



# Sistemas de Bancos de Dados

***Prof. M.Sc. Marcelo Fermann Guimarães***

***mscfermann @ gmail.com***

21 Outubro 2019

# Sistemas de Bancos de Dados

## Agenda

- Junções
- *Distinct*
- *SubQueries*
  - Relacionadas e não relacionadas
  - Operador IN
  - Operador EXISTS
- Funções e conversões de tipo
- Trabalhos (Quiz W3Schools e ControleLivros)

# Sistemas de Bancos de Dados

## Junções

O Join (ligação) entre tabelas permite extrair, através de um único SELECT, informações contidas em diferentes tabelas.

O **Modelo Relacional** estabelece claramente as regras para a divisão da informação entre tabelas, de forma a evitar a duplicação de informação.

Essa dispersão da informação por diferentes tabelas é facilmente manipulável através da linguagem SQL, uma vez que a ligação entre tabelas será realizada através da ligação entre as Chaves Estrangeiras e as respectivas Chaves Primárias de onde são originárias.

Uma Chave Estrangeira é um campo (ou conjunto de campos) existente em uma tabela que permite ligar os dados desta tabela a uma outra, onde este mesmo conjunto de campos existe como Chave Primária.

# Sistemas de Bancos de Dados

## Junções

A Junção Interna (*inner join*) conecta duas (ou mais) tabelas e retorna apenas as linhas que satisfazem a condição de junção.

Na sintaxe ANSI, junções internas são indicadas da forma:

**tabela1 INNER JOIN tabela2 ON condição\_de\_junção**

```
SELECT cliente.cod_cliente, nome_cliente
```

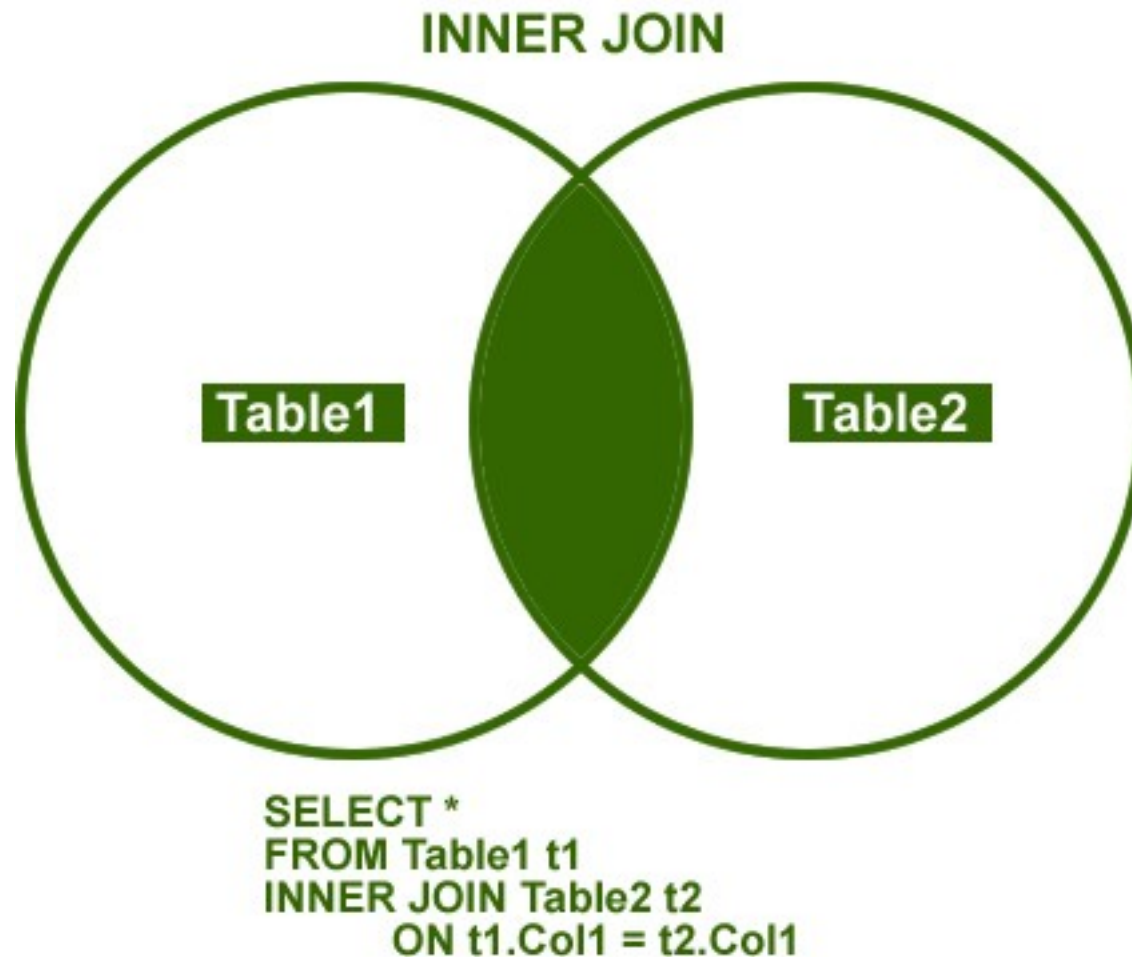
```
FROM Cliente
```

```
    INNER JOIN Venda ON Cliente.cod_cliente = Venda.cod_cliente
```

É possível juntar três ou mais tabelas com informações relacionadas. Neste caso, as junções de tabela são representadas com um INNER JOIN após o outro. O primeiro INNER JOIN cria uma "tabela" virtual. O segundo reúne essa tabela virtual à próxima e assim por diante.

# Sistemas de Bancos de Dados

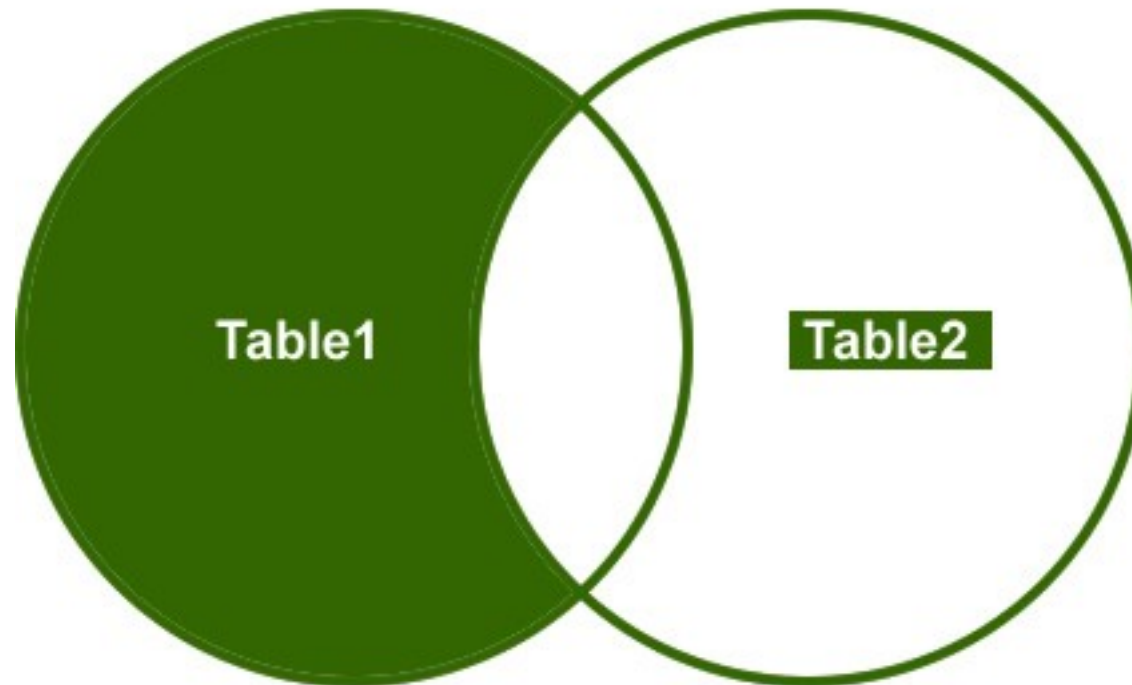
## Junções



# Sistemas de Bancos de Dados

## Junções

LEFT OUTER JOIN - WHERE NULL

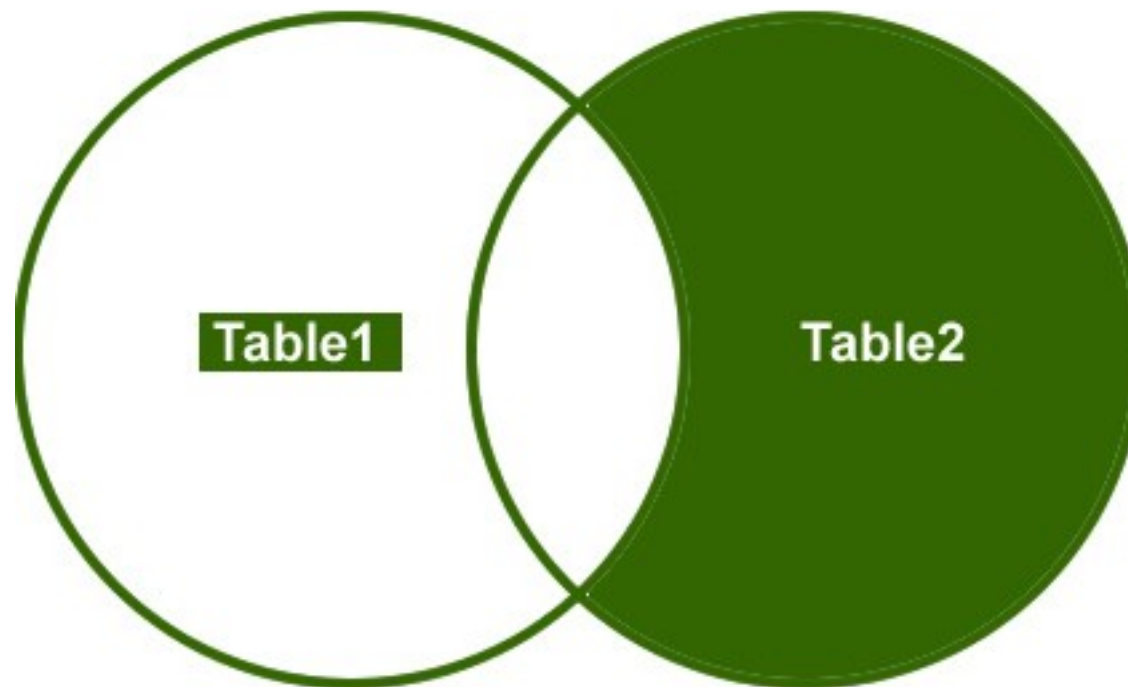


```
SELECT *  
FROM Table1 t1  
LEFT OUTER JOIN Table2 t2  
    ON t1.Col1 = t2.Col1  
WHERE t2.Col1 IS NULL
```

# Sistemas de Bancos de Dados

## Junções

RIGHT OUTER JOIN - WHERE NULL



```
SELECT *  
FROM Table1 t1  
RIGHT OUTER JOIN Table2 t2  
    ON t1.Col1 = t2.Col1  
WHERE t1.Col1 IS NULL
```

# Sistemas de Bancos de Dados

## Junções - UNION

Uma união não é propriamente uma ligação entre tabelas.

A UNION permite juntar o conteúdo de dois ou mais comandos SELECT.

```
select *  
from empregado
```

	nome	rg	cpf	depto	rg_supervisor	salario
1	NOME	1	111	1	NULL	1200
2	NOME 2	2	222	2	1	1000
3	NOME 3	3	333	1	1	1500
4	NOME 4	4	444	1	1	1500

```
select *  
from dependente
```

	rg_responsavel	nome_dependente
1	1	nome dependente filho do nome
2	2	nome dep 2 filho do nome 2
3	4	NOME



# Sistemas de Bancos de Dados

## Junções - UNION

Em uma UNION, o número de campos a serem selecionados em cada um dos comandos SELECT tem de ser igual. O nome dos campos não é relevante, mas o tipo de dados que pode ser agrupado varia de sistema para sistema.

*Obs1.:* Em uma UNION, o nome das colunas apresentado no resultado é o nome das colunas selecionadas na primeira instrução SELECT.

*Obs2.:* Cada comando SELECT pode conter a sua própria cláusula WHERE. No entanto, só poderá existir uma única cláusula ORDER BY no último SELECT, sendo a ordenação aplicada a todo o resultado.

```
select NOME
      from empregado
UNION
select nome_dependente
      from dependente
```

	NOME
1	NOME
2	NOME 2
3	NOME 3
4	NOME 4
5	nome dep 2 filho do nome 2
6	nome dependente filho do nome

# Sistemas de Bancos de Dados

## Questão

Quais são os departamentos que têm pessoas trabalhando?

```
Select * from empregado
```

	nome	rg	cpf	depto	rg_supervisor	salário
1	NOME	1	111	1	NULL	1200
2	NOME 2	2	222	2	1	1000
3	NOME 3	3	333	1	1	1500
4	NOME 4	4	444	1	1	1500

# Sistemas de Bancos de Dados

## DISTINCT

`select DISTINCT depto from empregado`

	depto
1	1
2	2

# Sistemas de Bancos de Dados

## ***SubQueries***

Uma das características do comando SELECT, que por vezes passa despercebida, reside no fato de qualquer comando SELECT não devolver dados, linhas ou colunas, mas devolver sempre uma Tabela como resultado da sua execução.

*Obs.:* O resultado do comando SELECT é sempre uma Tabela, ainda que da sua execução não resulte qualquer linha como resultado.

Uma Subconsulta (Subquery) consiste em um SELECT dentro de outro.

Um comando SELECT pode ser colocado:

- Dentro de um outro SELECT nas cláusulas WHERE, HAVING, SELECT e FROM.
- Dentro de uma outra subconsulta nas mesmas cláusulas do item anterior.
- Nos comandos INSERT, UPDATE e DELETE.
- Na definição de VIEW.

# Sistemas de Bancos de Dados

## ***SubQueries***

**Problema:** Qual é o Nome da Pessoa com o menor Salário na empresa?

Resolução: Temos aqui dois problemas para resolver.

- Qual é o valor do menor Salário?
- Qual é o Nome da pessoa a que esse salário corresponde?

Vamos começar por resolver o primeiro dos problemas.

**Problema:** Qual é o valor do menor Salário?

```
SELECT MIN(Salario) AS Menor  
FROM Pessoa
```


# Sistemas de Bancos de Dados

## ***SubQueries Não Relacionadas***

Se por acaso os salários forem aumentados, então teremos que calcular, novamente, o menor dos salários, alterando em seguida o valor existente no comando anterior.

Assim sendo, podemos obter o mesmo resultado através de comandos SELECT encadeados:

```
SELECT Nome  
FROM Pessoa  
WHERE Salario = ( SELECT MIN(Salario)  
                  FROM Pessoa )
```




# Sistemas de Bancos de Dados

## ***SubQueries Correlacionadas***

Quando um SELECT contém outro SELECT encadeado, pode acontecer que o SELECT interior necessite de valores do SELECT exterior.

**Problema:** Qual é o Nome das Pessoas cujo salário é menor que 15 vezes o conjunto das suas Comissões?

```
SELECT Nome, Salario
FROM Pessoa P
WHERE Salario < (SELECT SUM(Valor)
                  FROM Comissao C
                  WHERE C.Id = P.Id) * 15
```



O SELECT interior tem que considerar apenas o conjunto das comissões que pertencem a cada uma das pessoas que está sendo analisada individualmente pelo SELECT exterior, sendo por isso necessário fazer a ligação entre os dois comandos SELECT.

A subconsulta (colocada entre parênteses) que depende, para o seu funcionamento, de valores da consulta mais exterior é chamada de Consulta Correlacionada.

# Sistemas de Bancos de Dados

## ***SubQueries***

	CONSULTA NÃO-CORRELACIONADA	CONSULTA CORRELACIONADA
SENTIDO DE EXECUÇÃO	Do interior para o exterior.	Do exterior para o interior.
DEPENDÊNCIA DA CONSULTA INTERIOR	Não depende da Consulta exterior.	Depende da Consulta exterior.
EXECUÇÃO DA CONSULTA INTERIOR	Uma única vez.	Tantas vezes quantas as executadas pela Consulta exterior.



# Sistemas de Bancos de Dados

## *Operador IN*

O Operador IN permite verificar a existência de uma coluna em um conjunto de valores colocados entre parênteses.

O operador IN verifica se um valor existe no resultado de uma subconsulta, devolvendo True ou False.

**Problema:** Qual é o conjunto de pessoas que têm dependentes?

```
select rg_responsavel  
from dependentes
```

	rg_responsavel
1	1
2	2
3	4

```
select NOME  
from empregado  
where rg IN (1, 2, 4)
```

	NOME
1	NOME
2	NOME 2
3	NOME 4

# Sistemas de Bancos de Dados

## *Operador IN*

Podemos substituir os valores constantes pelo comando SELECT que lhes deu origem, obtendo-se o mesmo resultado.

**Problema:** Qual é o conjunto de pessoas que têm dependentes?

```
select NOME from empregado
where rg IN (select rg_responsavel
            from dependentes)
```

	NOME
1	NOME
2	NOME 2
3	NOME 4

# Sistemas de Bancos de Dados

## *Operador EXISTS*

O operador IN verifica se o resultado de uma determinada coluna ou expressão pertence ao conjunto de valores colocados entre parênteses ou que resulta de uma subconsulta. O operador EXISTS só pode ser utilizado para avaliar o resultado de subconsultas. O seu objetivo é verificar se a execução da subconsulta gerou ou não alguma linha de resultado.

O formato do operador EXISTS é:

```
SELECT ... ..  
FROM ... ..  
WHERE [NOT] EXISTS (Subconsulta)
```

Obs.: o operador EXISTS é um operador unário que verifica se a execução de uma subconsulta resultou alguma linha.

# Sistemas de Bancos de Dados

## *Operador EXISTS*

**Problema:** Qual é o conjunto de pessoas que têm dependentes?

```
select NOME from empregado
where EXISTS
    (select rg_responsavel
     from dependentes
     where rg = rg_responsavel)
```

# Sistemas de Bancos de Dados

## *Operador EXISTS*

Observações sobre o operador EXISTS:

- O operador EXISTS é um operador unário que devolve um valor lógico.
- O operador EXISTS verifica se resultou alguma linha da execução de uma subconsulta.
- O operador EXISTS nunca é precedido por qualquer coluna ou expressão.
- Normalmente, as Consultas associadas ao operador EXISTS são Correlacionadas.
- Como o operador EXISTS apenas devolve TRUE ou FALSE como resultado, as subconsultas associadas são, normalmente, do tipo (SELECT \* . . .).
- O operador NOT EXISTS devolve TRUE quando não resulta qualquer linha da subconsulta.

## Funções Matemáticas

- Pode-se usar vários operadores com colunas numéricas:
  - adição (+),
  - subtração (-),
  - multiplicação (\*),
  - divisão (/), e
  - módulo (%).
    - O módulo só pode ser usado com tipos inteiros e calcula o resto da divisão de dois números inteiros (Ex.:  $13 \% 4 = 1$ )

# Sistemas de Bancos de Dados

## Funções Matemáticas

- Além de operadores, você pode usar funções matemáticas do SQL Server, por exemplo:

ABS( <i>valor</i> )	retorna o valor absoluto (sem sinal) de um item.
POWER( <i>valor</i> , <i>p</i> )	retorna o <i>valor</i> elevado à potência <i>p</i> .
ROUND( <i>valor</i> , <i>n</i> )	arredonda o <i>valor</i> para <i>n</i> casas decimais.
SQRT ( <i>valor</i> )	retorna a raiz quadrada do valor especificado.
PI	valor constante 3.141592563589793

- Por exemplo, para arredondar o valor do preço de cada livro para duas casas decimais, pode ser feito o seguinte:

```
SELECT valor_livro Preço , ROUND(valor_livro, 2) "Preço com 2 casas  
decimais", Título_livro FROM livro
```

- Para outras funções, consulte o help do Transact-SQL em 'Math functions'.

# Sistemas de Bancos de Dados

## Funções de Caracteres

- Pode-se usar funções para manipular dados do tipo caracter (*char* ou *varchar*).
- Pode-se usar o operador + para concatenar dois valores de tipo caracter. Por exemplo:

```
SELECT DISTINCT nome_cliente, nome_municipio + ', ' +  
    UF 'Cidade, Estado' FROM Cliente INNER JOIN Venda  
    ON Cliente.cod_cliente = Venda.cod_cliente INNER JOIN  
    Cidade On Venda.Cod_municipio =  
    Cidade.Cod_municipio
```

- foi concatenado nome da cidade com o nome do estado, inserindo uma vírgula:  
Nome\_municipio + ', '+UF



# Funções de Caracteres

- Existem várias funções de manipulação de strings que podem ser usadas para outras tarefas, por exemplo:

ASCII( <i>caractere</i> )	retorna o código ASCII de um caractere.
CHAR( <i>inteiro</i> )	retorna o caractere, dado o seu código ASCII
LOWER( <i>expr</i> )	converte para minúsculas
UPPER( <i>expr</i> )	converte para maiúsculas
LTRIM( <i>expr</i> )	retira espaços à esquerda
RTRIM( <i>expr</i> )	retira espaços à direita
REPLICATE( <i>expr</i> , <i>n</i> )	repete uma expressão <i>n</i> vezes
SUBSTRING( <i>expr</i> , <i>início</i> , <i>tamanho</i> )	extrai uma parte de uma string desde <i>início</i> e com <i>tamanho</i> caracteres
RIGHT( <i>expr</i> , <i>n</i> )	retorna <i>n</i> caracteres à direita da string
REVERSE( <i>expr</i> )	inverte uma string
CHARINDEX('caractere', <i>expr</i> )	retorna a posição de um caractere dentro da string
SPACE( <i>n</i> )	retorna uma string com <i>n</i> espaços
STR( <i>número</i> , <i>n</i> , <i>d</i> )	converte um valor numérico para string, formatado com <i>n</i> caracteres na parte inteira (antes da vírgula) e <i>d</i> casas decimais depois da vírgula.
STUFF( <i>expr1</i> , <i>início</i> , <i>tamanho</i> , <i>expr2</i> )	substitui em <i>expr1</i> , os caracteres desde <i>início</i> até <i>tamanho</i> por <i>expr2</i>
DATALength( <i>expr</i> )	retorna a quantidade de caracteres em <i>expr</i>

# Sistemas de Bancos de Dados

## Funções de Caracteres

- Por exemplo:

```
SELECT substring(titulo_livro, 1, 30) Título , str(valor_livro,  
5, 1) Preço from Livro
```

- Esse comando mostrará até 30 caracteres da coluna título\_livro e a coluna valor\_livro com no máximo 5 números antes da vírgula e 1 casa decimal depois da vírgula.

```
SELECT Replicate('a', 10)
```

- Com este comando a letra “a” será mostrada 10 vezes.

# Sistemas de Bancos de Dados

## Funções de Data/Hora

- O tipo *datetime*, como vimos, armazena datas e horas. Algumas funções trabalham com esse tipo de dados:

DATEADD( <i>parte,número,data</i> )	adiciona um certo número de dias (ou meses, anos etc.) à data
DATEDIFF( <i>parte,data1,data2</i> )	subtrai as duas datas ( <i>data2 - data1</i> ), retornando um resultado em dias, meses etc. dependendo de <i>datepart</i>
DATEPART( <i>parte,data</i> )	retorna a parte especificada da data
DATENAME( <i>parte,data</i> )	retorna o nome por extenso da parte especificada
GETDATE()	retorna a data e hora atuais

- Nas funções acima, o argumento *parte*, especifica qual parte da data usar. Ele pode ser um dos seguintes valores:

# Sistemas de Bancos de Dados

## Funções de Data/Hora

yy	o ano
qq	o trimestre
mm	o mês
dy	o dia do ano (1-365)
dd	o dia do mês
wk	o número da semana (0-51)
dw	o dia da semana (domingo=1, segunda=2,...)
hh	a hora (0-23)
mi	os minutos
ss	os segundos
ms	os milisegundos

SELECT datepart(hh, getdate())

SELECT datepart(mi, getdate())

SELECT datepart(ss, getdate())

SELECT data\_venda, datepart(yy, data\_venda) Ano  
FROM Venda

# Sistemas de Bancos de Dados

## Conversão de Dados

- A função CONVERT permite converter de um tipo de dado para outro.
- A sua forma geral de uso é:

*CONVERT(tipo\_de\_dados, valor)*

- Por exemplo:

```
SELECT convert(char(10),nome_municipio) + '  
      '+ convert(char(2),UF) 'cidade, estado' FROM  
Cidade
```

# Sistemas de Bancos de Dados

## Conversão de Dados

- Com valores *datetime*, convert pode ter um parâmetro a mais, que especifica o formato de data a ser usado. Os formatos mais usados são:
  - 3 (padrão brasileiro dd/mm/aa;
  - 103 (dd/mm/aaaa);
  - 1 (mm/dd/yy); e
  - 101 (mm/dd/yyyy).
  - O default é 0, que mostra datas como:
    - Jan 01 1997 01:13:23 PM
- Por exemplo, para ver a data de hoje em formato brasileiro, execute:  
**SELECT convert(char,getdate(),103)**

# Sistemas de Bancos de Dados

## Conversão de Dados

- Outros exemplos:

```
SELECT convert(char(23),getdate ())
```

```
SELECT convert (datetime,convert(char(23),getdate ()))
```

```
SELECT convert(varchar, getdate(), 14)
```

```
SELECT convert(varchar, getdate(), 8)
```

```
SELECT substring(convert(char(20),getdate ()),12,8)
```

- Para converter valores numéricos em char pode ser utilizado a função CONVERT, por exemplo:

```
SELECT convert(char,cod_livro) FROM Livro
```

# Sistemas de Bancos de Dados

## **Moodle EDUCA (individual):**

- **Quiz W3Schools**
- **Banco de dados Controle de Livros**
- **Prazo: 28/10/2019**



# Sistemas de Bancos de Dados

## **Banco de dados Controle de Livros**

```
create table TB_Assunto(
```

```
    Sigla char(1) primary key,
```

```
    Descr varchar(50))
```

```
create table TB_Editora(
```

```
    CodEditora int primary key,
```

```
    Nome varchar(80))
```

# Sistemas de Bancos de Dados

```
create table TB_Autor(  
    MatrAutor int primary key,  
    NomeAutor varchar(80),  
    CPFAutor varchar(12) NOT NULL UNIQUE,  
    EndResidencialAutor varchar(120),  
    DataNascimentoAutor date,  
    NacionalidadeAutor varchar(30))
```

# Tópicos Especiais em Sistemas de Bancos de Dados

```
create table TB_Livro(  
    CodLivro int primary key,  
    Titulo varchar(80),  
    Preco money,  
    Lancamento date,  
    Assunto char(1),  
    Editora int,  
    constraint FK_Assunto_Livro foreign key (Assunto)  
references TB_Assunto(Sigla),  
    constraint FK_Editora_Livro foreign key (Editora)  
references TB_Editora(CodEditora))
```

# Sistemas de Bancos de Dados

```
create table TB_AutorLivro(  
    CodLivro int,  
    MatrAutor int,  
    primary key (CodLivro, MatrAutor),  
    constraint FK_Livro_AutorLivro foreign key (CodLivro)  
references TB_Livro(CodLivro),  
    constraint FK_Autor_AutorLivro foreign key  
(MatrAutor) references TB_Autor(MatrAutor))
```

# Sistemas de Bancos de Dados

**1a)** Apresente os comandos SQL para criação da nova tabela conforme o modelo relacional abaixo:

TB\_Pais (#Cod\_pais: texto (2),  
Nacionalidade: texto (30),  
Descr\_pais: texto (50))

**1b)** Apresente os comandos SQL para incluir **x** registros de países na tabela TB\_Pais. Sendo que:

**x** = resto da divisão por 10 da quantidade de caracteres do seu nome completo. Se resto igual 0, então **x** = 1

Exemplo: MARCELO FERMANN GUIMARAES = 23 caracteres sem espaços  
 $23/10 = 2,3 \rightarrow x = 3$

→ para o nome do exemplo deve-se inserir 3 países na tabela TB\_Pais

# Sistemas de Bancos de Dados

**1c)** Apresente os comandos SQL para incluir o código do país na tabela TB\_Autor como chave estrangeira incluindo a integridade referencial e excluindo a coluna NacionalidadeAutor (não é mais necessária).

Tabela TB\_Autor final:

TB\_Autor(#MatrAutor inteiro,

NomeAutor texto (80),

CPF Autor texto (12) ,

EndResidencialAutor texto (120),

DataNascimentoAutor data,

PaisAutor texto (2),

PaisAutor referencia TB\_Pais (CodPais))

**Perguntas ?**