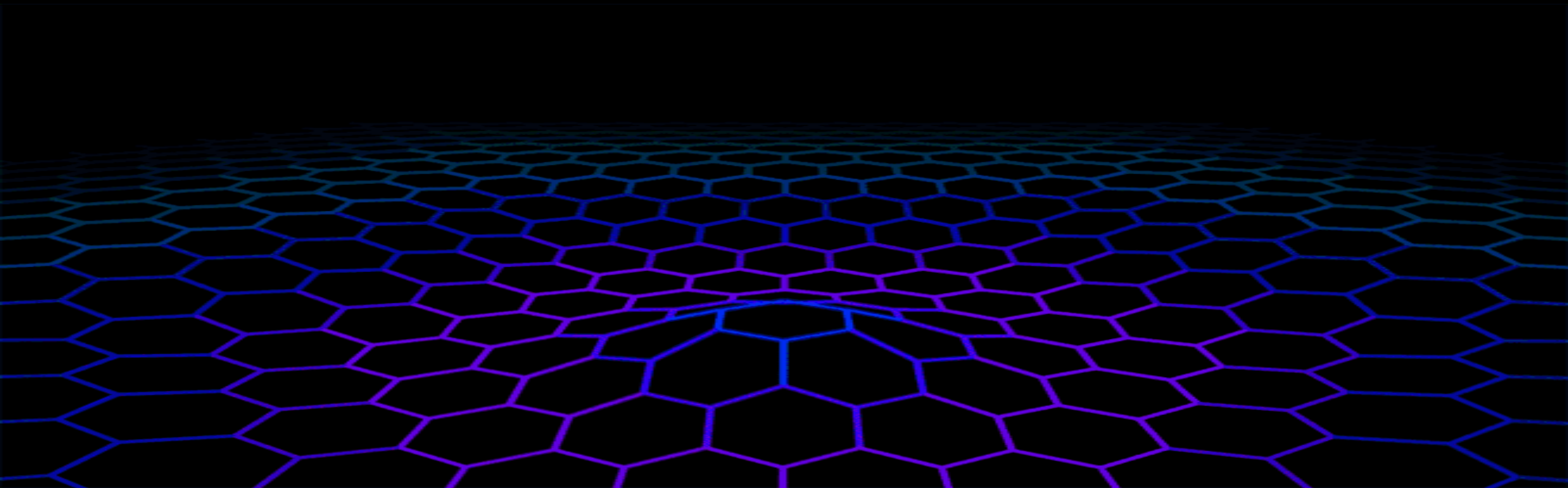


# Banco de Dados I



Resolução dos exercícios da aula anterior

```
-- a. Apuração do público por município
SELECT SUM(sessao.publico) AS Publico,
       cidade.nome        AS Cidade
FROM   sessao
       INNER JOIN sala    ON sessao.sala_id    = sala.id
       INNER JOIN cinema  ON sala.cinema_id    = cinema.id
       INNER JOIN cidade  ON cinema.cidade_id  = cidade.id
-- WHERE sessao.data = "2022-11-18"
GROUP BY cidade.nome
ORDER BY cidade.nome;
```

```
-- b.  Apuração do público por cinema
SELECT SUM(sessao.publico) AS Qtde,
       cinema.nomefantasia AS Cinema
FROM   sessao
       INNER JOIN sala    ON sessao.sala_id = sala.id
       INNER JOIN cinema  ON sala.cinema_id = cinema.id
-- WHERE sessao.data = "2022-11-18"
GROUP BY cinema.nomefantasia
ORDER BY cinema.nomefantasia;
```

```
-- c.  Apuração do público por sessão de cada cinema
SELECT cinema.nomefantasia AS Cinema,
       sessao.horainicio    AS Horario,
       SUM(sessao.publico)  AS Qtde
FROM   sessao
       INNER JOIN sala      ON sessao.sala_id = sala.id
       INNER JOIN cinema    ON cinema.id      = sala.cinema_id
GROUP  BY sessao.horainicio,
          cinema.nomefantasia
ORDER  BY cinema.nomefantasia,
          sessao.horainicio;
```

/\* d. Dado um determinado ator, sejam localizados todos os cinemas onde estão em cartaz os filmes em que este ator atua \*/

```
SELECT cinema.nomefantasia AS Cinema,
       filme.titulooriginal AS Filme,
       ator.nome           AS Ator
FROM   cinema
       INNER JOIN sala      ON cinema.id = sala.cinema_id
       INNER JOIN sessao    ON sala.id   = sessao.sala_id
       INNER JOIN filme     ON filme.id  = sessao.filme_id
       INNER JOIN elenco    ON filme.id  = elenco.filme_id
       INNER JOIN ator      ON ator.id   = elenco.ator_id
WHERE  ator.nome = 'Jason Clarke'
--     AND sessao.data = CURDATE();
      AND sessao.data = "2022-11-17";
```

```
-- e. Em quais cinemas está sendo exibido um determinado gênero de filme
SELECT DISTINCT cidade.nome           AS Cidade,
               cinema.nomefantasia AS Cinema,
               filme.titulooriginal AS Filme,
               genero.nome           AS Genero

FROM   cidade
       INNER JOIN cinema      ON cidade.id = cinema.cidade_id
       INNER JOIN sala        ON cinema.id = sala.cinema_id
       INNER JOIN sessao      ON sala.id   = sessao.sala_id
       INNER JOIN filme       ON filme.id  = sessao.filme_id
       INNER JOIN genero_filme ON filme.id  = genero_filme.filme_id
       INNER JOIN genero      ON genero.id  = genero_filme.genero_id

WHERE  genero.nome = 'Drama'
--     AND sessao.data = CURDATE();
      AND sessao.data = "2022-11-17";
```

```
-- f. Em quais cinemas estão sendo exibidos filmes nacionais
SELECT DISTINCT cidade.nome          AS Cidade,
               cinema.nomefantasia AS Cinema,
               filme.titulooriginal AS Filme

FROM cidade
      INNER JOIN cinema ON cidade.id = cinema.cidade_id
      INNER JOIN sala   ON cinema.id = sala.cinema_id
      INNER JOIN sessao ON sala.id   = sessao.sala_id
      INNER JOIN filme  ON filme.id  = sessao.filme_id
      INNER JOIN pais   ON pais.id   = filme.pais_id

WHERE pais.nome = 'Brasil'
-- AND sessao.data = CURDATE();
AND sessao.data = "2022-11-17";
```



+ SQL

# SubConsultas / Consultas Aninhadas

Pode ser implementada com o uso de blocos SELECT-FROM-WHERE dentro da cláusula WHERE de outra consulta.

Pode ser implementada com o operador IN, que compara um valor com um conjunto de valores ao mesmo tempo.

Exemplo 1  
=>

```
SELECT filme.tituloOriginal, genero.nome
FROM filme, genero
WHERE
    filme.idGenero = genero.idGenero AND
    genero.nome IN('Suspense', 'Comédia', 'Drama', 'Terror')
ORDER BY genero.idGenero;
```

# SubConsultas / Consultas Aninhadas

Pode ser implementada com o uso de blocos SELECT-FROM-WHERE dentro da cláusula WHERE de outra consulta.

Pode ser implementada com o operador IN, que compara um valor com um conjunto de valores ao mesmo tempo.

Exemplo 2  
=>

```
SELECT filme.tituloOriginal, genero.nome
FROM filme, genero
WHERE
    filme.idGenero = genero.idGenero AND
    genero.nome IN (SELECT genero.nome FROM genero)
ORDER BY genero.idGenero;
```

# Verificando com a estrutura de uma tabela foi criada

```
SHOW CREATE TABLE ator;
```

```
+-----+-----+
| ator | CREATE TABLE ator (
  id int NOT NULL AUTO_INCREMENT,
  pais_id int NOT NULL,
  nome varchar(45) NOT NULL,
  data_nas date DEFAULT NULL,
  PRIMARY KEY (id),
  KEY fk_ator_pais (pais_id),
  CONSTRAINT fk_ator_pais FOREIGN KEY (pais_id) REFERENCES pais (id)
) ENGINE=InnoDB AUTO_INCREMENT=1
+-----+-----+
```

# GROUP BY

Agrupar os resultados por uma coluna selecionada

```
SELECT pais_id, MAX(duracao)
FROM filme
GROUP BY pais_id;
```

Maior de cada grupo

duracao	id	tituloOriginal	pais_id
138	1	Flight	1
98	3	The Sessions	2
164	4	Django Unchained	2
144	5	The Master	2
102	6	Killer Joe	2
157	7	Zero Dark Thirty	2
105	9	The Cabin in The Woods	2
131	2	Anna Karenina	3
127	8	Amour	4

pais_id	MAX(duracao)
1	138
2	164
3	131
4	127

SAÍDA

# SUM()

Realiza um somatório dos valores por coluna.

```
SELECT SUM(duracao)
FROM filme;
```

+	_____	+
	SUM(duracao)	
+	_____	+
	1166	
+	_____	+

# AVG()

Realiza o cálculo da média dos valores por coluna.

```
SELECT AVG(duracao)
FROM filme;
```

+	_____	+
	AVG(duracao)	
+	_____	+
	129.5556	
+	_____	+

# MAX()

Determina o maior valor em uma coluna.

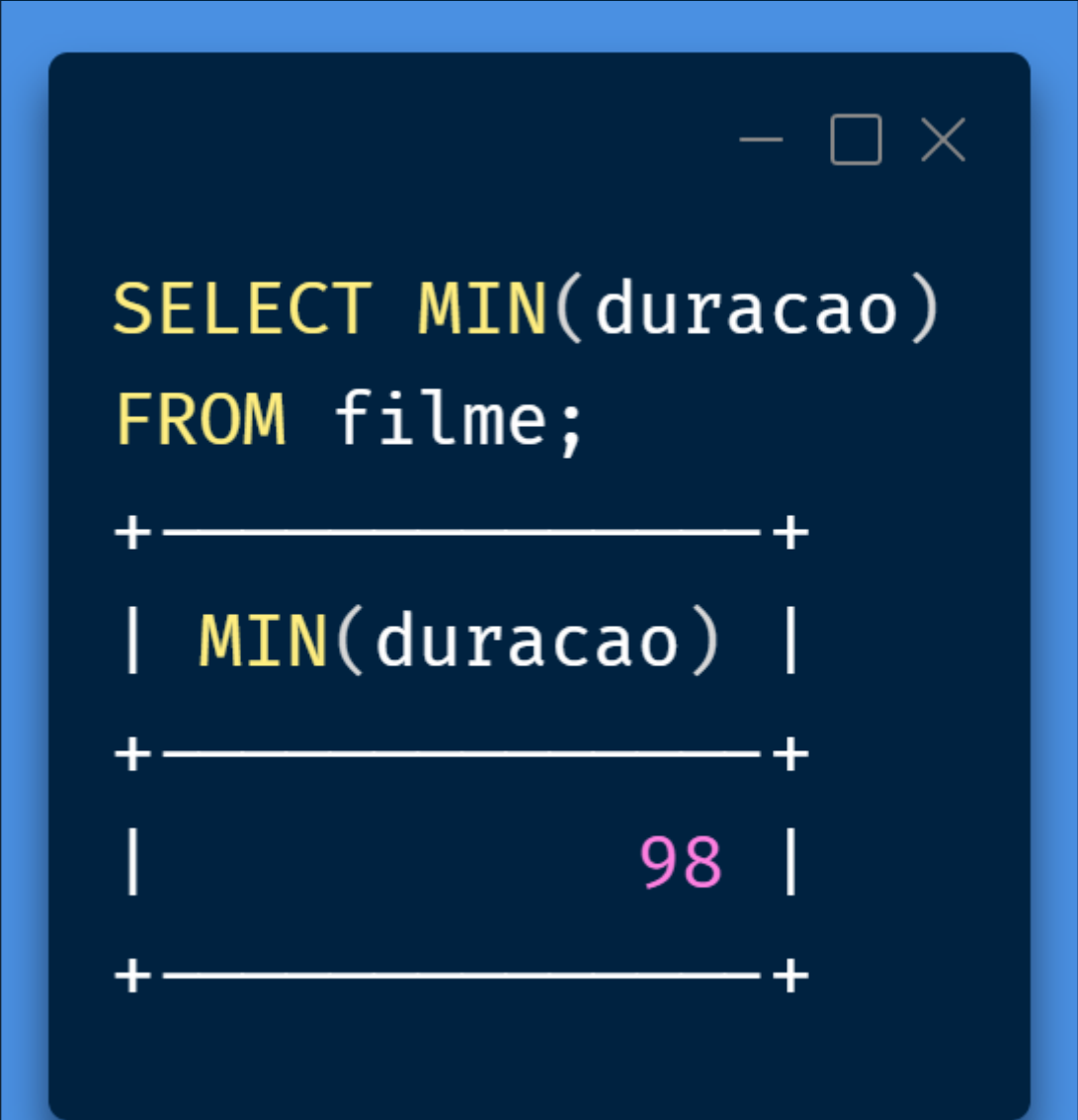
```
SELECT MAX(duracao)
FROM filme;
```

+	_____	+
	MAX(duracao)	
+	_____	+
	164	
+	_____	+



# MIN()

Determina o menor valor em uma coluna.



```
SELECT MIN(duracao)
FROM filme;
```

MIN(duracao)
98

# LIMIT número

Restringe os resultados de uma consulta para uma quantidade de valores explícita no argumento numero.

```
SELECT *  
FROM filme LIMIT 5;
```

id	pais_id	diretor_id	tituloOriginal	tituloPortugues	duracao
1	1	3	Flight	O Voo	138
2	3	3	Anna Karenina	Anna Karenina	131
3	2	1	The Sessions	As Sessões	98
4	2	7	Django Unchained	Django Livre	164
5	2	3	The Master	O Mestre	144

# COUNT(coluna)

Realiza a contagem de itens.

```
SELECT COUNT(*)  
FROM filme  
WHERE pais_id = 2;
```

+	-----	+
	COUNT(*)	
+	-----	+
	6	
+	-----	+

# RIGHT(campo, quant)

Selecione uma quantidade específica de caracteres de uma coluna (a partir da direita).

```
SELECT RIGHT(tituloOriginal,10)
FROM filme;
```

```
+-----+  
| RIGHT(tituloOriginal,10) |  
+-----+  
| Flight                    |  
| a Karenina               |  
| e Sessions               |  
|   Unchained              |  
| The Master                |  
| Killer Joe                |  
| ark Thirty                |  
| Amour                     |  
|   The Woods               |  
+-----+
```

# LEFT(campo, quant)

Seleciona uma quantidade específica de caracteres de uma coluna. (a partir da esquerda)

```
SELECT LEFT(tituloOriginal,10)
FROM filme;
```

+	-----	+
	LEFT(tituloOriginal,10)	
+	-----	+
	Flight	
	Anna Karen	
	The Sessio	
	Django Unc	
	The Master	
	Killer Joe	
	Zero Dark	
	Amour	
	The Cabin	
+	-----	+

# UPPER(campo) / UCASE(campo)

Aplica **CAIXA ALTA** nos caracteres de um campo.

```
SELECT UPPER(nome)
FROM genero;
```

+	_____	+
	UPPER(nome)	
+	_____	+
	COMÉDIA	
	FICÇÃO	
	DRAMA	
	AÇÃO	
	SUSPENSE	
	TERROR	
	FAROESTE	
	AVENTURA	
+	_____	+

# LOWER(campo) / LCASE(campo)

Aplica **caixa baixa** nos caracteres de um campo.

```
SELECT lcase(nome)
FROM genero;
```

lcase(nome)
comédia
ficção
drama
ação
suspense
terror
faroeste
aventura

# REVERSE(campo)

Inverte o texto do campo selecionado na consulta.

```
SELECT REVERSE(nome)  
FROM genero;
```

+	_____	+
	REVERSE(nome)	
+	_____	+
	aidémoC	
	oãçciF	
	amarD	
	oãçA	
	esnepsuS	
	rorreT	
	etseoraF	
	arutnevA	
+	_____	+



# LTRIM(campo) / RTRIM(campo)

Remove espaços em branco à esquerda ou à direita do campo selecionado.

```
SELECT LTRIM(nome)  
FROM genero;
```

+	_____	+
	LTRIM(nome)	
+	_____	+
	Comédia	
	Ficção	
	Drama	
	Ação	
	Suspense	
	Terror	
	Faroeste	
	Aventura	
+	_____	+

# LENGTH(campo)

Retorna a quantidade de caracteres do campo selecionado.

```
SELECT nome, LENGTH(nome)
FROM genero;
```

+	-----+	-----+
	nome	LENGTH(nome)
+	-----+	-----+
	Comédia	8
	Ficção	8
	Drama	5
	Ação	6
	Suspense	8
	Terror	6
	Faroeste	8
	Aventura	8
+	-----+	-----+

**FICA!**  
**& DICA!**

#1

# #1 DELETE COM WHERE

# #1 DELETE COM WHERE



```
DROP TABLE IF EXISTS produto;  
CREATE TABLE IF NOT EXISTS produto  
(  
    id          INT(8) NOT NULL AUTO_INCREMENT,  
    nome        VARCHAR(100),  
    valor       DECIMAL(10, 2),  
    estoque     DECIMAL(10, 2),  
    ativo       BOOLEAN,  
    PRIMARY KEY (id)  
);
```

# #1 DELETE COM WHERE



```
INSERT INTO produto VALUES (1,"Livro S", 99.99, 12, TRUE);  
INSERT INTO produto VALUES (2,"Livro E", 89.99, 0, TRUE);  
INSERT INTO produto VALUES (3,"Livro N", 49.99, 3, TRUE);  
INSERT INTO produto VALUES (4,"Livro A", 56.38, 2, TRUE);  
INSERT INTO produto VALUES (5,"Livro C", 32.41, 0, TRUE);
```

# #1 DELETE COM WHERE

Sempre que escrevemos um **DELETE** no console do banco de dados vale a pena começar com o **WHERE**.

A dica também serve para o comando **UPDATE**.



# #1 Iniciar pelo WHERE



```
WHERE estoque = 0;
```

# #1 Fazer um SELECT



```
SELECT * FROM produto WHERE estoque = 0;
```

# #1 Conferir o resultado



```
SELECT * FROM produto WHERE estoque = 0;
```

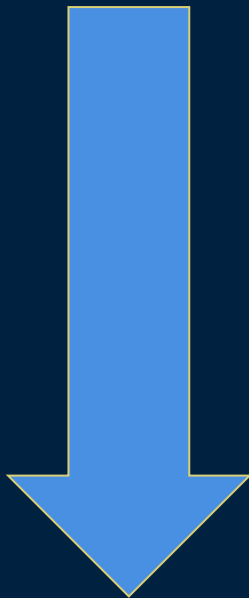
id	nome	valor	estoque	ativo
2	Livro E	89.99	0.00	1
5	Livro C	32.41	0.00	1

2 rows in set (0.00 sec)

# #1 Substituir o SELECT por DELETE



```
SELECT * FROM produto WHERE estoque = 0;
```



```
DELETE FROM produto WHERE estoque = 0;
```

Assim você evita que um [ENTER] acidental apague sua tabela toda.

#2

## #2 Promoção com Livro por 2 Reais? Ou ainda -2 Reais?

Alguns campos devem ser validados declarativamente, como o preço de um produto;

## #2 Promoção com Livro por 2 Reais? Ou ainda -2 Reais?

Alguns campos devem ser validados declarativamente, como o preço de um produto;

○ ○ ○

```
DROP TABLE IF EXISTS produto;
CREATE TABLE IF NOT EXISTS produto
(
    id      INT(8) NOT NULL AUTO_INCREMENT,
    nome    VARCHAR(100),
    valor   DECIMAL(10, 2),
    estoque DECIMAL(10, 2),
    ativo   BOOLEAN,
    PRIMARY KEY (id)
);
```

## #2 Promoção com Livro por 2 Reais? Ou ainda -2 Reais?

Alguns campos devem ser validados declarativamente, como o preço de um produto;

○ ○ ○

```
DROP TABLE IF EXISTS produto;
CREATE TABLE IF NOT EXISTS produto
(
    id      INT(8) NOT NULL AUTO_INCREMENT,
    nome    VARCHAR(100),
    valor   DECIMAL(10, 2),
    estoque DECIMAL(10, 2),
    ativo   BOOLEAN,
    PRIMARY KEY (id)
);

INSERT INTO produto VALUES (8,"Livro X",-2.00, 12, TRUE);
-- Query OK, 1 row affected (0.00 sec)
```



## #2 Promoção com Livro por 2 Reais? Ou ainda -2 Reais?

Alguns campos devem ser validados declarativamente, como o preço de um produto;

Inserindo linha com preço negativo :(  
Problema! Eu queria integridade dos meus dados.  
Queria que ele checasse por valores inválidos.

```
DROP TABLE IF EXISTS produto;  
CREATE TABLE IF NOT EXISTS produto  
(  
    id      INT(8) NOT NULL AUTO_INCREMENT,  
    nome    VARCHAR(100),  
    valor   DECIMAL(10, 2),  
    estoque DECIMAL(10, 2),  
    ativo   BOOLEAN,  
    PRIMARY KEY (id)  
);  
  
INSERT INTO produto VALUES (8,"Livro X",-2.00, 12, TRUE);  
-- Query OK, 1 row affected (0.00 sec)
```

## #2 Manda o banco checar o valor

## #2 Manda o banco checar o valor

○ ○ ○

```
DROP TABLE IF EXISTS produto;
CREATE TABLE IF NOT EXISTS produto
(
    id      INT(8) NOT NULL AUTO_INCREMENT,
    nome    VARCHAR(100),
    valor   DECIMAL(10, 2),
    estoque DECIMAL(10, 2),
    ativo   BOOLEAN,
    PRIMARY KEY (id),
    CHECK(valor > 0)
);
```

## #2 Manda o banco checar o valor

○ ○ ○

```
DROP TABLE IF EXISTS produto;
CREATE TABLE IF NOT EXISTS produto
(
    id      INT(8) NOT NULL AUTO_INCREMENT,
    nome    VARCHAR(100),
    valor   DECIMAL(10, 2),
    estoque DECIMAL(10, 2),
    ativo   BOOLEAN,
    PRIMARY KEY (id),
    CHECK(valor > 0)
);

INSERT INTO produto VALUES (8, "Livro X", -2.00, 12, TRUE);
/* ERROR 3819 (HY000):
Check constraint 'produto_chk_1' is violated.*/
```

## #2 Manda o banco checar o valor

○ ○ ○

```
DROP TABLE IF EXISTS produto;
CREATE TABLE IF NOT EXISTS produto
(
    id      INT(8) NOT NULL AUTO_INCREMENT,
    nome    VARCHAR(100),
    valor   DECIMAL(10, 2),
    estoque DECIMAL(10, 2),
    ativo   BOOLEAN,
    PRIMARY KEY (id),
    CHECK(valor > 0)
);

INSERT INTO produto VALUES (8, "Livro X", -2.00, 12, TRUE);
/* ERROR 3819 (HY000):
Check constraint 'produto_chk_1' is violated.*/
```

Ainda bem que o MySQL e o MariaDB já suportam **CHECK**  
(da mesma forma que Oracle, PostgreSQL, SQL Server, ...)

#3



```
DROP TABLE IF EXISTS produto;  
CREATE TABLE IF NOT EXISTS produto  
(  
    id        INT(8) NOT NULL AUTO_INCREMENT,  
    nome      VARCHAR(100),  
    valor     DECIMAL(10, 2),  
    estoque  DECIMAL(10, 2),  
    ativo     BOOLEAN,  
    PRIMARY KEY (id),  
    CHECK(valor > 0)  
);
```



```
INSERT INTO produto VALUES (10,"Livro A",1.09, 10, TRUE);
INSERT INTO produto VALUES (11,"Livro B",2000.56, 12, TRUE);
INSERT INTO produto VALUES (12,"Livro C",3000.99, 2, FALSE);
INSERT INTO produto VALUES (13,"Livro D",45.99, 9, TRUE);
INSERT INTO produto VALUES (14,"Livro E",2.99, 11, FALSE);
INSERT INTO produto VALUES (15,"Livro F",80.19, 10, FALSE);
INSERT INTO produto VALUES (16,"Livro G",39.21, 10, TRUE);
INSERT INTO produto VALUES (17,"Livro H",68.23, 8, TRUE);
INSERT INTO produto VALUES (19,"Livro I",2.00, 3, TRUE);
INSERT INTO produto VALUES (20,"Livro J",2500.38, 4, FALSE);
```



# #3

## Desafio:

Gostaria de trazer os livros ativos cujos preços tem algo de estranho: são muito baratos ( $< 10$ ) OU muito caros ( $> 1000$ )

Qual o problema da query?



```
SELECT *  
FROM produto  
WHERE ativo = true  
      AND valor < 10  
      OR valor > 1000;
```

# #3

## Desafio:

Gostaria de trazer os livros ativos cujos preços tem algo de estranho: são muito baratos ( $< 10$ ) OU muito caros ( $> 1000$ )

Qual o problema da query?

Query bonita escrita, executada, resultado "errado".  
Quem nunca teve uma query respondendo um monte de coisa que não foi pedido?

○ ○ ○

```
SELECT *  
FROM produto  
WHERE ativo = true  
      AND valor < 10  
      OR valor > 1000;
```

id	nome	valor	estoque	ativo
10	Livro A	1.09	10.00	1
11	Livro B	2000.56	12.00	1
12	Livro C	3000.99	2.00	0
19	Livro I	2.00	3.00	1
20	Livro J	2500.38	4.00	0

5 rows in set (0.00 sec)

# #3

## Desafio:

Gostaria de trazer os livros ativos cujos preços tem algo de estranho: são muito baratos ( $< 10$ ) OU muito caros ( $> 1000$ )

Qual o problema da query?

Query bonita escrita, executada, resultado "errado".  
Quem nunca teve uma query respondendo um monte de coisa que não foi pedido?

Problema: O MySQL poderá retornar livros inativos.

○ ○ ○

```
SELECT *  
FROM produto  
WHERE ativo = true  
      AND valor < 10  
      OR valor > 1000;
```

id	nome	valor	estoque	ativo
10	Livro A	1.09	10.00	1
11	Livro B	2000.56	12.00	1
12	Livro C	3000.99	2.00	0
19	Livro I	2.00	3.00	1
20	Livro J	2500.38	4.00	0

5 rows in set (0.00 sec)

# #3 Parenteses

○ ○ ○

```
SELECT *  
FROM produto  
WHERE ativo = true  
      AND (valor < 10  
          OR valor > 1000);
```

# #3 Parenteses

○ ○ ○

```
SELECT *  
FROM produto  
WHERE ativo = true  
      AND (valor < 10  
           OR valor > 1000);
```

id	nome	valor	estoque	ativo
10	Livro A	1.09	10.00	1
11	Livro B	2000.56	12.00	1
19	Livro I	2.00	3.00	1

3 rows in set (0.00 sec)

#4

## #4 Que tal criar uma tabela a partir de outra?

## #4 Que tal criar uma tabela a partir de outra?



```
CREATE TABLE produtoClone LIKE produto;
```



# #4 Bora fazer um Chupacabra



```
CREATE TABLE produtoClone LIKE produto;
```

```
INSERT INTO produtoClone  
SELECT *  
FROM produto;
```

# #4

Se você quiser a tabela exista somente durante a sua conexão atual crie ela de maneira **temporária**:



```
CREATE TEMPORARY TABLE produtoClone LIKE produto;
```

#5



```
DROP TABLE IF EXISTS produto;  
CREATE TABLE IF NOT EXISTS produto  
(  
    id          INT(8) NOT NULL AUTO_INCREMENT,  
    nome        VARCHAR(100),  
    valor       DECIMAL(10, 2),  
    estoque     DECIMAL(10, 2),  
    ativo       BOOLEAN,  
    PRIMARY KEY (id),  
    CHECK(valor > 0)  
);
```



```
INSERT INTO produto VALUES (21, NULL, 38.42, 4, TRUE);
INSERT INTO produto VALUES (22, '', 37.42, 8, TRUE);
INSERT INTO produto VALUES (23, "Livro XA", 36.42, 10, TRUE);
INSERT INTO produto VALUES (24, "", 35.42, 11, TRUE);
INSERT INTO produto VALUES (25, 'Livro XB', 34.42, 15, TRUE);
INSERT INTO produto VALUES (26, 'Livro XC', 33.42, 2, TRUE);
INSERT INTO produto VALUES (27, 'Livro XD', 32.42, 8, TRUE);
INSERT INTO produto VALUES (28, NULL, 31.42, 4, TRUE);
INSERT INTO produto VALUES (29, 'Livro XE', 30.42, 3, TRUE);
INSERT INTO produto VALUES (30, 'Livro XF', 9.32, 2, TRUE);
INSERT INTO produto (valor, estoque, ativo) VALUES (9.32, 2, TRUE);
INSERT INTO produto (valor, estoque, ativo) VALUES (8.45, 3, TRUE);
```



```
SELECT COUNT(nome) FROM produto;
```



```
SELECT COUNT(nome) FROM produto;
```

```
+-----+
```

```
| COUNT(nome) |
```

```
+-----+
```

```
|           8 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```



```
SELECT COUNT(id) FROM produto;
```





```
SELECT COUNT(id) FROM produto;
```

```
+-----+
```

```
| COUNT(id) |
```

```
+-----+
```

```
|          12 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

# #5

Qual resultado é o correto?

8 ou 12?

○ ○ ○

```
SELECT COUNT(nome) FROM produto;
```

```
+-----+  
| COUNT(nome) |  
+-----+  
|           8 |  
+-----+
```

```
1 row in set (0.00 sec)
```

```
SELECT COUNT(id) FROM produto;
```

```
+-----+  
| COUNT(id) |  
+-----+  
|          12 |  
+-----+
```

```
1 row in set (0.00 sec)
```

# #5

Qual o resultado?

○ ○ ○

```
SELECT id, nome  
FROM produto  
WHERE nome IS NULL;
```

# #5

Qual o resultado?



```
SELECT id, nome  
FROM produto  
WHERE nome IS NULL;
```

+	—	+	—	+
	id		nome	
+	—	+	—	+
	21		NULL	
	28		NULL	
	31		NULL	
	32		NULL	
+	—	+	—	+

```
4 rows in set (0.00 sec)
```

# #5

Se você quer contar valores **NÃO NULOS**, use o **COUNT(campo)**

Se você quer contar **todos**, use **COUNT(\*)**.



```
SELECT COUNT(nome) FROM produto;
```

```
+-----+
```

```
| COUNT(nome) |
```

```
+-----+
```

```
|          8 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

```
SELECT COUNT(id) FROM produto;
```

```
+-----+
```

```
| COUNT(id) |
```

```
+-----+
```

```
|        12 |
```

```
+-----+
```

```
1 row in set (0.00 sec)
```

#6

Como trazer produtos com **nome** não preenchido?

# #6

Como trazer produtos com **nome** não preenchido?



```
SELECT id, nome, valor  
FROM produto  
WHERE nome = '';
```

id	nome	valor
22		37.42
24		35.42

2 rows in set (0.00 sec)



Temos 4 produtos com o campo nome não preenchido. Onde estão eles?

# #6

Temos 4 produtos com o campo nome não preenchido. Onde estão eles?



```
SELECT id, nome, valor  
FROM produto  
WHERE nome IS NULL;
```

id	nome	valor
21	NULL	38.42
28	NULL	31.42
31	NULL	9.32
32	NULL	8.45

4 rows in set (0.00 sec)

Alguns produtos estão com o campo EM BRANCO, outros estão NULOS (NULL).

Algumas produtos não foram editados, e continuam com o valor padrão do banco: NULL

Outros foram editados mas não tiveram o valor preenchido, estão agora com valor EM BRANCO.

# #6

Agora toda query terá de fazer uma verificação dupla.



```
SELECT *  
FROM produto  
WHERE (nome = '' OR nome IS NULL);
```

id	nome	valor	estoque	ativo
21	NULL	38.42	4.00	1
22		37.42	8.00	1
24		35.42	11.00	1
28	NULL	31.42	4.00	1
31	NULL	9.32	2.00	1
32	NULL	8.45	3.00	1

6 rows in set (0.00 sec)

# #6

Agora toda query terá de fazer uma verificação dupla.



```
SELECT *  
FROM produto  
WHERE (nome = '' OR nome IS NULL);
```

id	nome	valor	estoque	ativo
21	NULL	38.42	4.00	1
22		37.42	8.00	1
24		35.42	11.00	1
28	NULL	31.42	4.00	1
31	NULL	9.32	2.00	1
32	NULL	8.45	3.00	1

6 rows in set (0.00 sec)

Dica: Usar o valor padrão. Usar o default:



```
UPDATE produto SET nome = '' WHERE nome IS NULL;
```

```
ALTER TABLE produto  
    MODIFY COLUMN nome VARCHAR(100) DEFAULT '' NOT NULL;
```

Dica: Usar o valor padrão. Usar o default:



```
UPDATE produto SET nome = '' WHERE nome IS NULL;
```

```
Query OK, 4 rows affected (0.00 sec)
```

```
Rows matched: 4 Changed: 4 Warnings: 0
```

```
ALTER TABLE produto
```

```
    MODIFY COLUMN nome VARCHAR(100) DEFAULT '' NOT NULL;
```

Dica: Usar o valor padrão. Usar o default:



```
UPDATE produto SET nome = '' WHERE nome IS NULL;
```

```
Query OK, 4 rows affected (0.00 sec)
```

```
Rows matched: 4 Changed: 4 Warnings: 0
```

```
ALTER TABLE produto
```

```
    MODIFY COLUMN nome VARCHAR(100) DEFAULT '' NOT NULL;
```

```
Query OK, 0 rows affected (0.05 sec)
```

```
Records: 0 Duplicates: 0 Warnings: 0
```



#7

## #7 Dando valor pro nulo

Sabemos que NULO é NULO, VAZIO é VAZIO.

Nosso sistema permite o campo NULL, então temos alguns nulos no banco.



```
ALTER TABLE produto
  MODIFY COLUMN nome VARCHAR(100) DEFAULT '' NULL;

UPDATE produto
SET   nome = NULL
WHERE nome = '';
```

# #7

Trazer o nome do produto. Se ele for NULL, trazer o valor "Não informado".  
Como trazer um valor padrão na hora de executar a query?

# #7

Trazer o nome do produto. Se ele for NULL, trazer o valor "Não informado".  
Como trazer um valor padrão na hora de executar a query?



```
SELECT id, nome, COALESCE(nome, "Não informado")  
FROM produto;
```

# #7

Trazer o nome do produto. Se ele for NULL, trazer o valor "Não informado".

O COALESCE traz o primeiro valor não nulo do que passamos pra ele.

Isto é, se o nome for nulo, ele devolve "Não informado".

Também podemos usar com números:  
COALESCE(valor, 0).

```
○ ○ ○

+-----+-----+-----+
| id | nome | COALESCE(nome, "Não informado") |
+-----+-----+-----+
| 10 | Livro A | Livro A |
| 11 | Livro B | Livro B |
| 12 | Livro C | Livro C |
| 13 | Livro D | Livro D |
| 14 | Livro E | Livro E |
| 15 | Livro F | Livro F |
| 16 | Livro G | Livro G |
| 17 | Livro H | Livro H |
| 19 | Livro I | Livro I |
| 20 | Livro J | Livro J |
| 21 | NULL | Não informado |
| 22 | NULL | Não informado |
| 23 | Livro XA | Livro XA |
| 24 | NULL | Não informado |
| 25 | Livro XB | Livro XB |
| 26 | Livro XC | Livro XC |
| 27 | Livro XD | Livro XD |
| 28 | NULL | Não informado |
| 29 | Livro XE | Livro XE |
| 30 | Livro XF | Livro XF |
| 31 | NULL | Não informado |
| 32 | NULL | Não informado |
+-----+-----+-----+

22 rows in set (0.00 sec)
```

# Tabela exemplo para usar com as próximas dicas



```
DROP TABLE IF EXISTS venda;  
CREATE TABLE IF NOT EXISTS venda  
(  
    id          INT auto_increment,  
    produtora VARCHAR (20) DEFAULT '',  
    valor       DECIMAL (10, 2) DEFAULT 0,  
    PRIMARY KEY (id)  
);
```

○ ○ ○

```
INSERT INTO venda (produtora, valor) VALUES
('Microsoft', 100.00),
('Ubisoft', 99.00),
('Microsoft', 89.00),
('EA Games', 88.00),
('EA Games', 50.00),
('Microsoft', 80.00),
('', 0.00),
('', 100.00),
('Ubisoft', 0.00),
('EA Games', 100.00),
('Microsoft', 100.00),
('Microsoft', 100.00),
('EA Games', NULL),
('Microsoft', 0.00),
('EA Games', 100.00),
('', 0.00),
('Microsoft', 100.00),
('Ubisoft', 100.00),
('Ubisoft', 100.00),
(NULL, NULL),
('Microsoft', 100.00),
(NULL, 100.00),
('Microsoft', 100.00),
('Ubisoft', 100.00),
('Ubisoft', 100.00),
(NULL, 100.00);
```

#8



# #8

Numa consulta para saber como estão as vendas por produtora, agrupamos:



```
SELECT produtora, SUM(valor) AS total
FROM venda
GROUP BY produtora
ORDER BY produtora;
```

produtora	total
NULL	200.00
	100.00
EA Games	338.00
Microsoft	769.00
Ubisoft	499.00

5 rows in set (0.00 sec)

# #8

Numa consulta para saber como estão as vendas por produtora, agrupamos:

Tem vários resultados "menores" que não estou tão interessado.  
Só quero enxergar quem vendeu mais de **400**.



```
SELECT produtora, SUM(valor) AS total
FROM venda
GROUP BY produtora
ORDER BY produtora;
```

produtora	total
NULL	200.00
	100.00
EA Games	338.00
Microsoft	769.00
Ubisoft	499.00

5 rows in set (0.00 sec)

Usando subconsuta:

# #8

Usando subconsulta:



```
SELECT produtora,  
       total  
FROM   (SELECT produtora,  
               SUM(valor) AS total  
        FROM   venda  
        GROUP BY produtora) AS temp  
WHERE  total > 400  
ORDER BY produtora;
```

```
+-----+-----+  
| produtora | total |  
+-----+-----+  
| Microsoft | 769.00 |  
| Ubisoft   | 499.00 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

# #8

Mesmo resultado sem usar subconsulta  
(usando o HAVING)

# #8

Mesmo resultado sem usar subconsulta  
(usando o HAVING)



```
SELECT produtora,  
       SUM(valor) AS vendas  
FROM   venda  
GROUP BY produtora  
HAVING vendas > 400  
ORDER BY produtora;
```

```
+-----+-----+  
| produtora | vendas |  
+-----+-----+  
| Microsoft | 769.00 |  
| Ubisoft   | 499.00 |  
+-----+-----+  
2 rows in set (0.00 sec)
```

#9

# #9

Quais são as minhas tabelas?

○ ○ ○

-- MySQL

SHOW TABLES; -- (Que barbada esse tal de MySQL)

-- Microsoft SQL Server

SELECT \*

FROM information\_schema.TABLES;

-- ORACLE - Minhas tabelas

SELECT table\_name

FROM user\_tables;

-- ORACLE - Todas tabelas que tenho acesso

SELECT table\_name

FROM all\_tables;

-- POSTGRESQL

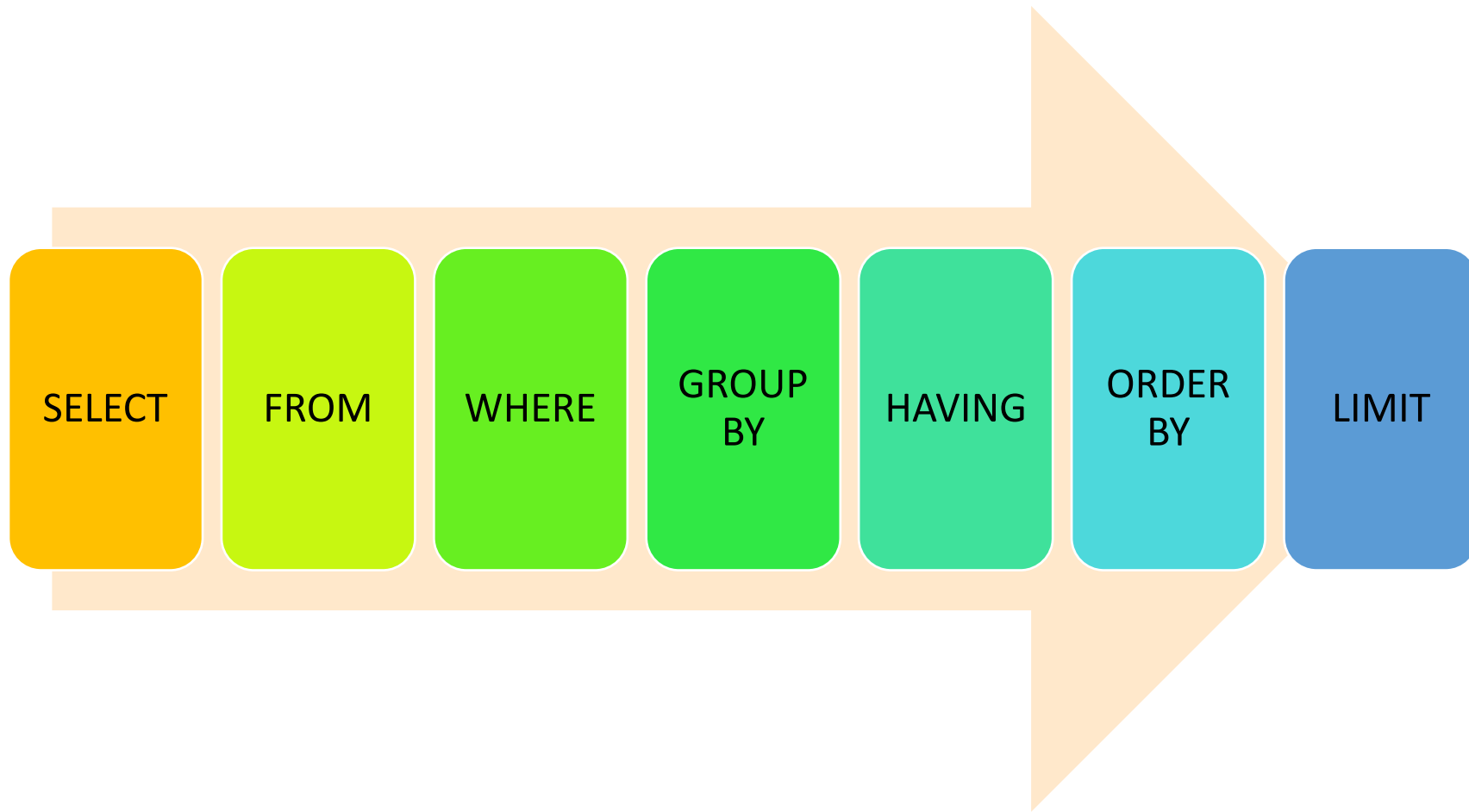
SELECT \*

FROM pg\_catalog.pg\_tables;



**Fonte:** [www.alura.com.br](http://www.alura.com.br)

# Sempre nessa ordem



# Fazer agora

Adaptar as seguintes alterações no script de criação do banco de dados utilizado no banco **cinema**:

1. Criar uma nova tabela **usuario**, contendo os campos **id**, **nome**, **email**, **cidade\_id**.
2. Criar a tabela **venda** (com os campos **data**, **hora** e **valorIngresso**), relacionar esta tabela com a tabela **sessao**. Em seguida criar um relacionamento entre as tabelas **usuario** e **venda**.
3. Criar INSERT's de dados necessários a fim de cadastrar 10 usuários, e seguida cadastrar 50 vendas de ingressos;