

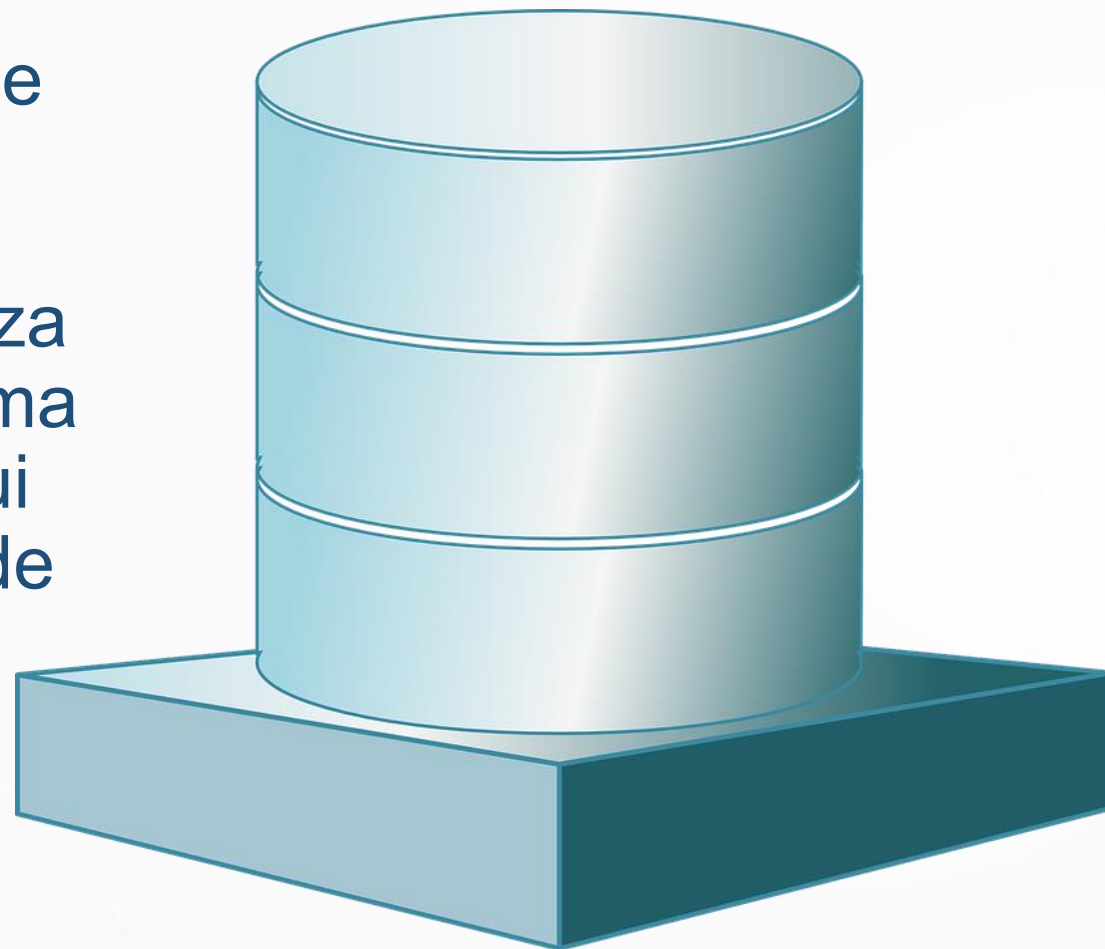


# UC8

## Banco de Dados para Internet

# O que é um banco de dados?

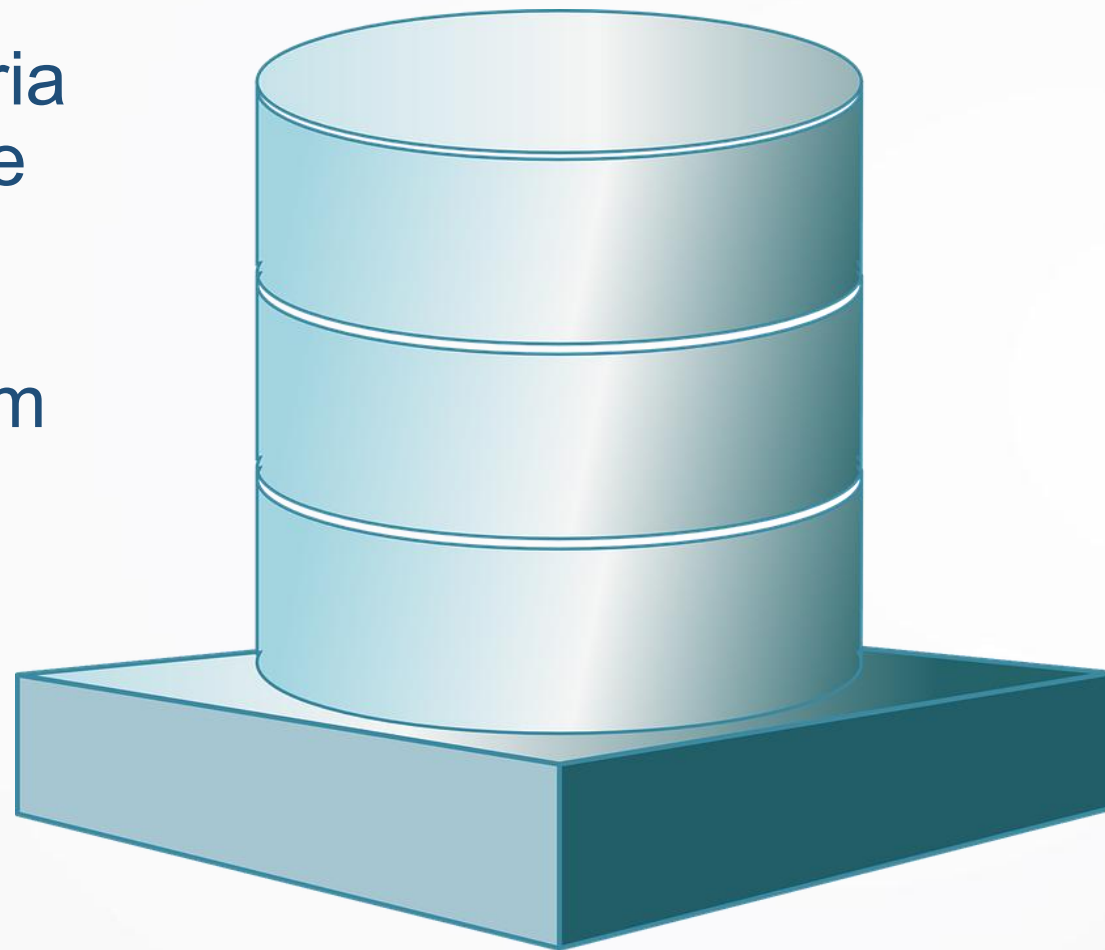
Os bancos de dados são repositórios de dados essenciais para todas as aplicações de sistemas de informação. Por exemplo, sempre que alguém realiza uma pesquisa na Web ou em um sistema local, faz login em uma conta ou conclui uma transação, um sistema de banco de dados armazena as informações para que elas possam ser acessadas posteriormente.





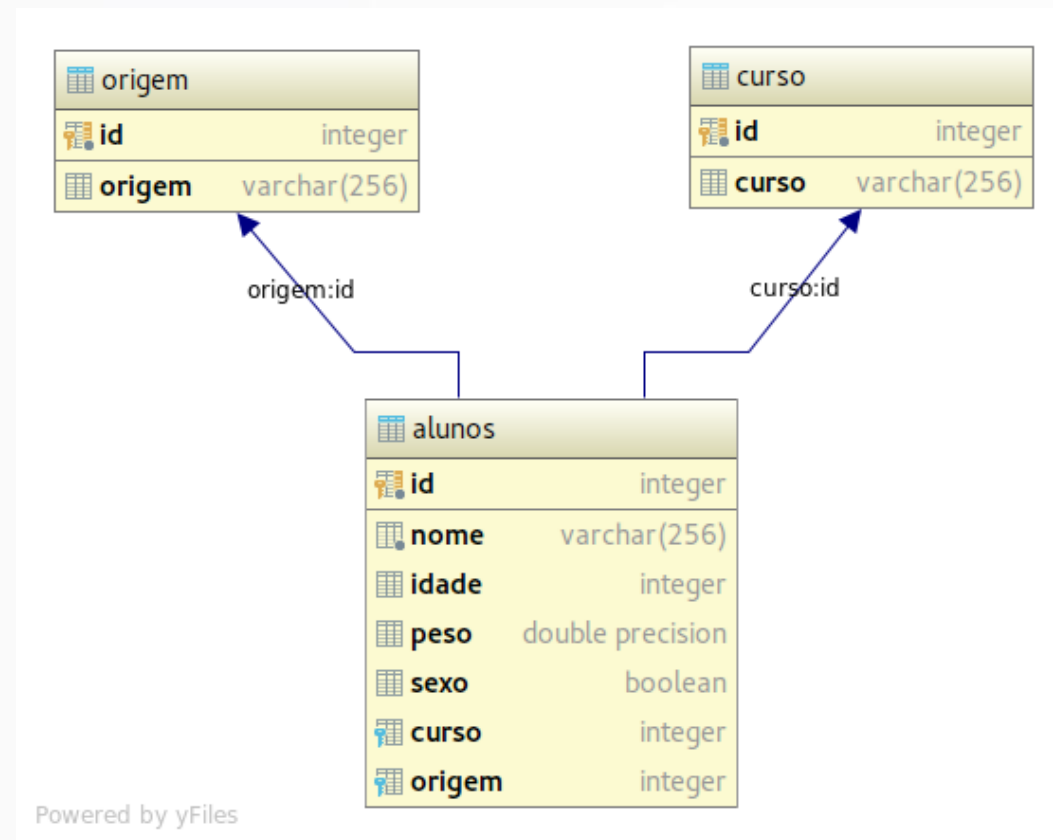
# O que é um banco de dados?

Além disso, o conceito de chave primária é fundamental para garantir a unicidade dos registros dentro de uma tabela. Relacionamentos entre tabelas e normalização são aspectos que também devem ser considerados, visando à integridade e eficiência no armazenamento de dados.



# O que é um Banco de Dados Relacional

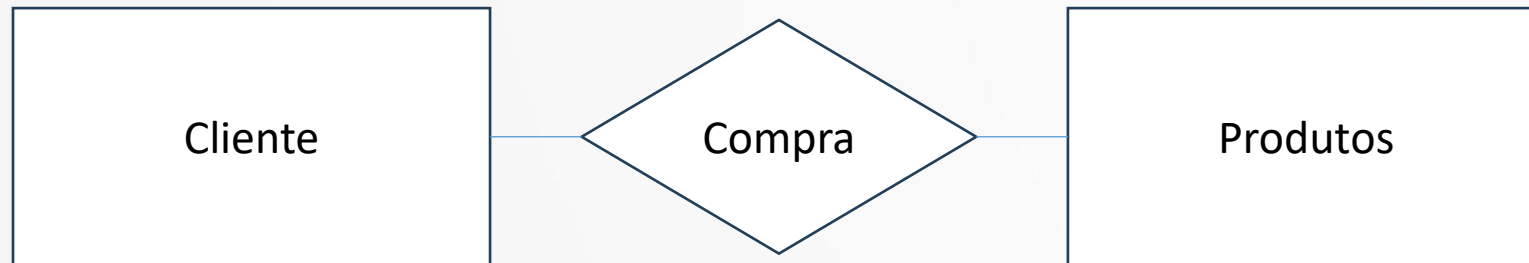
Um banco de dados relacional armazena dados em tabelas separadas, em vez de colocar todos os dados em um único local. A estrutura do banco de dados é organizada em tabelas que podem ter relacionamento umas com as outras. Esse modelo permite maior escalabilidade com baixa latência se bem modelado.



# Entidade Relacional

Primeiramente é importante saber que antes de uma tabela ser efetivamente implementada, damos o nome de Entidade.

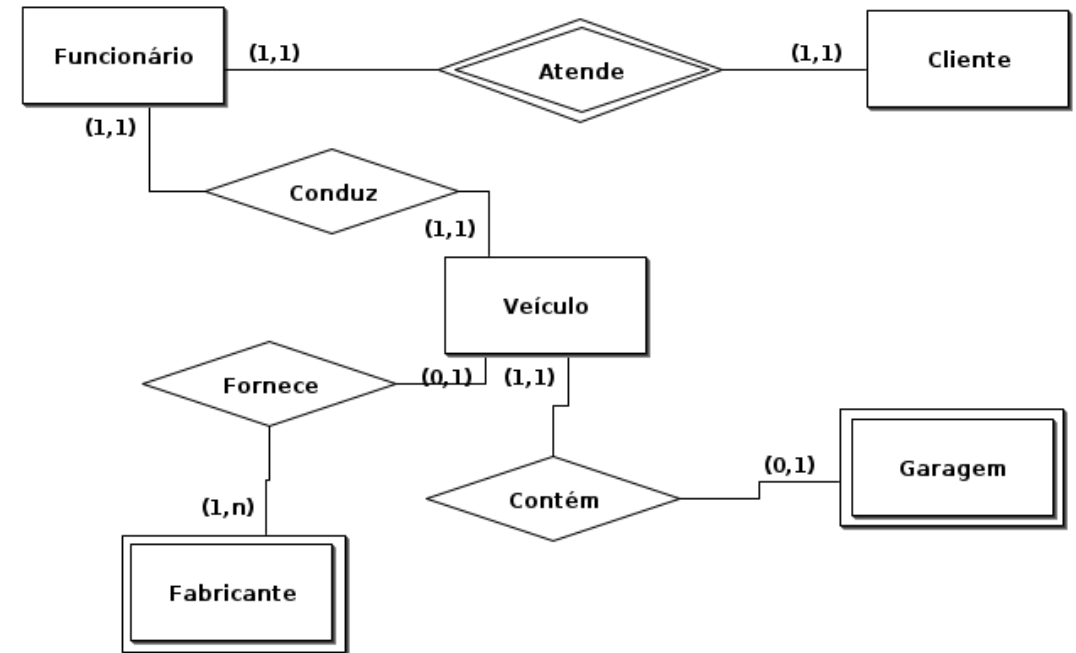
Toda Entidade, que também pode ser conhecida por Objeto, tem atributos e a partir desses, temos uma ação Ex:



# MER

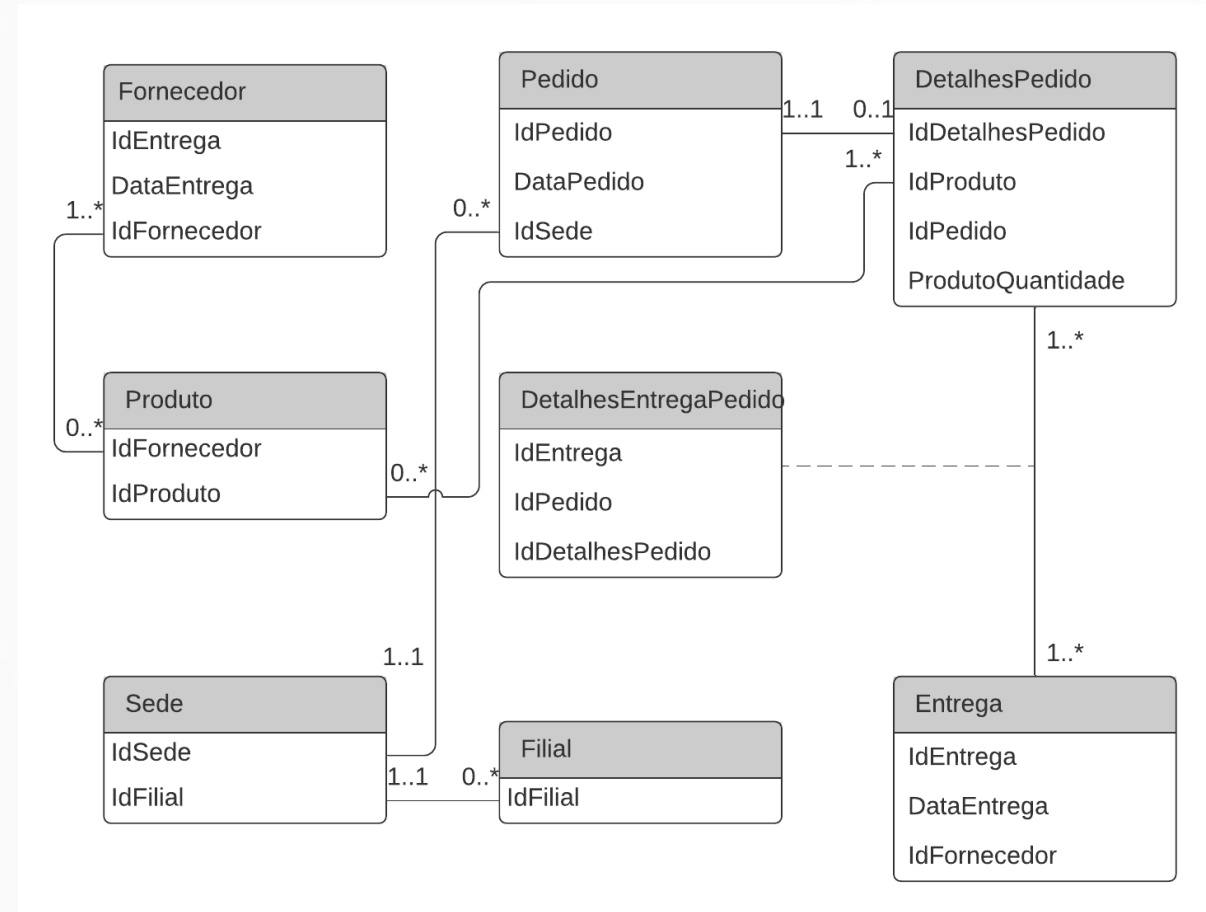
## MER – Modelo Entidade Relacionamento

É utilizado para descrever os objetos do mundo real através de entidades, com suas propriedades que são os atributos e os seus relacionamentos.



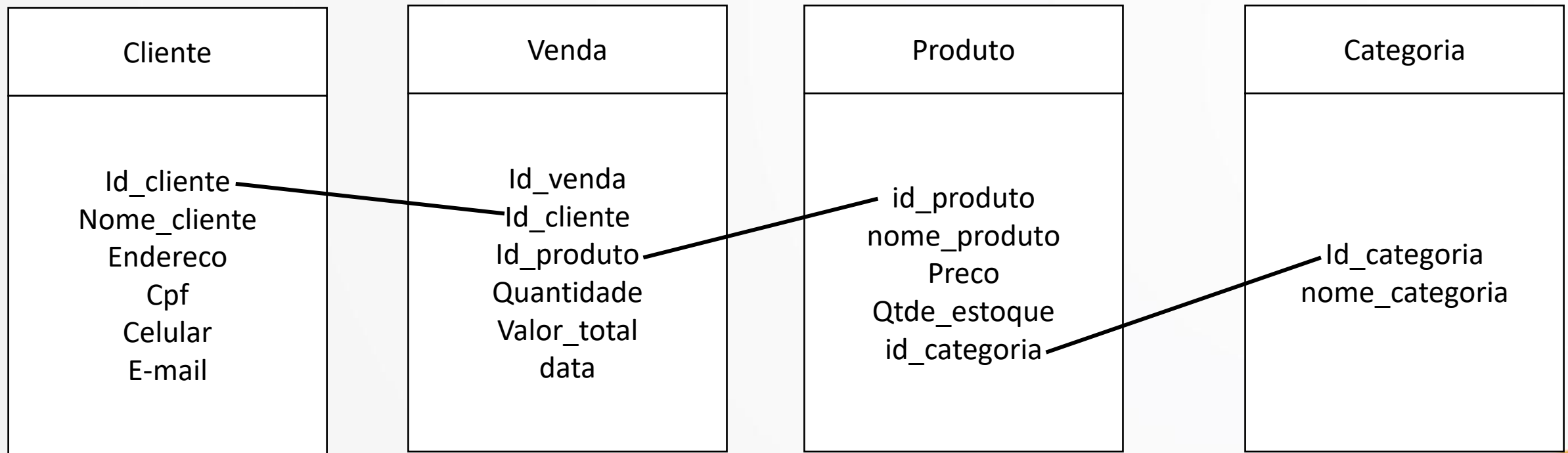
# DER

**DER** – Diagrama Entidade Relacionamento é utilizado para representar em forma gráfica o que foi descrito no **MER**



# Diagrama de Entidade Relacional: DER

O Diagrama é o detalhamento do modelo, onde as entidades tomam forma detalhando os campos e a ação não aparece.





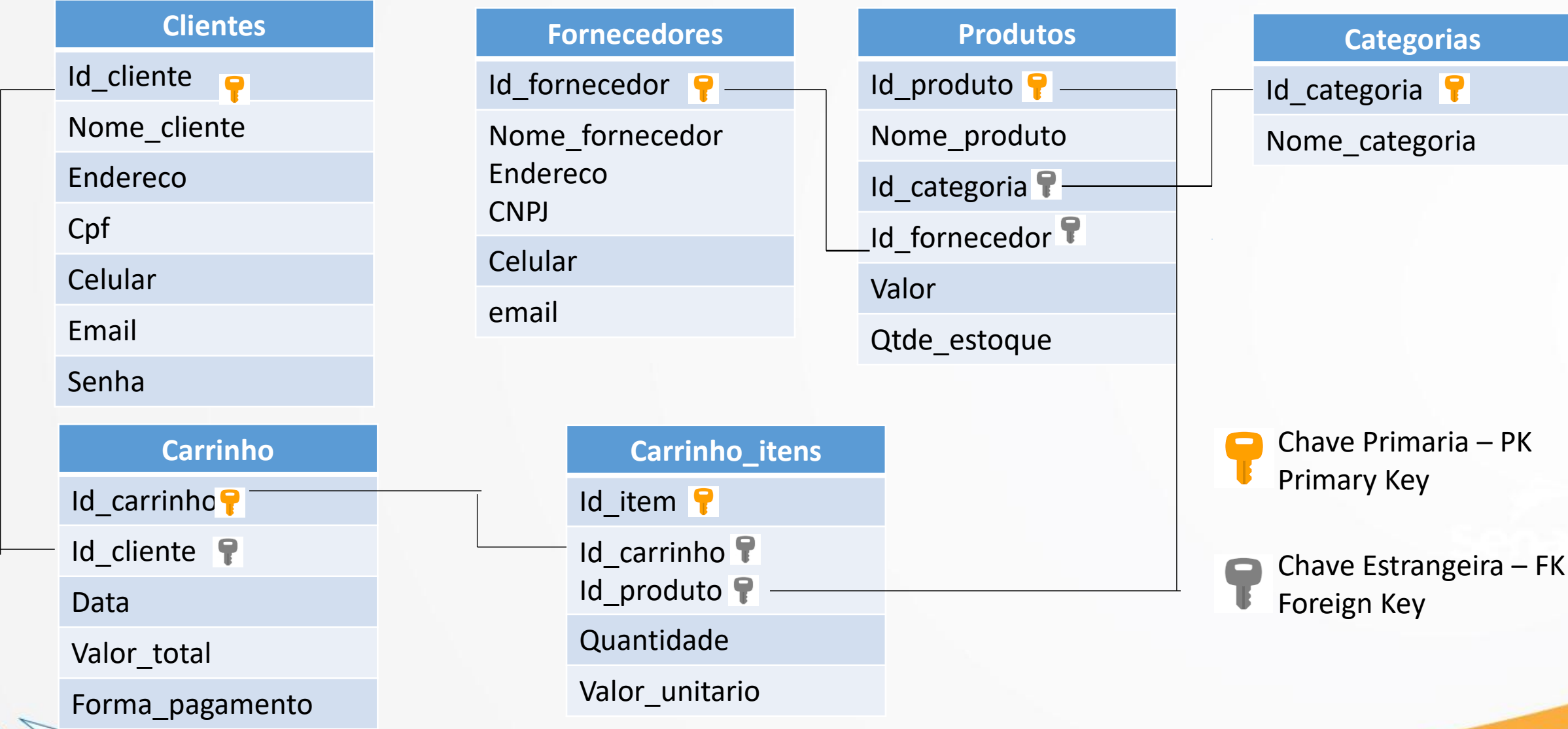
# Cardinalidade

A cardinalidade é um número que expressa o comportamento (número de ocorrências) de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento.

Existem dois tipos de cardinalidade: mínima e máxima. A cardinalidade máxima, expressa o número máximo de ocorrências de determinada entidade, associada a uma ocorrência da entidade em questão, através do relacionamento. A cardinalidade mínima, expressa o número mínimo de ocorrências de determinada entidade associada a uma ocorrência da entidade em questão através do relacionamento.

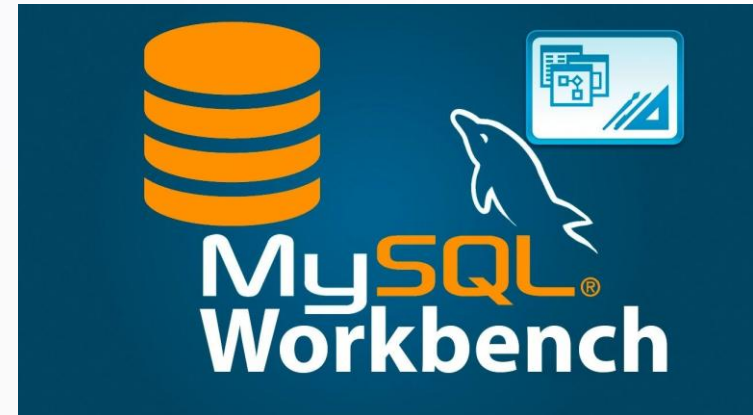
<https://www.devmedia.com.br/tecnologias-de-banco-de-dados-e-modelagem-de-dados/1660>

# Diagrama de Entidade Relacional: DER – E-Commerce Básico



# SGBD

Sistema Gerenciador de Banco de Dados



# Comandos

```
CREATE DATABASE lojatech; // criar o banco
```

```
USE lojatech; // acessar o banco
```

```
CREATE TABLE categoria(  
id_cat int(11) AUTO_INCREMENT,  
nome_categoria varchar(60) NOT NULL,  
PRIMARY KEY (id_cat))
```



# Criando chave estrangeira

// Criando tabela de produto com chave estrangeira

```
CREATE TABLE produto(  
id_produto int(11) PRIMARY KEY AUTO_INCREMENT,  
nome_produto varchar(60),  
descricao_produto text,  
valor double(11,2),  
id_cat int(11),  
foreign key(id_cat) references categoria(id_cat)  
)
```

# No caso da tabela já existir

// Caso a tabela já exista, podemos criar a chave estrangeira com o comando:

```
ALTER TABLE alunos  
ADD CONSTRAINT fk_alunos_curso  
FOREIGN KEY (id_curso)  
REFERENCES curso (id_curso);
```

# Resultado

loja2 categorias
id_categoria : int(11)
desc_categoria : varchar(60)
excluido : int(11)

loja2 produtos
id_produto : int(11)
desc_produto : varchar(80)
id_categoria : int(11)
preco_compra : float

# Alguns atributos dos campos

**AUTO\_INCREMENT** pode ser utilizado para automatizar um código que sirva de chave primária de uma tabela.

**PRIMARY KEY** define a chave primária da tabela, isto é, o campo que serve como chave da tabela e que não pode ser repetido.

**NOT NULL** define que um determinado campo seja de preenchimento obrigatório.



# Tipos de Campos

INT: Utiliza-se o tipo de dados INT para armazenar valores inteiros. Veja o exemplo:

```
INSERT INTO meus_valores (valor_inteiro) VALUES (10);
```

BIGINT: O tipo de dados BIGINT é semelhante ao tipo INT, mas pode armazenar valores inteiros maiores, adequado para armazenar valores inteiros grandes e nesse sentido, utilizado em tabelas com muitas linhas e colunas. Veja o exemplo:

```
INSERT INTO meus_valores (valor_grande) VALUES (9223372036854775808);
```

SMALLINT: O tipo de dados SMALLINT é usado para armazenar valores inteiros menores e é usado em tabelas com valores limitados. Veja o exemplo:

```
INSERT INTO meus_valores (valor_pequeno) VALUES (-32768);
```

TINYINT: O tipo de dados TINYINT é semelhante ao tipo SMALLINT, mas pode armazenar valores inteiros em um intervalo de -128 a 127. Veja o exemplo:

```
INSERT INTO meus_valores (valor_muito_pequeno) VALUES (-128);
```

# Tipos de Campos

**DECIMAL:** O tipo de dados DECIMAL é usado para armazenar valores numéricos com uma precisão especificada, adequado para armazenar valores monetários e financeiros. Veja o exemplo:

```
INSERT INTO meus_valores (valor_decimal) VALUES (123.45);
```

**FLOAT:** O tipo de dados FLOAT é usado para armazenar valores numéricos com uma precisão específica. Assim, armazenar valores de números decimais de até 6 dígitos. Veja o exemplo:

```
INSERT INTO meus_valores (valor_float) VALUES (3.141592);
```

**DOUBLE:** O tipo de dados DOUBLE é semelhante ao tipo FLOAT, mas pode armazenar valores numéricos com uma precisão maior, até 15-16 dígitos. Dessa forma, armazena valores decimais, como valores financeiros e científicos. Veja o exemplo:

```
INSERT INTO meus_valores (valor_double) VALUES (3.14159265358979323846);
```

# Tipos de Campos

## Tipos de dados de caractere

Os tipos de dados de caractere do MySQL são usados para armazenar dados de texto e caracteres.

**CHAR:** O tipo de dados CHAR é usado para armazenar dados de texto fixos com um comprimento específico. Nesse sentido, ele é usado ao criar tabelas para armazenar valores de texto com o mesmo tamanho em todas as colunas. Veja o exemplo que cria uma tabela “meus\_valores” com uma coluna “nome” e armazena textos de no máximo 10 caracteres de comprimento.

```
CREATE TABLE meus_valores (  
    nome CHAR(10)  
);
```

**VARCHAR:** O tipo de dados VARCHAR é semelhante ao tipo CHAR, mas permite que os valores de texto sejam variáveis, com comprimentos diferentes em cada coluna. Dessa forma, ele é usado ao criar tabelas para armazenar valores de texto com comprimentos variáveis. Veja o exemplo abaixo que cria uma tabela “meus\_valores” com uma coluna “email” que pode armazenar textos de até 200 caracteres de comprimento, sendo flexível quanto ao tamanho dos dados armazenados :

```
CREATE TABLE meus_valores (  
    email VARCHAR(200)  
);
```

**TEXT:** Utiliza-se o tipo de dados TEXT para armazenar dados de texto longos e complexos, com comprimentos que variam de 1 a 4 GB. Assim, armazena valores de texto com muito conteúdo, como artigos de blog e documentos. Veja o exemplo:

```
INSERT INTO meus_valores (descricao) VALUES ('Este é um exemplo de valor de texto longo');
```

# Tipos de Campos

**BLOB:** Utiliza-se o tipo de dados BLOB para armazenar dados binários, como imagens, vídeos e arquivos de documentos. Nesse sentido, armazena valores de grande porte e pode armazenar até 65.535 bytes de dados. Veja o exemplo:

```
INSERT INTO meus_valores (imagem) VALUES (FILE('/caminho/para/imagem.jpg'));
```

Tipos de dados de texto completo

**FULLTEXT:**

Utiliza-se o tipo de dados FULLTEXT para armazenar dados de texto e realizar consultas de texto completo. Assim, suporta consultas de texto completo em vários campos e permite o uso de operações de colisão e proximidade para encontrar resultados mais precisos.

```
CREATE TABLE livros (  
  id INT PRIMARY KEY,  
  titulo TEXT,  
  autor TEXT,  
  conteudo TEXT,  
  KEY(conteudo)(FULLTEXT)  
);
```

```
INSERT INTO livros (id, titulo, autor, conteudo) VALUES (1, 'Livro 1', 'Autor 1', 'Conteúdo 1, Conteúdo 2, Conteúdo 3');
```

Neste exemplo, a coluna “conteudo” é do tipo TEXT e é indexada com FULLTEXT. Assim, isso permite que o MySQL realize consultas de texto completo em todas as colunas indexadas com FULLTEXT.



# Tipos de Campos

## Tipos de dados de data e hora do MySQL

Os tipos de dados de data e hora do MySQL são usados para armazenar valores de data e hora.

**DATE:** Utiliza-se o tipo de dados DATE para armazenar apenas a data sem a hora. Portanto, está composto por 8 bytes e pode armazenar valores de data no intervalo de 1000 a 9999 para o ano e de 0 a 65535 para o dia do mês. Veja o exemplo:

```
INSERT INTO meus_valores (data) VALUES ('2022-03-14');
```

**TIME:** Utiliza-se o tipo de dados TIME para armazenar apenas a hora sem a data. Veja o exemplo:

```
INSERT INTO meus_valores (hora) VALUES ('13:30:00');
```

**DATETIME:** Utiliza-se o tipo de dados DATETIME para armazenar valores de data e hora juntos. Veja o exemplo:

```
INSERT INTO meus_valores (data_hora) VALUES ('2022-03-14 13:30:00');
```

**TIMESTAMP:** O tipo de dados TIMESTAMP é semelhante ao tipo DATETIME, mas inclui a precisão do segundo. Veja o exemplo:

```
INSERT INTO meus_valores (data_hora_segundos) VALUES ('2022-03-14 13:30:00');
```

# Tipos de Campos

## Tipos de dados binários do MySQL

O MySQL suporta vários tipos de dados binários, cada um dos quais tem suas próprias características e utilidades. Aqui estão:

BIT: O tipo de dados BIT é um tipo binário simples que pode ter um único valor de 0 ou 1. Veja o exemplo:

```
CREATE TABLE pessoas (  
  id INT PRIMARY KEY,  
  sexo BIT);
```

```
INSERT INTO pessoas (id, sexo) VALUES (1, 1);
```

# Tipos de Campos

**BITMAP:** O tipo de dados BITMAP é um tipo binário mais complexo que permite armazenar vários valores de bits em uma única coluna. Veja o exemplo:

```
CREATE TABLE pessoas (  
    id INT PRIMARY KEY,  
    hobbies BITMAP);
```

```
INSERT INTO pessoas (id, hobbies) VALUES (1, 0b00100000);
```

# Tipos de Campos

SET: O tipo de dados SET é um tipo binário que permite armazenar vários valores de bits em uma única coluna. Nesse sentido, o SET é semelhante ao BITMAP, porém mais fácil de usar e oferece uma melhor desempenho. Veja o exemplo:

```
CREATE TABLE pessoas (  
  id INT PRIMARY KEY,  
  hobbies SET);
```

```
INSERT INTO pessoas (id, hobbies) VALUES (1, 'leitura, cinema');
```

ENUM: O tipo de dados ENUM é um tipo binário que permite armazenar um valor de uma lista prédefinida de valores. Veja o exemplo:



# Tipos de Campos

```
CREATE TABLE cores (  
  id INT PRIMARY KEY,  
  cor ENUM('verde', 'azul', 'roxo'));
```

```
INSERT INTO cores (id, cor) VALUES (1, 'verde');
```

# Tipos de Campos

## Tipos de dados de intervalo

**YEAR:** Utilizamos o tipo de dados YEAR para armazenar um número de ano, armazena valores de 1 a 9999. Dessa forma, muito usado para armazenar a data de nascimento ou a data de início de um contrato, por exemplo:

```
CREATE TABLE pessoas (  
  id INT PRIMARY KEY,  
  nascimento YEAR);
```

```
INSERT INTO pessoas (id, nascimento) VALUES (1, 1990);
```

# Tipos de Campos

MONTH: Utilizamos o tipo de dados MONTH para armazenar um número de mês. Assim, armazena valores de 1 a 12, por exemplo:

```
CREATE TABLE pessoas (  
    id INT PRIMARY KEY,  
    nascimento MONTH);
```

```
INSERT INTO pessoas (id, nascimento) VALUES (1, 1);
```

# Tipos de Campos

DAY: usamos o tipo de dados DAY para armazenar um número de dia da semana. Nesse sentido, armazena valores de 0 a 6, onde 0 representa domingo e 6 representa sábado, por exemplo:

```
CREATE TABLE pessoas (  
  id INT PRIMARY KEY,  
  nascimento DAY);
```

```
INSERT INTO pessoas (id, nascimento) VALUES (1, 0);
```

## Tipos de dados geométricos do MySQL

Os tipos de dados geométricos do MySQL são usados para armazenar dados de geometria espaciais, como pontos, linhas e polígonos.

# Tipos de Campos

**POINT:** Utiliza-se o tipo de dados POINT para armazenar um ponto em uma superfície. Nesse sentido, representado por uma tupla de valores (x, y), onde x e y são os valores de latitude e longitude do ponto.

```
CREATE TABLE locais (  
    id INT PRIMARY KEY,  
    posicao POINT);
```

```
INSERT INTO locais (id, posicao) VALUES (1, (10.5, -69.5));
```

Neste exemplo, a coluna “posicao” é do tipo POINT e armazena a localização de um ponto em uma superfície. O valor “(10.5, -69.5)” neste caso representa a latitude e longitude do ponto.

# Tipos de Campos

LINE: Usamos o tipo de dados LINE para armazenar uma linha em uma superfície. Assim, representado por uma sequência de pontos (x1, y1, x2, y2, ...), onde cada ponto representa um ponto na linha. Veja o exemplo:

```
CREATE TABLE fronteiras (  
  id INT PRIMARY KEY  
  fronteira LINE);
```

```
INSERT INTO fronteiras (id, fronteira) VALUES (1, ('0,0' , '10,0', '10,10', '0,10',  
'0,0'));
```

Assim, Neste exemplo, a coluna “fronteira” é do tipo LINE e armazena a localização de uma linha em uma superfície.



# Tipos de Campos

POLYGON: Utilizamos o tipo de dados POLYGON para armazenar um polígono em uma superfície. Nesse sentido, representado por uma sequência de linhas (x1, y1, x2, y2, ..., xn, yn), onde cada linha representa um ponto na linha.

```
CREATE TABLE poligonos (  
  id INT PRIMARY KEY,  
  poligono POLYGON);
```

```
INSERT INTO poligonos (id, poligono) VALUES (1, POLYGON  
( 'LINESTRING(-10.5 -69.5, 0 -69.5, 0 -10.5, -10.5 -69.5)'));
```

Neste exemplo, a coluna “poligono” é do tipo POLYGON e armazena a localização de um polígono em uma superfície.

# Tipos de Campos

GEOMETRY: O tipo de dados GEOMETRY é um tipo de dados que suporta vários tipos de geometria espaciais, incluindo POINT, LINE e POLYGON. Nesse sentido, utilizamos para armazenar dados de geometria espaciais mais complexos, como curvas e polígonos irregulares.

Tipos de dados de Unicode

UTF-8: Utilizamos o tipo de dados UTF-8 para armazenar dados em formato Unicode em byte.

```
CREATE TABLE example_table (  
  id INT NOT NULL AUTO_INCREMENT,  
  name VARCHAR(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci,  
  PRIMARY KEY (id)  
);
```

Neste exemplo, definimos a coluna “name” como do tipo VARCHAR e utiliza o conjunto de caracteres utf8mb4, que lida com caracteres em branco de até 3 bytes, o que inclui a maioria dos idiomas e scripts.

# Tipos de Campos

UTF-16: Utilizamos o tipo de dados UTF-16 para armazenar dados em formato Unicode em 16 bits. Nesse sentido, utilizamos muito em aplicações que exigem mais precisão de ponto flutuante.

```
CREATE TABLE example_table (  
    id INT NOT NULL AUTO_INCREMENT,  
    name VARBINARY(65535) CHARACTER SET utf16 COLLATE  
    utf16_general_ci,  
    PRIMARY KEY (id)  
);
```

# CRUD

CREATE - INSERIR DADOS DE REGISTRO EM UMA TABELA

READ - LEITURA (FAZER UMA CONSULTA DE REGISTROS)

UPDATE - ALTERAÇÃO DE DADOS DE REGISTRO EM UMA TABELA

DELETE - EXCLUSÃO DE REGISTROS EM UMA TABELA

# Operações com banco de dados

Excluir um banco de dados:

`DROP DATABASE <nomedobanco>`

Excluir uma tabela:

`DROP TABLE <nomedatabela>`

# CRUD

## C – Create

Inclusão de registros em uma tabela

- INSERT INTO <nometabela> (campo1,campo2,etc...)

Values ('valor1','valor2','etc')

insert into produto (nome\_produto,descricao\_produto,valor,id\_cat)

values ('Notebook ACER 2500','I5 16Gb RAM 1TB SSD','2500','3')

## R – Read

Consulta ou pesquisa dentro de uma tabela

- SELECT \* ou campo1,campo2 FROM <nometabela> WHERE id=1

Podemos ver todos os campos ou alguns.



# CRUD

U - UPDATE, Alterar/Editar campos em uma tabela

UPDATE <nometabela> SET campo1='valor2',campo2='valor2' Where  
id = 1 or id=3

```
update produto set valor=7000 where id_produto=3;
```

D – DELETE, Exclui um registro em uma tabela

DELETE FROM <nometabela> WHERE id=1

DELETE FROM `produto` WHERE ID\_produto=5

# Relacionamento INNER JOIN

A cláusula INNER JOIN compara cada linha da tabela A com as linhas da tabela B para encontrar todos os pares de linhas que satisfazem a condição de junção. Se a condição de junção for avaliado como TRUE, os valores da coluna das linhas correspondentes das tabelas A e B serão combinados em uma nova linha e incluídos no conjunto de resultados.

```
select * from produto inner join categoria on  
produto.id_cat=categoria.id_cat
```

// Aqui somamos os valores e agrupamos por categoria

```
select nome_categoria,sum(valor) as total from produto inner join categoria on  
produto.id_cat=categoria.id_cat GROUP by nome_categoria;
```

// Aqui contamos os registros agrupados por categoria

```
select nome_categoria,count(*) as total from produto inner join categoria on  
produto.id_cat=categoria.id_cat GROUP by nome_categoria;
```

```
select nome_categoria,count(*) as total from produto inner join categoria on  
produto.id_cat=categoria.id_cat where categoria.nome_categoria='Notebooks';
```

```
select nome_categoria,count(*) as total from produto inner join categoria on  
produto.id_cat=categoria.id_cat where categoria.id_cat=3;
```

# Mysql X MariaDB

MariaDB é um sistema de gerenciamento de banco de dados que surgiu como fork do MySQL, criado pelo próprio fundador do projeto após sua aquisição pela Oracle.

Se verificarmos a versão atual do Mysql, veremos o número da versão e a palavra MariaDB

Portanto houve união das tecnologias em um único banco de dados.

**\*\* fork** é basicamente a modificação do código-fonte da blockchain do projeto.

**\*\* Blockchain** é uma base compartilhada de dados que faz o registro e validação de transações digitais, trocas de informações processadas por usuários de uma rede descentralizada de computadores.

# Comandos importantes

```
SELECT VERSION(); // Exibe a versão do mysql
```

```
SELECT CURRENT_DATE; // Exibe a data atual
```

Você pode exibir a data com outro nome:

```
SELECT CURRENT_DATE as data_atual;
```

```
SELECT NOW() as data_hora; // Exibe data e hora
```

# Funções matemáticas

Podemos fazer cálculos avançados com a consulta no Mysql.

Ex:

```
SELECT SIN(PI()/4) as resultado;
```

```
SELECT NUMERO,SIN(NUMERO/4) AS RESULTADO FROM  
NOMETABELA WHERE ID=1
```

```
SELECT PRECO*QUANTIDADE AS VALOR FROM PRODUTO WHERE  
ID=1
```

```
SELECT (4+1)*5 as resultado
```

```
SELECT round(pi(),2) as numeropi; // round função para  
arredondar
```



# Links importantes

Funções matemáticas:

<https://dev.mysql.com/doc/refman/8.0/en/numeric-functions.html>

Funções String:

<https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

- `Select min(campo) // mostrará o menor valor`
- `Select max(campo) // mostrará o maior valor`
- `Select avg(valor) from produto; // retorna a médias dos valores na tabela`

# Exemplo interessante

`SELECT * FROM `pessoas``

☐ Perfil [ [Editar em linha](#) ] [ [Editar](#) ] [ [Demonstrar SQL](#) ] [ [Criar código PHP](#) ] [ [Atualizar](#) ]

☐ Mostrar tudo | Número de linhas: 25 ▼ | Filtrar linhas:

Opções extras

				id_pessoa	nome_pessoa
<input type="checkbox"/>	Editar	Copiar	Remover	1	Camila Batista
<input type="checkbox"/>	Editar	Copiar	Remover	2	João Carlos Rossi
<input type="checkbox"/>	Editar	Copiar	Remover	3	Maria Izabel Assis
<input type="checkbox"/>	Editar	Copiar	Remover	4	Roberto de moura

Imagine que precisamos exibir somente o primeiro nome dessa tabela.

A lógica seria, exibir o nome da primeira posição até encontrar o primeiro espaço em branco:

```
SELECT *,SUBSTRING(nome_pessoa,1,LOCATE(' ', nome_pessoa)) as primeiro_nome FROM  
pessoas
```

# Resultado



Select \*,SUBSTRING(nome\_pessoa,1,LOCATE(' ',nome\_pessoa)) as primeiro\_nome FROM pessoas

```
SELECT *,SUBSTRING(nome_pessoa,1,LOCATE(' ', nome_pessoa)) as primeiro_nome FROM pessoas;
```

☐ Perfil [ [Editar em linha](#) ] [ [Editar](#) ] [ [Demonstrar SQL](#) ] [ [Criar código PHP](#) ] [ [Atualizar](#) ]

☐ Mostrar tudo | Número de linhas: 25 ▼ Filtrar linhas:  Ordenar

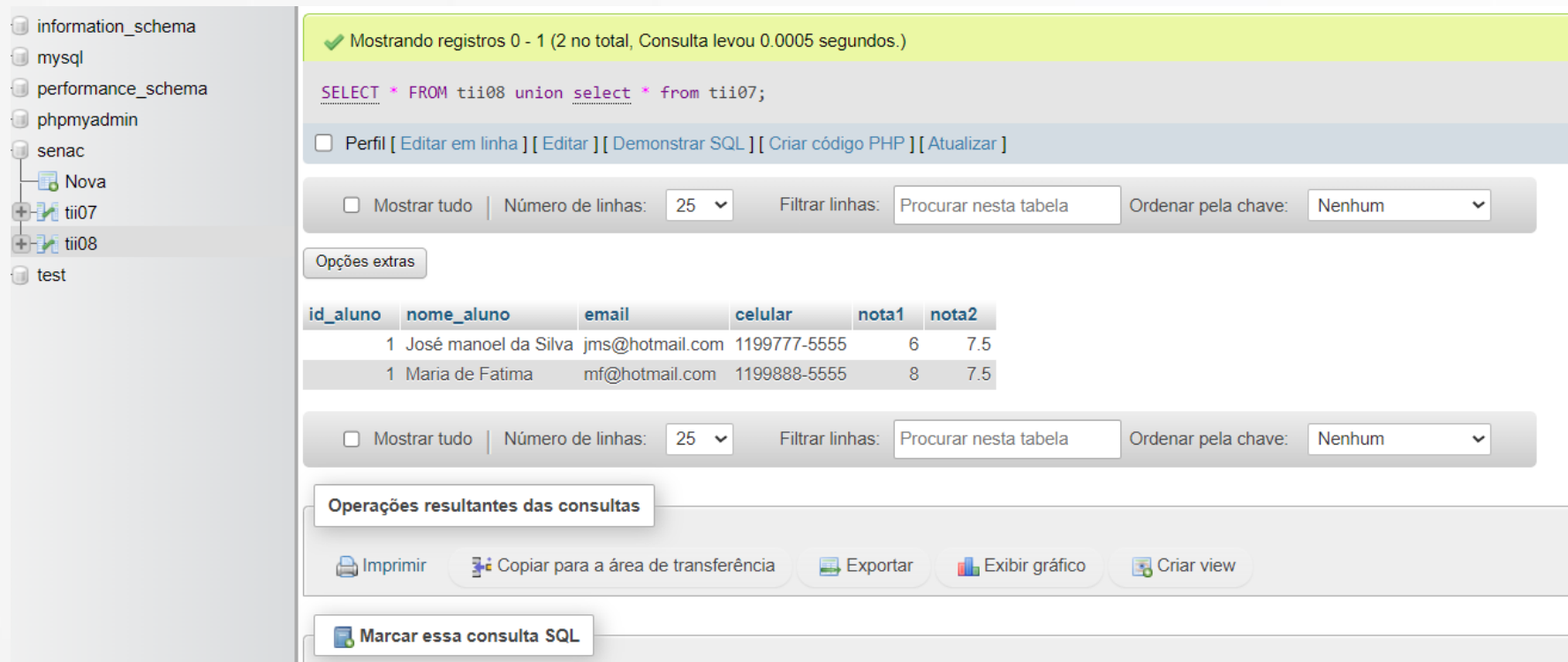
Opções extras

				id_pessoa	nome_pessoa	primeiro_nome
<input type="checkbox"/>	 Editar	 Copiar	 Remover	1	Camila Batista	Camila
<input type="checkbox"/>	 Editar	 Copiar	 Remover	2	João Carlos Rossi	João
<input type="checkbox"/>	 Editar	 Copiar	 Remover	3	Maria Izabel Assis	Maria
<input type="checkbox"/>	 Editar	 Copiar	 Remover	4	Roberto de moura	Roberto

# Mais comandos

UNION – Através do UNION podemos unir duas tabelas de igual estrutura em uma consulta. Ex:

```
SELECT * FROM tii08 union select * from tii07;
```



The screenshot shows the phpMyAdmin interface. On the left, a tree view shows the database structure with 'senac' selected, containing 'Nova', 'tii07', 'tii08', and 'test'. The main panel displays the execution of the SQL query: `SELECT * FROM tii08 union select * from tii07;`. A status bar at the top indicates 'Mostrando registros 0 - 1 (2 no total, Consulta levou 0.0005 segundos.)'. Below the query, there are controls for 'Perfil', 'Editar em linha', 'Editar', 'Demonstrar SQL', 'Criar código PHP', and 'Atualizar'. A table of results is shown with columns: id\_aluno, nome\_aluno, email, celular, nota1, and nota2. The table contains two rows of data. Below the table, there are controls for 'Mostrar tudo', 'Número de linhas' (set to 25), 'Filtrar linhas' (Procurar nesta tabela), and 'Ordenar pela chave' (Nenhum). At the bottom, there is a section for 'Operações resultantes das consultas' with buttons for 'Imprimir', 'Copiar para a área de transferência', 'Exportar', 'Exibir gráfico', and 'Criar view'. A button 'Marcar essa consulta SQL' is also present.

id_aluno	nome_aluno	email	celular	nota1	nota2
1	José manael da Silva	jms@hotmail.com	1199777-5555	6	7.5
1	Maria de Fatima	mf@hotmail.com	1199888-5555	8	7.5

# Ver estrutura de uma tabela

```
SHOW COLUMNS FROM nome_tabela
```

O Comando acima permite visualizar a estrutura de uma tabela

```
SHOW COLUMNS FROM tii07;
```

☐ Perfil [ [Editar em linha](#) ] [ [Editar](#) ] [ [Criar código PHP](#) ] [ [Atualizar](#) ]

Opções extras

Field	Type	Null	Key	Default	Extra
id_aluno	int(11)	NO	PRI	NULL	auto_increment
nome_aluno	varchar(80)	NO		NULL	
email	varchar(100)	NO		NULL	
celular	varchar(15)	NO		NULL	
nota1	float	NO		NULL	
nota2	float	NO		NULL	



# Limite nas Consultas

Caso sua tabela seja muito grande, você pode limitar a quantidade de registros que quer mostrar:

Ex: `SELECT * FROM tii07 limit 1`

Nesse caso, estamos exibindo apenas 1 registro



# Between

- Filtra valores dentro de um intervalo especificado.

Ex:

```
SELECT * FROM pessoas WHERE data_nascimento BETWEEN '2000-01-01' and '2005-12-01';
```

# IN

A Função IN retorna os registros selecionados em uma lista

Ex:

```
SELECT * FROM `pessoas` WHERE ID_PESSOA IN(1,2);
```

# Procedures de Functions

Procedures e Functions são rotinas definidas no banco de dados, identificadas por um nome pelo qual podem ser invocadas. Dessa forma, um procedimento desses pode executar uma série de instruções, receber parâmetros e retornar valores.

# Function

Uma função é usada para gerar um valor que pode ser usado em uma expressão. Esse valor é geralmente baseado em um ou mais parâmetros fornecidos à função. As funções são executadas geralmente como parte de uma expressão.

O MySQL possui diversas funções internas que o desenvolvedor pode utilizar, e também permite que criemos nossas próprias funções, e é isso que mostraremos como fazer agora.

# Functions

DELIMITER \$\$

CREATE FUNCTION nome\_function (parâmetros)

RETURNS retorna\_valor

BEGIN

/\*CORPO DA FUNÇÃO\*/

END \$\$

DELIMITER ;



# Function Exemplo

```
CREATE FUNCTION fn_teste (a DECIMAL(10,2), b INT)  
RETURNS INT  
RETURN a * b;
```

Invocando a função:

```
SELECT fn_teste(2.5, 4) AS Resultado;
```

# Functions Exemplo

```
CREATE FUNCTION fn_verSalario (a SMALLINT)
RETURNS VARCHAR(60)
RETURN
(SELECT CONCAT('O salario de ', nome, ' é ', salario)
FROM funcionarios
WHERE idfuncionarios = a);

SELECT fn_verSalario (2) AS Resultado;
```

# Procedure

Uma PROCEDURE (também chamada stored procedure) é uma subrotina que fica armazenada no banco de dados. Uma PROCEDURE tem um nome, uma lista de parâmetros e declarações de comandos SQL.

# Procedures

DELIMITER \$\$

CREATE PROCEDURE nome\_procedimento (parâmetros)

BEGIN

/\*CORPO DO PROCEDIMENTO\*/

END \$\$

DELIMITER ;

# Procedures Exemplo

```
DELIMITER $$
```

```
CREATE PROCEDURE ContaPacientes(OUT quantidade INT)  
BEGIN  
SELECT COUNT(*) INTO quantidade FROM pacientes;  
END $$  
DELIMITER ;
```

Para executar:

```
CALL ContaPacientes(@total);  
SELECT @total;
```

# Formatando Valor para reais

```
select nome,FORMAT(salario,2,'de_DE') as remuneracao from  
funcionarios;
```

O Código de\_DE é responsável pela conversão no formato da nossa moeda.



# Views

Uma View é um objeto que pertence a um banco de dados, definida baseada em declarações SELECT's, retornando uma determinada visualização de dados de uma ou mais tabelas. Esses objetos são chamados por vezes de “virtual tables”, formada a partir de outras tabelas que por sua vez são chamadas de “based tables” ou ainda outras Views.

# Views

Exemplo: CREATE OR REPLACE VIEW vw\_pessoas AS  
SELECT nome\_pessoa  
FROM pessoas;

Foi criada uma tabela de apoio em views com o nome vw\_pessoas contendo somente o campo nome

Consideremos a criação dessa tabela como um facilitador no processamento de consultas antes de ser apresentada ao usuário

O Replace permite alterar a view existente.

Para executar: SELECT \* FROM vw\_pessoas

# TRIGGERS

Um Trigger (gatilho) no MySQL é um conjunto de instruções SQL que residem em um catálogo do sistema. É um tipo especial de procedimento armazenado que é invocado automaticamente em resposta a um evento. Cada gatilho é associado a uma tabela, que é ativada em qualquer instrução **DML**.

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

São os comandos que interagem com os objetos do banco.

São comandos DDL : CREATE, ALTER e DROP

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

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

São comandos DML : INSERT, DELETE e UPDATE

# TRIGGERS

Para melhor compreensão, podemos criar um trigger para que automaticamente após o usuário ter deletado um registro, ser inserido um registro em uma tabela de logs com a cópia do registro excluído, o usuário que excluiu e a data.

Ou, quando fechamos um carrinho de compras em um e-commerce, atualizar o estoque, gerar o recebimento no financeiro, agendar a entrega da compra, enviar um aviso para o gerente de vendas e outras coisas, de acordo com a regra de negócio de forma mais rápida e segura, evitando muitas linhas de programação em outra linguagem.

# Exercício com Trigger

-- 1. Criar a tabela de exemplo:

```
CREATE TABLE Novoproduto (  
  idProduto INT NOT NULL AUTO INCREMENT,  
  Nome_Produto VARCHAR(45) NULL,  
  Preço_Normal DECIMAL(10,2) NULL,  
  Preço_Desconto DECIMAL(10,2) NULL,  
  PRIMARY KEY (idProduto));
```

-- 2. Criar o Trigger:

```
CREATE TRIGGER tr_desconto BEFORE INSERT  
ON Novoproduto  
FOR EACH ROW  
SET NEW.Preço_Desconto = (NEW.Preço_Normal * 0.90);
```

# Exercício com Trigger

-- 3. Executar uma inserção que irá disparar o Trigger:

```
INSERT INTO Novoproduto (Nome_Produto, Preco_Normal)  
VALUES ("Impressora", 900), ("Monitor", 800);
```

-- 4. Verificar se trigger foi disparado observando o preço com desconto:

```
SELECT * FROM Novoproduto;
```

Para exibir as triggers: SHOW TRIGGERS

Para excluir um trigger: DROP TRIGGER tr\_desconto



# Relacionamentos JOIN

Estudar:

<https://www.devmedia.com.br/sql-join-entenda-como-funciona-o-retorno-dos-dados/31006#1>



# Registro único

Na sua tabela você tem a necessidade de que o conteúdo de um determinado campo seja único, por exemplo o CPF de um aluno para que não haja duplicidade. Você pode resolver isso de duas maneiras:

A primeira é pela lógica de programação, onde antes de dar o INSERT, você faça uma consulta pra ver se já existe esse registro na tabela.

A Segunda é criando uma chave indexadora do tipo UNIQUE, para que não faça o INSERT se o registro já existir. O problema é que o usuário não saberá que não gravou, pois não haverá retorno do banco para a linguagem de programação.

# Registro Único

// Para uma chave única

```
CREATE TABLE clientes (  
    ID_Cliente int NOT NULL,  
    nome_cliente varchar(100) NOT NULL,  
    cpf varchar(25) NOT NULL,  
    UNIQUE (cpf),  
    PRIMARY KEY (ID_Cliente)  
);
```

# Registro único

// Múltiplas chaves únicas

```
CREATE TABLE clientes (  
    ID_Cliente int NOT NULL,  
    nome_cliente varchar(100) NOT NULL,  
    cpf varchar(25) NOT NULL,  
    CONSTRAINT UC_Clientes UNIQUE (nome_cliente,cpf)  
);  
// mais Informações: https://www.w3schools.com/mysql/mysql\_unique.asp
```

# Códigos de erro do mysql

## Erro 1064: Comando de sintaxe incorreta

O erro 1064 é um dos erros mais comuns do MySQL e ocorre quando um comando SQL possui uma sintaxe incorreta. Isso pode acontecer quando há um erro de digitação, falta de aspas ou uso incorreto de palavras-chave. Para corrigir esse erro, é necessário revisar o comando SQL e corrigir qualquer erro de sintaxe.

## Erro 1045: Acesso negado para o usuário

O erro 1045 ocorre quando o MySQL não permite o acesso de um usuário ao banco de dados. Isso pode acontecer quando as credenciais de acesso estão incorretas ou quando o usuário não possui permissões suficientes. Para resolver esse erro, é necessário verificar as credenciais de acesso e garantir que o usuário tenha as permissões adequadas.

## Erro 1215: Não é possível adicionar restrição de chave estrangeira

O erro 1215 ocorre quando há um problema ao adicionar uma restrição de chave estrangeira a uma tabela. Isso pode acontecer quando a coluna referenciada não existe ou quando há uma incompatibilidade entre os tipos de dados das colunas. Para solucionar esse erro, é necessário verificar a existência da coluna referenciada e garantir que os tipos de dados sejam compatíveis.

# Códigos de erro do mysql

Erro 2002: Nenhum servidor MySQL está sendo executado

O erro 2002 ocorre quando o MySQL não consegue se conectar ao servidor. Isso pode acontecer quando o servidor MySQL não está em execução ou quando a configuração de conexão está incorreta. Para resolver esse erro, é necessário verificar se o servidor MySQL está em execução e revisar as configurações de conexão.

Erro 1062: Duplicação de chave primária

O erro 1062 ocorre quando há uma tentativa de inserir um registro com uma chave primária que já existe na tabela. Isso pode acontecer quando há uma violação da unicidade da chave primária. Para corrigir esse erro, é necessário verificar se o valor da chave primária já existe na tabela antes de inserir o registro.

Erro 1054: Coluna desconhecida

O erro 1054 ocorre quando uma consulta SQL faz referência a uma coluna que não existe na tabela. Isso pode acontecer quando há um erro de digitação no nome da coluna ou quando a coluna foi renomeada ou removida. Para resolver esse erro, é necessário verificar se o nome da coluna está correto e se ela existe na tabela.



# Códigos de erro do mysql

## Erro 1364: Campo obrigatório não preenchido

O erro 1364 ocorre quando uma tentativa de inserir um registro em uma tabela falha porque um campo obrigatório não foi preenchido. Isso pode acontecer quando um campo que possui a restrição NOT NULL não recebe um valor durante a inserção. Para solucionar esse erro, é necessário garantir que todos os campos obrigatórios sejam preenchidos durante a inserção.

## Erro 121: Tabela não existe

O erro 121 ocorre quando uma consulta SQL faz referência a uma tabela que não existe no banco de dados. Isso pode acontecer quando há um erro de digitação no nome da tabela ou quando a tabela foi renomeada ou removida. Para corrigir esse erro, é necessário verificar se o nome da tabela está correto e se ela existe no banco de dados.

## Erro 2006: Tempo de espera excedido

O erro 2006 ocorre quando uma conexão com o servidor MySQL é perdida devido a um tempo de espera excedido. Isso pode acontecer quando uma consulta SQL leva muito tempo para ser executada ou quando há problemas de rede. Para resolver esse erro, é necessário otimizar as consultas SQL e verificar a conexão de rede.

# Códigos de erro do mysql

## Erro 1005: Não é possível criar tabela

O erro 1005 ocorre quando há um problema ao criar uma tabela no banco de dados. Isso pode acontecer quando há uma incompatibilidade entre as colunas referenciadas por chaves estrangeiras ou quando há um erro de sintaxe na definição da tabela. Para solucionar esse erro, é necessário verificar as definições das colunas e garantir que não haja incompatibilidades.

## Erro 1452: Restrição de chave estrangeira falhou

O erro 1452 ocorre quando uma tentativa de inserir ou atualizar um registro falha devido a uma violação de uma restrição de chave estrangeira. Isso pode acontecer quando o valor da chave estrangeira não existe na tabela referenciada. Para corrigir esse erro, é necessário verificar se o valor da chave estrangeira existe na tabela referenciada antes de realizar a inserção ou atualização.

# Códigos de erro do mysql

## Erro 126: Índice duplicado

O erro 126 ocorre quando há uma tentativa de criar um índice com um nome que já existe na tabela. Isso pode acontecer quando há uma duplicação de nomes de índices. Para resolver esse erro, é necessário verificar se o nome do índice já está sendo usado na tabela e escolher um nome único.