

# Banco de Dados

## (MySQL – Workbench)

- São conjuntos de tabelas. Tabelas são conjuntos de registros. Registros são compostos por campos.

### #Classificação:

**DDL** – Data Definition Language, são comandos de definição: (Create Database, Create Table, Alter Table, Drop Table)

**DML** – Data Manipulation Language – Comandos de manipulação de dados: (Insert to, Update, Delete, Truncate)

**DQL** – Data Query Language – Comandos que seleciona os dados: (Select)

## #Criar Banco de Dados

Create database (ex: cadastro)

## #Criar Tabela

Create table (ex: pessoas) (

nome,

idade,

sexo,

peso,

altura,

nacionalidade

;

## #Tipos variáveis de Banco de Dados – Tipos Primitivos

A) Numérico

- a. Inteiro – TinyInt, BigInt, SmallInt, MediumInt
- b. Real – Decimal, Float, Double, Real
- c. Lógico – Bit, Boolean

**B) Data/Tempo** – Date, DateTime, TimeStamp, Time, Year

**C) Literal**

- a) Character – Char, Varchar
- b) Texto – TinyText, MediumText, LongText
- c) Binário – TinyBlob, Blob, MediumBlob, LongBlob
- d) Coleção – Enum, Set

**D) Espacial**

- a) Geometry, Point, Polygon, MultiPolygon

#Excluindo a tabela anterior

```
drop database (ex: cadastro);
```

#Criando novamente o banco de dados  
com utf8mb4

```
create database (ex: cadastro)
```

```
default character set utf8mb4
```

```
default collate utf8mb4_general_ci;
```

#**CONSTRAINTS** são regras que definimos  
para a criação e dimensionalidade do  
banco de dados.

## #Criar tabela definindo CONSTRAINTS e VARIÁVEIS

use (ex: cadastro)

create table (ex: pessoas) (

nome varchar(30) not null,

nascimento date,

sexo enum ('M', 'F'),

peso decimal (5,2),

altura decimal (3,2),

nacionalidade varchar(20)

) default charset = utf8mb4;

## #Usando o PRIMARY KEY (Chave Primária)

```
create table (ex: pessoas) (  
    id int not null auto_increment,  
    nome varchar(30) NOT NULL,  
    nascimento date,  
    sexo enum ('M', 'F'),  
    peso decimal (5,2),  
    altura decimal (3,2),  
    nacionalidade varchar(20),  
    primary key (id)  
) default charset = utf8mb4;
```

## #Criar comandos

Se for inserir dados, e a ordem for exatamente do tamanho dos campos não precisa citar os campos, apenas os dados, exemplo:

## #Citando os campos e dados:

use cadastro;

INSERT INTO pessoas

(id, nome, nascimento, sexo, peso, altura, nacionalidade)

VALUES

(default, 'Andre', '1995-05-10', 'M', 75.9, '1.80', 'Portugal');

- SELECT \* from pessoas;

### #Citando apenas os dados:

use cadastro;

INSERT INTO pessoas VALUES

(default, 'Andre', '1995-05-10', 'M', '75.9',  
'1.80', 'Portugal');

- SELECT \* from pessoas;

### #Comando ALTER TABLE para inserir uma coluna na tabela:

Exemplo: Acrescentando a coluna de profissões:

ALTER TABLE

ALTER TABLE pessoas

ADD COLUMN profissão varchar(10);



#Adicionando a coluna depois do nome:

```
ADD COLUMN profissão varchar(10) after  
nome;
```

#Para posicionar uma coluna em PRIMEIRO LUGAR, utilizar o comando FIRST em vez do AFTER, para qualquer outra posição, utilize o AFTER

```
ALTER TABLE pessoas
```

```
ADD COLUMN código int first;
```

#Para alterar a estrutura da definição, ou seja, para modificar o tamanho da profissão. A função MODIFY permite alterar o tipo de primitivo ou a constraint da coluna

```
alter table pessoas
```

```
modify column profissao varchar(20) not  
null default;
```

#Foi incluído “default” no final para não dar erro no NOT NULL, partindo do princípio de que qualquer nova coluna incluída vem com valores nulos.

#Se quiser mudar o nome, utilize no alter table o CHANGE

```
alter table pessoas
```

```
change column profissao prof varchar(20)  
not null default;
```

#Para alterar o nome da tabela:

```
alter table pessoas  
rename to gafanhotos;
```

#Criando uma nova tabela "CURSOS"

```
create table if not exists cursos (  
nome varchar(30) NOT NULL UNIQUE,  
(Para colocar nomes únicos)  
descricao text,  
carga int UNSIGNED, (Para não aparecer  
números negativos)  
total aulas int,  
ano year DEFAULT '2016'  
) DEFAULT CHARSET = UTF8MB4;
```

#O comando IF NOT EXISTS permite que seja criada uma tabela sem correr o risco de apagar outra (sobrescrever).

#Para incluir a coluna idcurso

```
alter table cursos
```

```
add COLUMN idcurso int first;
```

#Alterando para PRIMARY KEY

```
alter table cursos
```

```
add PRIMARY KEY (idcurso);
```

#Para apagar a tabela curso

```
DROP TABLE IF EXISTS cursos;
```

- Manipulando registros, linhas ou tuplas é a mesma coisa.

- Campos ou atributos são as colunas

#Inserindo os dados à tabela cursos

INSERT INTO cursos VALUES

('1', 'HTML4', 'Cursos de HTML5', '40',  
'37','2014'),

('2','Algoritmos','Lógica de  
Programação','20','15','2014'),

('3','Photoshop','Dicas de Photoshop  
CC','10','8','2014'),

- SELECT \* from cursos;

#Alterando um dado:

UPDATE cursos

SET nome = 'HTML5' WHERE idcurso = '1';

#Para alterar 2 dados de uma vez que  
pertencem à mesma linha:

UPDATE cursos

SET nome = 'PHP', ano = '2015' WHERE  
idcurso = '4';

#No próximo exemplo de alteração, vamos alterar 3 registros. Utilizamos o comando LIMIT 1 para evitar que a alteração ocorra em outra linha:

```
UPDATE cursos
```

```
SET nome = 'Java', carga = '40', ano =  
'2015' WHERE idcurso = '5'
```

```
LIMIT 1;
```

#Para apagar linhas:

```
DELETE FROM cursos WHERE idcursos = '8'
```

#Para apagar todos os cursos de 2018:

```
DELETE FROM cursos WHERE ano = '2018'
```

```
LIMIT 3;
```

#Removendo todas as linhas:

```
TRUNCATE TABLE cursos;
```

## phpMyAdmin

É um aplicativo web livre e de código aberto. A partir deste sistema é possível:

- Criar e remover bases de dados;
- Criar, remover e alterar tabelas;
- Inserir, remover e editar campos;
- Executar códigos SQL e manipular campos chaves;

#Para filtrar colunas, substitui-se o \* pelo nome das colunas:

```
SELECT nome, carga, ano FROM cursos  
ORDER BY nome; (Ordem alfabética)
```

#É possível também filtrar em qualquer ordem e ordenar a partir de 2 ou mais colunas:

```
SELECT ano, nome, carga FROM cursos  
ORDER BY ano, nome;
```

#Para filtrar linhas: WHERE ('Onde'):

```
SELECT * FROM cursos  
WHERE ano = '2016' (onde o ano é 2016)  
ORDER BY nome;
```

#Usando outros operadores:

```
SELECT nome, carga, ano FROM cursos  
WHERE ano <= '2015'  
ORDER by ano, nome;
```

**OPERADORES:** = (igual) , >=(maior igual),  
<=(menor igual), <>(ou), !=(diferente)



#Selecionando intervalos: BETWEEN  
(significado: 'entre um valor e outro'):

```
SELECT * from cursos  
WHERE totaulas BETWEEN '20' and '30'  
ORDER by nome;
```

```
SELECT * from cursos  
WHERE ano BETWEEN 2014 and 2016  
ORDER by ano, nome;
```

#Escolhendo intervalos: IN (significado:  
'dados específicos'):

```
SELECT nome, ano FROM cursos  
WHERE ano IN (2014, 2016) (vai mostrar só  
o ano de 2014 e 2016)  
ORDER by ano;
```

## #Combinando testes: WHERE e AND

(A execução só vai ser bem sucedida se a combinação dos valores for correta):

```
SELECT nome, carga, totaulas FROM cursos
```

```
WHERE carga <35 AND totaulas <30
```

```
-SELECT nome, carga, totaulas
```

#Utilizando o OR: (Basta um dos valores ser correto que a execução vai ser bem sucedida, devido ao 'OR'):

```
SELECT nome, carga, totaulas FROM cursos
```

```
WHERE carga <35 AND totaulas <30
```

```
-SELECT nome, carga, totaulas
```

#Usando o operador 'LIKE' % ('coringa'):

```
SELECT * FROM cursos
```

WHERE nome LIKE 'P%'; (Mostra os cursos  
começados pela letra 'P')

#Usando o operador 'LIKE' % ('coringa'):

```
SELECT * FROM CURSOS
```

WHERE nome LIKE '%P'; (Mostra os cursos  
terminados pela letra 'P')

#Usando o operador 'LIKE' % ('coringa'):

```
SELECT * FROM cursos
```

WHERE nome like '%P%'; (Mostra os cursos  
que tenham qualquer letra com 'P')

#DISTINCT (Serve para apresentar determinada lista de forma não repetitiva):

```
SELECT DISTINCT nacionalidade FROM  
gafanhotos; (Não repete as nacionalidades)
```

#AGREGAÇÕES (Servem para selecionar ou totalizar alguma coisa):

```
SELECT COUNT(*) FROM cursos;
```

Total de cursos: 30

- Quero saber quantos cursos têm mais de 40 horas, no lugar do SELECT \* troca-se por SELECT COUNT(\*):

```
-SELECT COUNT(*) FROM cursos WHERE  
carga > 40;
```

Total de cursos: 6

### #Função max e min (máximo, mínimo)

```
SELECT MAX(carga) FROM cursos;
```

```
SELECT MIN(carga) FROM cursos;
```

### - Em 2015 qual curso teve mais aulas?

```
SELECT MAX(totaulas) FROM cursos  
WHERE ano = '2015';
```

### #Função SUM (somar)

```
SELECT SUM(totaulas) FROM cursos  
WHERE ano = '2016';
```

### #Função AVG (tirar a média)

```
SELECT AVG(totaulas) FROM cursos WHERE  
ano = '2016';
```

## GROUP BY – AGRUPANDO E AGREGANDO (AGRUPAR E SOMAR)

#Serve para contar quantas vezes um registo de um determinado campo aparece

```
SELECT carga, COUNT(nome) FROM cursos  
GROUP by carga;
```

#Mostrar quem tem o contador maior que 3, agrupando em cargas horárias

```
SELECT carga, COUNT(nome) FROM cursos  
GROUP by carga  
HAVING COUNT(nome) > 3;
```

#Saber a média da carga

```
SELECT AVG(carga) from CURSOS;
```

Total: 35.6

## EXERCÍCIOS

#Faça uma lista com profissões e seus respectivos quantitativos

```
SELECT profissao, COUNT(*) FROM cursos  
GROUP by profissao  
ORDER by COUNT(*)
```

#Quantos gafanhotos homens e quantas mulheres nasceram após 01-01-2005?

```
SELECT sexo, COUNT(*) FROM gafanhotos  
WHERE nascimento > '2005-01-01'  
GROUP by sexo;
```

Uma lista com os gafanhotos que nasceram fora do Brasil, mostrando qual o país de origem e o total de pessoas nascidas lá. Mostre os países que tiverem mais de 3 gafanhotos com essa nacionalidade.

```
SELECT nacionalidade, COUNT(*) FROM  
gafanhotos
```

```
WHERE nacionalidade != Brasil
```

```
GROUP by nacionalidade
```

```
HAVING COUNT (nacionalidade) = 3;
```



#Uma lista agrupada pela altura dos gafanhotos, mostrando quantas pessoas pesam mais de 100KG e que está acima da média da altura de todos os cadastros

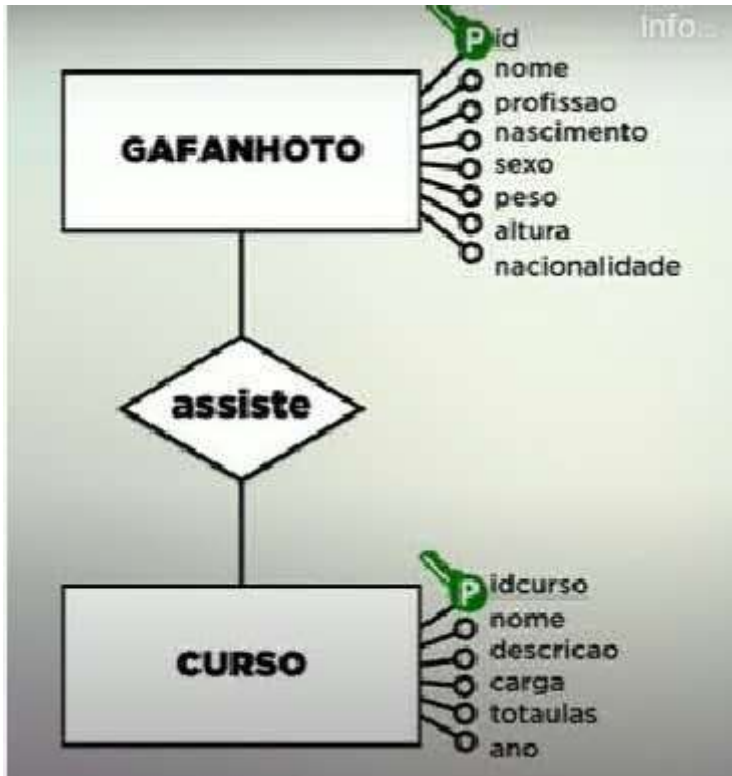
```
SELECT altura, COUNT(*) FROM gafanhotos
```

```
WHERE peso > 100
```

```
GROUP by sexo
```

```
HAVING COUNT(SELECT AVG(altura) FROM  
gafanhotos)
```

## MODELO RELACIONAL (N:N / 1:1 / 1:N)

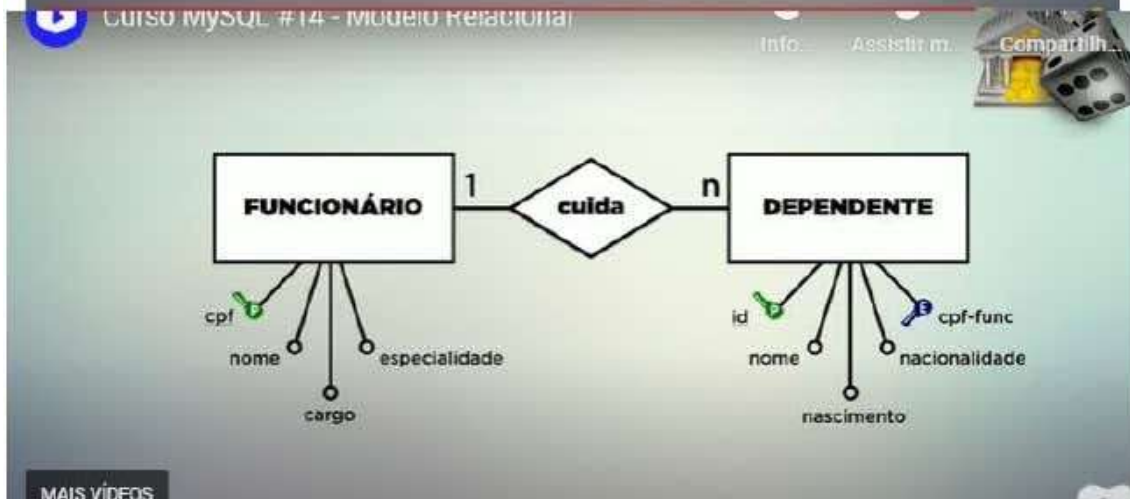
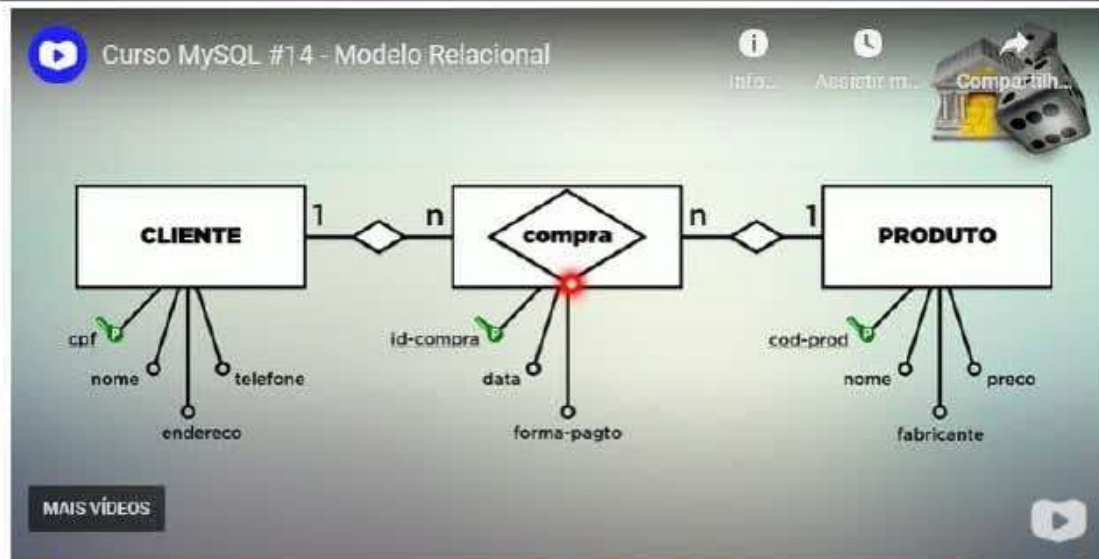
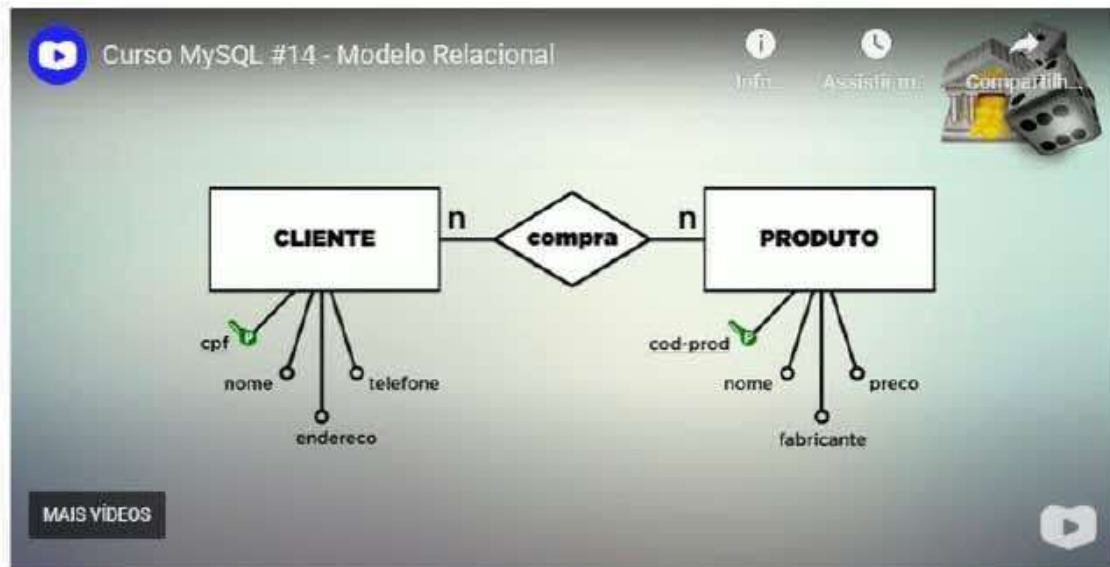


**Entidades:** Gafanhoto e Curso

**Relacionamento:** Assiste

**Cardinalidade:** Muitos-para-Muitos

-Num relacionamento de Muitos-para-Muitos o relacionamento transforma-se em uma entidade com alguns atributos.



A) N:N | B) 1:1 | C) 1:N

- A) **Relação N:N** (Muitos-para-Muitos) é tratada por meio de uma tabela de junção que conecta as chaves primárias das tabelas envolvidas, permitindo o mapeamento dos relacionamentos entre elas.
- B) **Relação 1:1** (Um-para-Um). Decide-se qual é a entidade dominante. Pega-se na chave primária de quem não é dominante e fica como chave estrangeira na entidade dominante.
- C) **Relação 1:N** (Um-para-Muitos). Leva-se chave primária da entidade 1 para chave estrangeira da entidade N.

## #Primary Key (Chave Primária)

Identificador único de um registo na tabela. A chave primária é necessária para não haver registos duplicados e para se relacionarem. Se não houver chave primária não há relação.

## #Foreing Key (Chave Estrangeira)

É a chave primária de algum lugar que veio para outro lugar.

**InnoDB** é uma ENGINE, ou seja, uma máquina de criação de tabelas com características. A principal é suportar chaves estrangeiras.

**Transação** é toda a ação que um banco de dados pode executar.

## As 4 regras da transação - ACID

### A – Atomocidade

Garante que a transição seja realizada por completo ou não seja realizada de forma alguma.

### C – Consistência

Garante que os dados permaneçam consistentes antes e após a execução da transação.

### I – Isolamento

Garante que as transações não interfiram umas nas outras.

### D – Durabilidade

Garante que as alterações feitas em uma transação sejam permanentes e não se percam, mesmo em caso de falha do sistema.

#Adicionando a chave estrangeira na tabela gafanhotos

```
ALTER TABLE gafanhotos
```

```
ADD COLUMN cursopreferido int,
```

#Convertendo a coluna em chave estrangeira e referenciado a idcursos

```
ALTER TABLE gafanhotos
```

```
ADD FOREIGN KEY (cursopreferido)
```

```
REFERENCES cursos(idcurso);
```

#Adicionando os cursos preferidos aos gafanhotos

```
UPDATE gafanhotos SET cursopreferido =  
'6' WHERE id = '1';
```

Para evitar o trabalho de fazer cada UPDATE como linha de código, digitamos diretamente na tabela de gafanhotos.

#Ver o nome do curso e do ano na tabela gafanhotos

```
SELECT gafanhotos.nome,  
gafanhotos.cursopreferido, cursos.nome,  
cursos.ano from GAFANHOTS join cursos  
ON cursos.idcurso (chave primária) =  
gafanhotos.cursopreferido (chave  
estrangeira)
```



The screenshot displays a database IDE with two windows. The top window shows a sequence of SQL queries: a select from 'gafanhotos', a select from 'cursos', and a join query connecting the two tables on 'cursos.idcurso = gafanhotos.cursopreferido'. The bottom window shows the 'Result Grid' with a table of data. The table has four columns: 'nome', 'cursopreferido', 'nome', and 'ano'. The data rows list names like Daniel Moraes, Julia Nascimento, Emerson Gabriel, etc., along with their preferred course IDs and the course names and years.

```
1 * select nome, cursopreferido from gafanhotos;
2
3 * select nome, ano from cursos;
4
5 * select gafanhotos.nome, gafanhotos.cursopreferido, cursos.nome, cursos.ano
6   from gafanhotos join cursos
7   on cursos.idcurso = gafanhotos.cursopreferido;
```

nome	cursopreferido	nome	ano
Daniel Moraes	6	MySQL	2016
Julia Nascimento	22	Premiere	2017
Emerson Gabriel	12	C++	2017
Lucas Damasceno	7	Word	2016
Lela Martins	1	HTML5	2014
Letícia Neves	8	Python	2017
Janaína Couto	4	PHP	2015
Carlson Rosa	5	Java	2015
Jackson Teles	3	Photoshop5	2014
Danielo Araujo	30	PHP4	2010
Andressa Delfino	22	Premiere	2017

```
2
3 * select nome, ano from cursos;
4
5 * select * from gafanhotos;
6
7 * select g.nome, c.nome, c.ano
8   from gafanhotos as g inner join cursos as c
9   on c.idcurso = g.cursopreferido
10  order by g.nome;
```

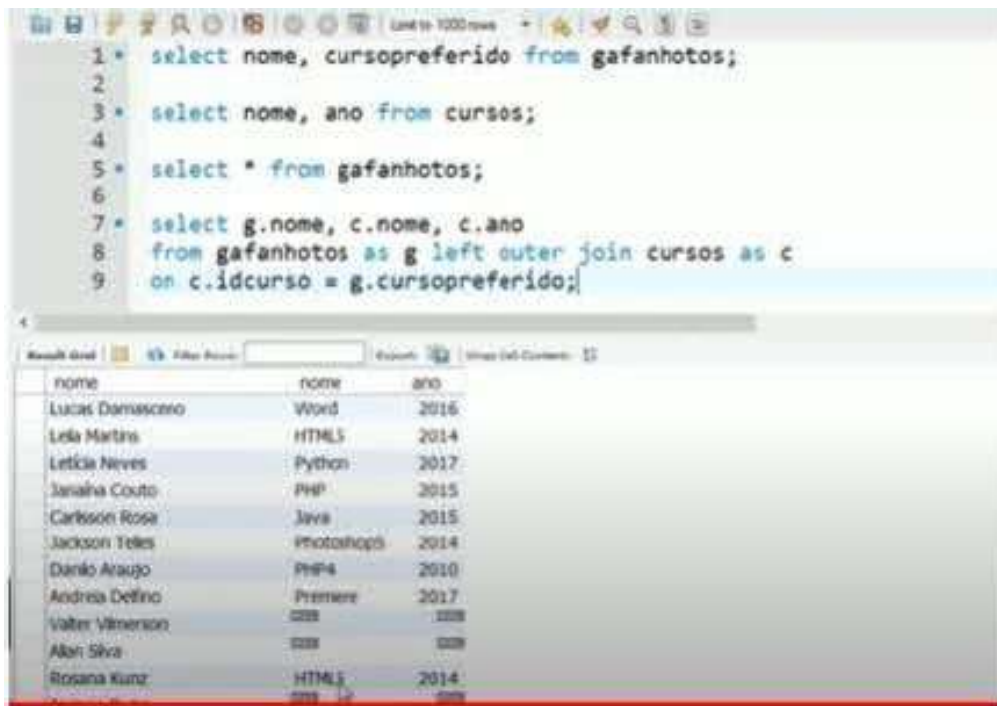
#O InnerJoin ou Join faz a ligação apenas dos que não tem valores NULL

SELECT gafanhotos.nome, cursos.nome, cursos.ano from gafanhotos join cursos

ON cursos.idcurso =  
gafanhotos.cursopreferido;

## #LEFT OUTER JOIN ou LEFT JOIN

Mantém todos os dados da tabela à **esquerda** e apresenta os NULOS na tabela à **direita**.



The screenshot shows a SQL query editor with the following code:

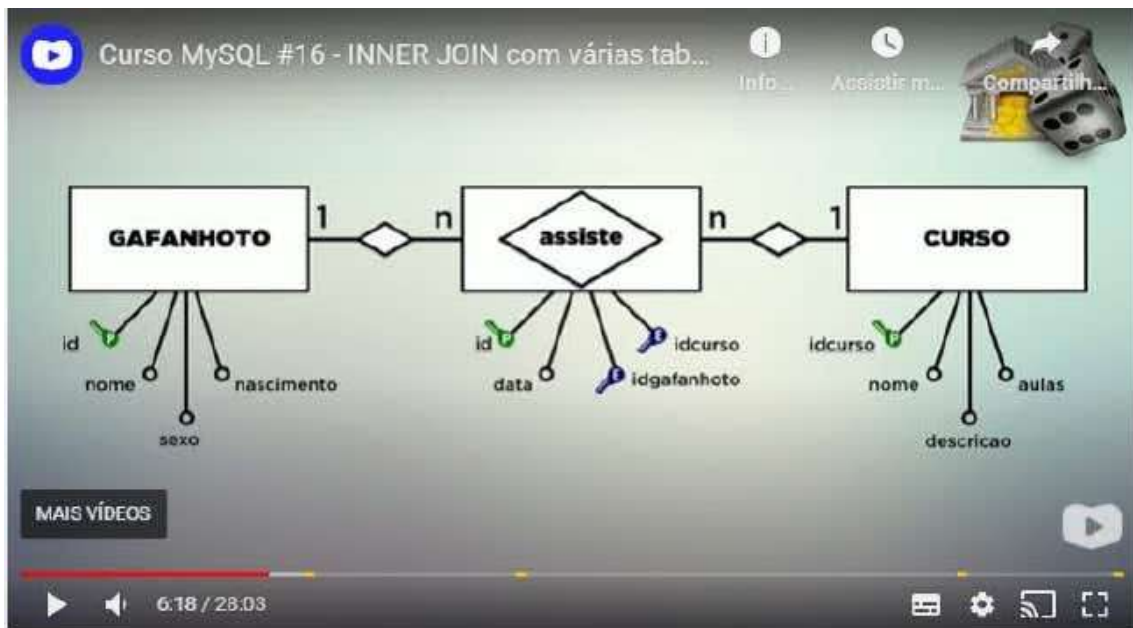
```
1 * select nome, cursopreferido from gafanhotos;
2
3 * select nome, ano from cursos;
4
5 * select * from gafanhotos;
6
7 * select g.nome, c.nome, c.ano
8   from gafanhotos as g left outer join cursos as c
9   on c.idcurso = g.cursopreferido;
```

Below the query, the results are displayed in a table with three columns: nome, nome, and ano. The data is as follows:

nome	nome	ano
Lucas Damasceno	Word	2016
Lela Martins	HTML5	2014
Leticia Neves	Python	2017
Janaína Couto	PHP	2015
Carlosson Rosa	Java	2015
Jackson Telles	Photoshop	2014
Danió Araujo	Prep4	2010
Andressa Defino	Premiere	2017
Valter Vilmerison	css	css
Alan Silva	css	css
Rosana Kunz	HTML5	2014
	css	css

## #RIGHT OUTER JOIN ou RIGHT JOIN

Mantém todos os dados da tabela à **direita** e apresenta os NULOS na tabela à **esquerda**.



```

12
13 • select g.nome, c.nome, c.ano
14   from gafanhotos as g RIGHT OUTER join cursos as c
15   ON c.idcurso = g.cursopreferido
16   order by g.nome;
17
18 • select nome, ano from cursos;
19

```

Result Grid

nome	nome	ano
NULL	JavaScript	2017
NULL	Redes	2016
NULL	WordPress	2019
NULL	PHP7	2020
Andreia Delfino	Premiere	2017
Carlisson Rosa	Java	2015
Daniel Moreira	MySQL	2016
Danilo Araujo	PHP4	2010
Emerson Gabriel	C++	2017
Jackson Teles	Photoshop5	2014
Jenaina Couto	PHP	2015
Leila Martins	HTML5	2014
Leticia Neves	Python	2017
Lucas Damasceno	Word	2016
Rosana Kunz	HTML5	2014

#Criando a tabela ASSISTE com as duas  
chaves estrangeiras

create table gafanhoto\_assiste\_curso (
 id int NOT NULL AUTO\_INCREMENT,
 data date,

```
idgafanhoto int,  
idcurso int,  
PRIMARY KEY (id),  
FOREIGN KEY (idgafanhoto)  
REFERENCES gafanhoto(id),  
FOREIGN KEY (idcurso)  
REFERENCES cursos(idcurso)  
) DEFAULT CHARSET = utf8mb4;
```

JOIN com várias tabelas

#Vamos juntar a tabela 'Gafanhoto' com a  
tabela 'Assiste'

g = gafanhotos

a = gafanhoto\_assiste\_curso

c = curso

```
SELECT g.nome, idcurso from gafanhotos g  
JOIN gafanhoto_assiste_curso a  
ON g.id = a.idgafanhoto;
```



#Vamos juntar a tabela 'Curso' com a tabela 'Assiste' (Para saber o nome do curso)

g = gafanhotos

a = gafanhoto\_assiste\_curso

c = curso

SELECT g.nome, c.nome from gafanhotos g

JOIN gafanhoto\_assiste\_curso a

ON g.id = a.idgafanhotos (chave primária  
com chave estrangeira)

JOIN curso c

ON c.idcurso = a.idcurso (chave primária  
com chave estrangeira)

