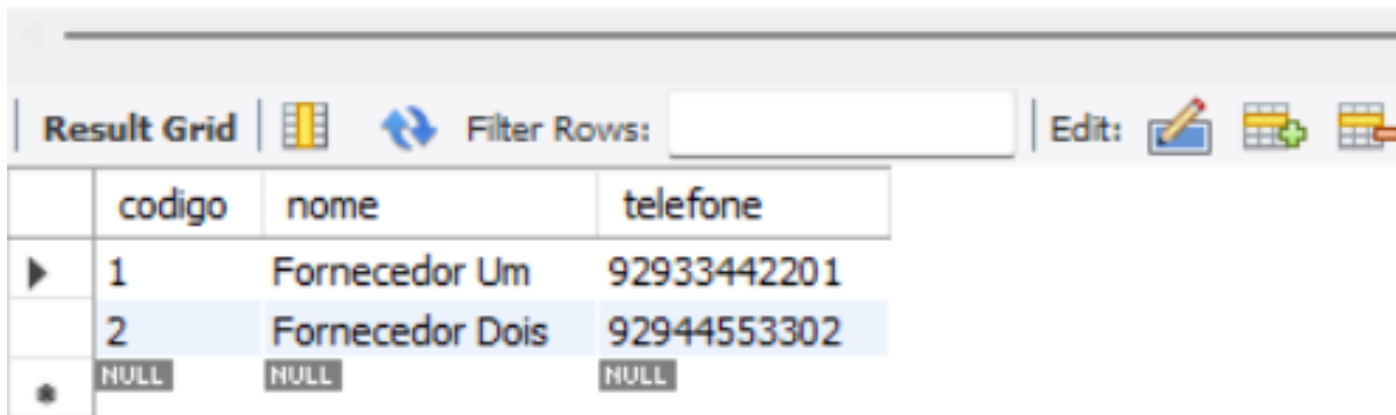
```
INSERT INTO fornecedor(codigo, nome, telefone)  
VALUES ('2', 'Fornecedor Dois', '92944553302');
```

ADICIONANDO DADOS NA TABELA FORNECEDOR

Verificando se os dados foram inseridos na tabela Fornecedor

23

24 • `select *from fornecedor;`



The screenshot shows a database application interface. At the top, there is a toolbar with icons for 'Result Grid', 'Filter Rows', and 'Edit'. Below the toolbar is a table with the following data:

	codigo	nome	telefone
▶	1	Fornecedor Um	92933442201
	2	Fornecedor Dois	92944553302
•	NULL	NULL	NULL

ADICIONANDO DADOS NA TABELA *DEPARTAMENTO*

#Inserindo dados na tabela DEPARTAMENTO

```
INSERT INTO departamento(codigo, descricao)  
VALUES ('1', 'Vendas');
```

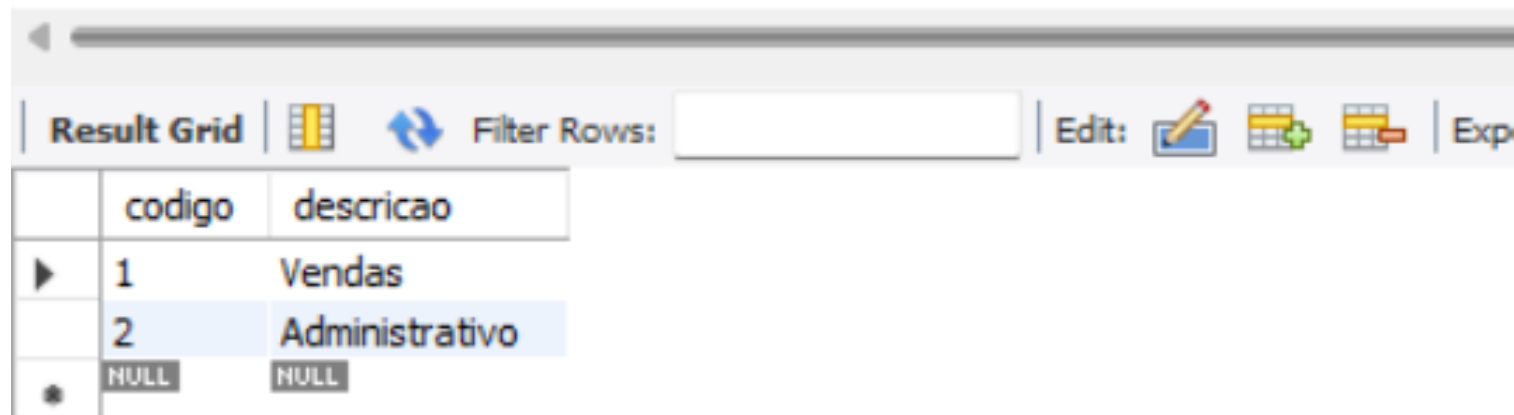
```
INSERT INTO departamento(codigo, descricao)  
VALUES ('2', 'Administrativo');
```

ADICIONANDO DADOS NA TABELA FORNECEDOR

Verificando se os dados foram inseridos na tabela Departamento

35

36 • `select *from departamento;`



The screenshot shows a database application window with a 'Result Grid' tab selected. The grid displays the contents of the 'departamento' table. The columns are 'codigo' and 'descricao'. There are three rows: a row with '1' and 'Vendas', a row with '2' and 'Administrativo' (which is highlighted), and a row with 'NULL' and 'NULL'. The interface includes a toolbar with icons for filtering, editing, and expanding/collapsing rows.

	codigo	descricao
▶	1	Vendas
	2	Administrativo
•	NULL	NULL

POPULANDO TABELAS

Ao inserir dados, temos **dois pontos de muita atenção**: primeiro, **não podemos repetir o mesmo número de chave primária** em uma tabela, ou seja, se um departamento de código 1 já foi inserido, o próximo deve ter o código 2 e assim por diante. Caso a mesma chave primária seja inserida mais de uma vez na mesma tabela, o SGBD retornará um erro. **Segundo, quando informamos que, por exemplo**, o campo **func_depto** é **1**, o departamento 1 deve estar inserido na tabela de departamentos. Caso esse departamento não exista na tabela, também obteremos um erro.

POPULANDO TABELAS

Agora, vamos popular as tabelas de **PRODUTO**, **FUNCIONARIO** e , que possuem **relacionamentos um para muitos**, lembrando que **não** será possível atribuímos um dado relacionado a essas tabelas se as tabelas de cardinalidade **um** ainda não tiverem dados.

ADICIONANDO DADOS NA TABELA *PRODUTO*

#Inserindo dados na tabela PRODUTO

```
INSERT INTO produto(codigo, cod_forn, descricao, valor)
VALUES ('1', '1', 'Produto um', '10.99');
```

```
INSERT INTO produto(codigo, cod_forn, descricao, valor)
VALUES ('2', '2', 'Produto dois', '20.99');
```

```
INSERT INTO produto(codigo, cod_forn, descricao, valor)
VALUES ('3', '1', 'Produto tres', '30.99');
```

ADICIONANDO DADOS NA TABELA FORNECEDOR

Verificando se os dados foram inseridos na tabela Produto

```
49 • select *from produto;
```

Result Grid

Filter Rows:

Edit:

	codigo	cod_forn	descricao	valor
▶	1	1	Produto um	10.99
	2	2	Produto dois	20.99
	3	1	Produto tres	30.99
⬇	NULL	NULL	NULL	NULL

ADICIONANDO DADOS NA TABELA *FUNCIONARIO*

#Inserindo dados na tabela FUNCIONARIO

```
INSERT INTO funcionario(codigo, cod_depto, nome, cpf)  
VALUES ('1', '1', 'Iolanda', '12345678900');
```

```
INSERT INTO funcionario(codigo, cod_depto, nome, cpf)  
VALUES ('2', '2', 'Martin', '33322244400');
```

ADICIONANDO DADOS NA TABELA *FUNCIONARIO*

Verificando se os dados foram inseridos na tabela *Funcionario*

59

```
60 • select *from funcionario;
```

Result Grid

Filter Rows:

Edit:

	codigo	cod_depto	nome	cpf
▶	1	1	Iolanda	12345678900
	2	2	Martin	33322244400
•	NULL	NULL	NULL	NULL

ADICIONANDO DADOS NA TABELA *CLIENTE_TELEFONE*

#Inserindo dados na tabela CLIENTE_TELEFONE

```
INSERT INTO cliente_telefone(codigo, cod_cli, numero)
VALUES ('1', '1', '9233334444');
```


POPULANDO TABELAS



Agora, vamos popular as tabelas de **COMPRA** e **PROD_COMPRA**, que possuem **relacionamentos muitos para muitos**.

ADICIONANDO DADOS NA TABELA COMPRA

#Inserindo dados na tabela COMPRA

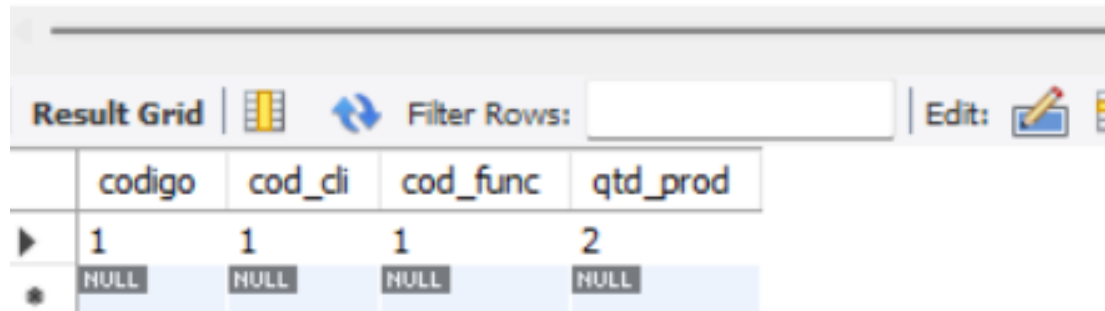
```
INSERT INTO compra(codigo, cod_cli, cod_func, qtd_prod)
VALUES ('1', '1', '1', '2');
```


ADICIONANDO DADOS NA TABELA COMPRA

Verificando se os dados foram inseridos na tabela Compra

```
67 • select *from compra;
```

```
68
```



The screenshot shows a database interface with a 'Result Grid' tab. The grid displays the results of a SQL query. The columns are labeled 'codigo', 'cod_cli', 'cod_func', and 'qtd_prod'. The first row contains the values '1', '1', '1', and '2'. The second row contains 'NULL' values for all four columns. The interface includes a 'Filter Rows' field and an 'Edit' button.

	codigo	cod_cli	cod_func	qtd_prod
▶	1	1	1	2
•	NULL	NULL	NULL	NULL

ADICIONANDO DADOS NA TABELA *PROD_COMPRA*

#Inserindo dados na tabela PROD_COMPRA

```
INSERT INTO prod_compra(cod_prod, cod_compra)  
VALUES ('1', '1');
```




```
INSERT INTO prod_compra(cod_prod, cod_compra)  
VALUES ('2', '1');
```

ADICIONANDO DADOS NA TABELA *PRO_COMPRA*

Verificando se os dados foram inseridos na tabela *Prod_Compra*

```
77 • select *from prod_compra;
```

```
78
```

Result Grid   Filter Rows: <input type="text"/> Export:  Wrap Cell		
	cod_prod	cod_compra
▶	1	1
	2	1

ALTERANDO A ESTRUTURA DE UMA TABELA

`alter table`

Em determinada situação é necessário alterar a estrutura de uma tabela.



Vamos começar visualizando a estrutura da tabela atual.

`desc cliente;`

Result Grid						
		Filter Rows:				
		Export:				
		Wrap				
	Field	Type	Null	Key	Default	Extra
►	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	end_ua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Criando um novo Campo no final da Tabela:



```
alter table cliente add column email varchar(80);  
  
desc clientes;
```

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap Cell						
	Field	Type	Null	Key	Default	Extra
▶	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	
	email	varchar(80)	YES		NULL	

Excluindo um Campo da Tabela:

`alter table cliente drop column email;`



`desc clientes;`

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap (
	Field	Type	Null	Key	Default	Extra
▶	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	end_ua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Criando um Campo em qualquer ponto da Tabela:

alter table cliente add column email varchar(60) after cpf;

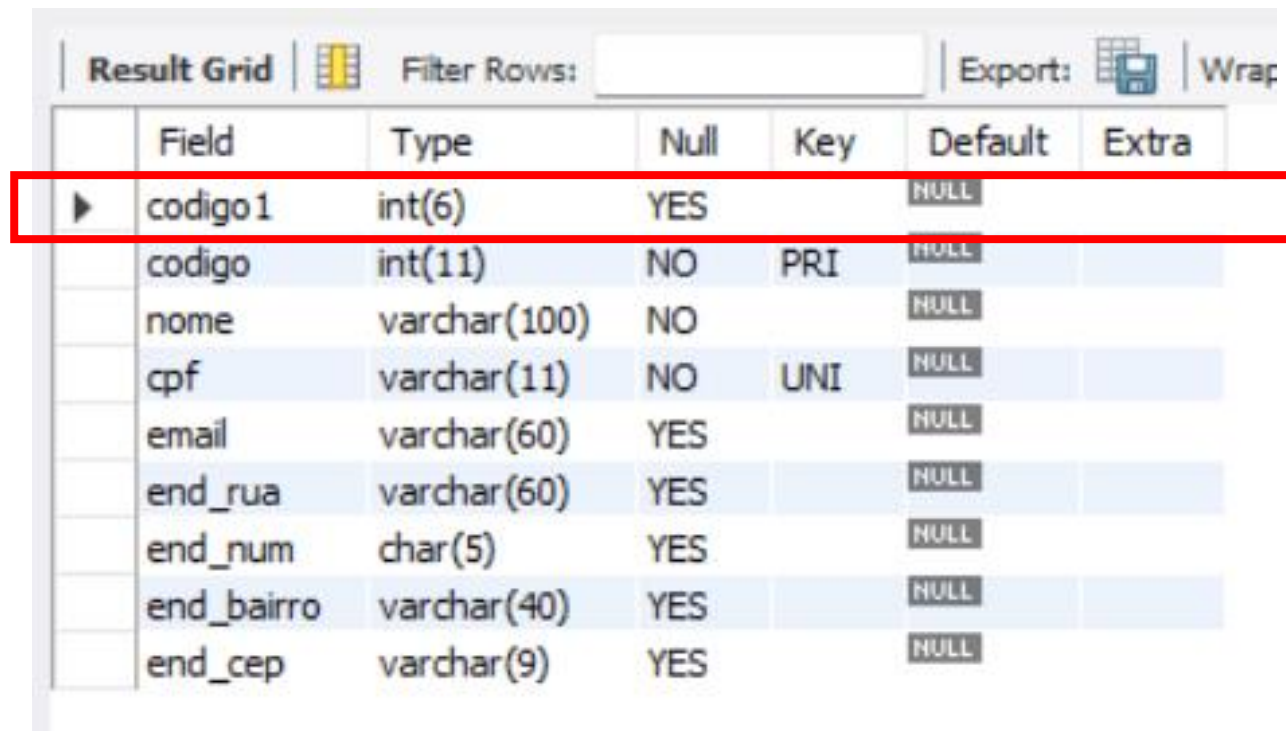
desc clientes;

Result Grid  Filter Rows: <input type="text"/> Export:  Wrap						
	Field	Type	Null	Key	Default	Extra
▶	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	email	varchar(60)	YES		NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Criando um Campo como primeiro da Tabela:

`alter table cliente add column codigo1 int(6) first;`

`desc cliente;`



	Field	Type	Null	Key	Default	Extra
▶	codigo1	int(6)	YES		NULL	
	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	email	varchar(60)	YES		NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Excluindo um Campo da Tabela:

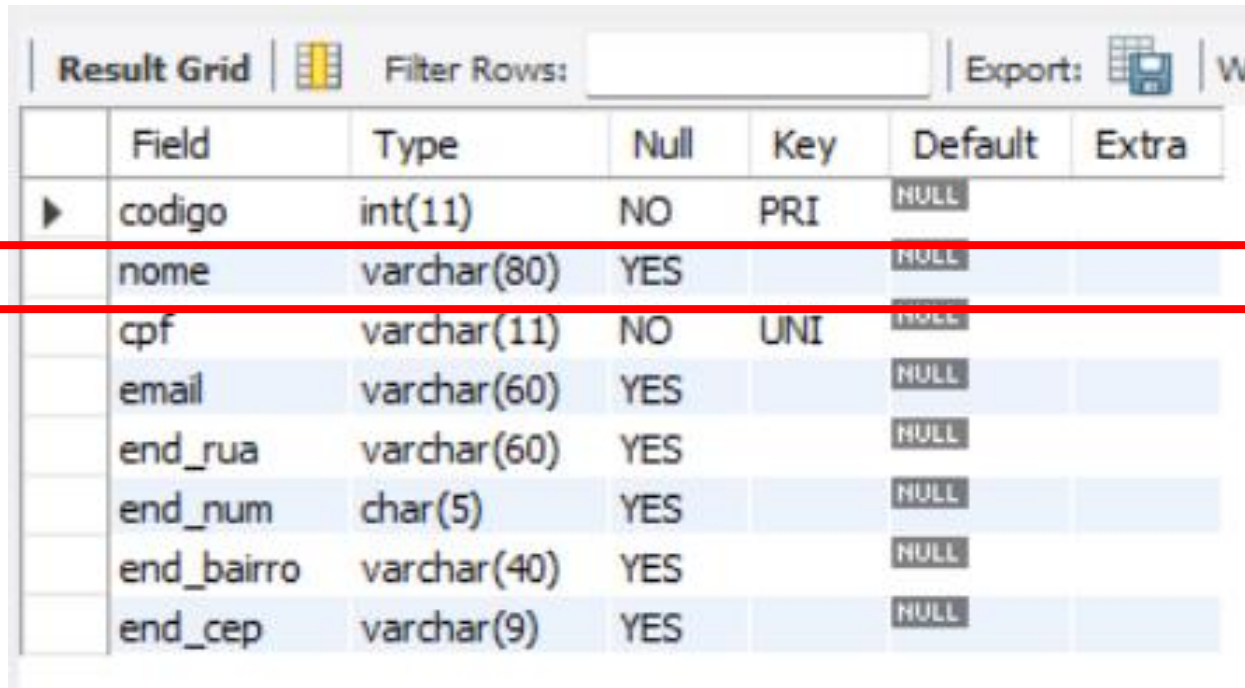
`alter table cliente drop column codigo1;`

`desc cliente;`

Result Grid Filter Rows: Export: Wrap						
	Field	Type	Null	Key	Default	Extra
▶	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(100)	NO		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Modificando Definições

`alter table cliente modify column nome varchar(80);`



The screenshot shows a 'Result Grid' window with a table structure. The table has columns: Field, Type, Null, Key, Default, and Extra. The row for the 'nome' field is highlighted with a red rectangle.

	Field	Type	Null	Key	Default	Extra
▶	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(80)	YES		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	email	varchar(60)	YES		NULL	
	end_ rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Com o **Modify**,
pode alterar os
tipos **primitivos** e
as **constraint**,
não dá para
renomear o
campo.

Renomeando o Nome da Coluna

```
alter table cliente change column nome empresa varchar(80);
```

+

```
5 • desc clientes;
```

5

Result Grid						Filter Rows:	Exports:
	Field	Type	Null	Key	Default		
▶	codigo	int(11)	NO	PRI	NULL		
	empresa	varchar(80)	YES		NULL		
	cpf	varchar(11)	NO	UNI	NULL		
	email	varchar(60)	YES		NULL		
	end_rua	varchar(60)	YES		NULL		
	end_num	char(5)	YES		NULL		
	end_bairro	varchar(40)	YES		NULL		
	end_cep	varchar(9)	YES		NULL		

Renomeando o Nome da Tabela

alter table cliente rename to alunos;
desc alunos;

Result Grid						
		Filter Rows:			Export:	W
	Field	Type	Null	Key	Default	Extra
►	codigo	int(11)	NO	PRI	NULL	
	nome	varchar(80)	YES		NULL	
	cpf	varchar(11)	NO	UNI	NULL	
	email	varchar(60)	YES		NULL	
	end_rua	varchar(60)	YES		NULL	
	end_num	char(5)	YES		NULL	
	end_bairro	varchar(40)	YES		NULL	
	end_cep	varchar(9)	YES		NULL	

Modificado Linhas ou Tuplas

- Comando **UPDATE** → O comando para atualizar registros.
- Comando **SET** → utilizada para alterar configurações da sessão atual.
- Cláusula **WHERE** → Cláusula não obrigatória que restringe os dados obtidos através de operações que testam se cada registro satisfaz a condição ou não;

Modificado Linhas ou Tuplas

1) Modificando apenas **1** campo da Linha

Ex_1 - Modificar o **nome** do cliente do registro 1.

```
update cliente set nome='Rafael' where codigo='1';
```

```
select *from cliente;
```

2 - Modificando **mais** de um campos da Linha

Ex_**2** - Modificar o **endereço** do cliente do registro 2

```
#Modificando mais de um campo da Linha
```

```
update cliente set end_rua='Rua das flores', end_num='155', end_bairro='Bairro D', end_cep='69444444' where codigo='2';
```


DELETE - Removendo uma Linha

1 – Apagando um registro:

Ex: 1 – **Apagar** o registro de **codigo 3**.

```
delete from cliente where codigo='3';
```

```
select *from cliente;
```

TRUNCATE - Removendo Todas Linhas

TRUNCAR → Significa apagar todas as linhas de uma tabela:

Ex: 1 – **Apagar** todos registro da Tabela cliente_telefone

```
truncate cliente;
```

```
truncate cliente_telefone;
```

Melhorando a organização e a performance

Agora que sabemos modelar um banco de dados, inserir dados nele e manuseá-los, é necessário entender alguns pontos importantes relacionados à organização e à performance da base de dados.

CONSULTAS COM RELACIONAMENTOS

Cláusula JOIN

A junção de tabelas com o comando *JOIN* permite vincular dados entre uma ou mais tabelas, com base nos valores das colunas em comum entre elas, ou seja, chaves primárias e estrangeiras. Para associar tabelas é preciso utilizar a cláusula *JOIN* no comando *SELECT*, após a cláusula *FROM*, conforme a sintaxe a seguir.

```
SELECT column_list  
FROM TABELA_PAI  
TIPO_JUNCAO JOIN TABELA_FILHO ON CONDICA0;
```

JOINS - Categorias

- **INNER JOIN**: Retorna linhas quando houver pelo menos uma correspondência em ambas as tabelas.
- **OUTER JOIN**: Retorna linhas mesmo quando não houver pelo menos uma correspondência em uma das tabelas (ou ambas). O OUTER JOIN divide-se em LEFT JOIN, RIGHT JOIN e FULL JOIN.

Exemplo 1: Listar os **clientes** e os seus **telefones** registrados:

```
select *from cliente
inner join cliente_telefone
on cliente.codigo = cliente_telefone.cod_cli;
```

Result Grid										
		Filter Rows:		Export:		Wrap Cell Content:				
	codigo	nome	cpf	end_rua	end_num	end_bairro	end_cep	codigo	cod_cli	numero
▶	1	Rafael	11111111111	Rua A	99	Bairro A	69111111	1	1	9233334444



Exemplo 2: Listar apenas o **nome** do cliente e o seu **telefone**:

```
select cliente.nome, cliente_telefone.numero from cliente  
inner join cliente_telefone  
on cliente.codigo = cliente_telefone.cod_cli;
```

Result Grid			Filter Rows:
	nome	numero	
▶	Rafael	9233334444	

Exemplo 3: Listar o **nome** do cliente, o nome do **funcionário** e o **qtd_prod** que foi comprado: (com três tabelas)

```
select cliente.nome, funcionario.nome, compra.qtd_prod from compra
inner join cliente
on compra.cod_cli = cliente.codigo
inner join funcionario
on compra.cod_cli=funcionario.codigo;
```

Result Grid			
  Filter Rows:			
	nome	nome	qtd_prod
▶	Rafael	Iolanda	2

Referência

INTRODUÇÃO A BANCO DE DADOS

Pereira, Paloma Cristina - Introdução a bancos de dados - Editora Senac São Paulo – São Paulo – 2021

E-book. Disponível em:

<https://www.bibliotecadigitalsenac.com.br/?from=busca%3FcontentInfo%3D2915%26term%3DLeite%2525252C%25252520Leonardo%25252520Alexandre%25252520Ferreira%25252520-%25252520Programa%252525C3%252525A7%252525C3%252525A3o%25252520de%25252520banco%25252520de%25252520dados#/legacy/epub/2915>

Acesso em 29/10/2023