

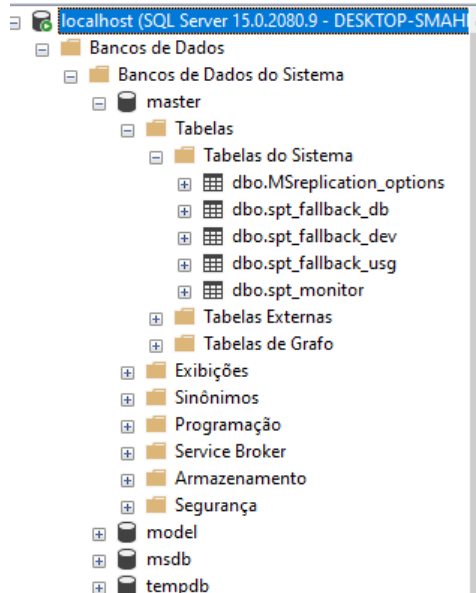
LINGUAGEM SQL

SQL Server Management Studio

Todo o script é apenas um COMANDO, uma LINHA DE CÓDIGO, sempre para algo ser efetivado, é necessário que o comando seja executado.

Observações:

Não crie dentro do Master, não é uma boa prática.

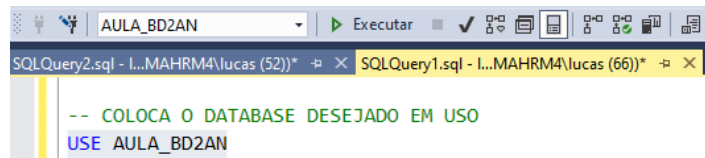


Sempre que criar, crie dentro de algum banco específico.

Use Base

Para utilizar a database correta e onde serão criadas as tabelas e os dados inseridos.

USE AULA_BD2AN



Create Table

Veja o exemplo da criação total de uma database e que essa database seja utilizada e a tabela criada dentro dessa database.

-- CRIA A DATABASE

CREATE DATABASE AULA_BD2AN

-- APAGA A DATABASE

DROP DATABASE AULA_BD2AN

-- COLOCA O DATABASE DESEJADO EM USO

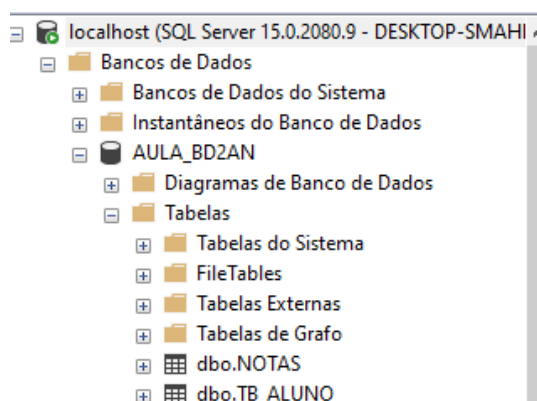
USE AULA_BD2AN

CREATE TABLE TB_ALUNO

```
(
    CPF INT,          -- TIPO: Inteiro
    NOME CHAR(20)     -- TIPO: Caractere com 20 Posições
)
```

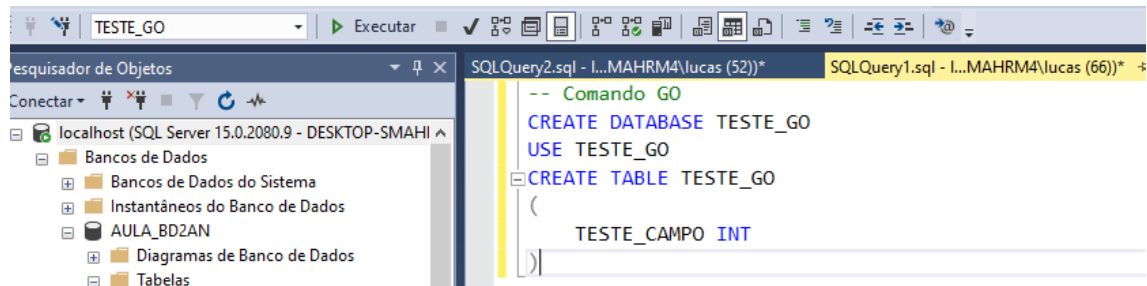
CREATE TABLE NOTAS

```
(
    ID INT,
    CPF_ALUNO INT,
    NOTA INT
)
```



Comando GO – Encerrador de Lotes

Começando o teste



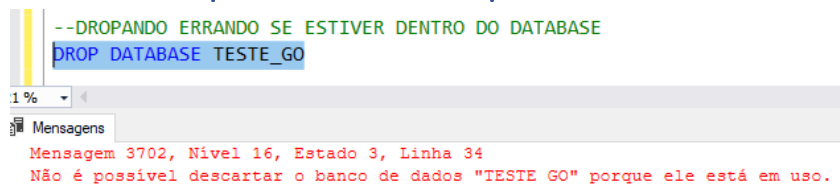
Atenção a estar dentro do TESTE_GO.

```
CREATE DATABASE TESTE_GO
GO
```

```
USE TESTE_GO
GO
```

```
CREATE TABLE TESTE_GO
(
    TESTE_CAMPO INT
)
GO
```

Erro ao Dropar a Database que está dentro



Drop Database

Para dropar uma database corretamente, é necessário sair, ir para uma de transição e de lá, dropar a que deseja.

```
-- TROQUE PARA UM DATABASE DE TRANSIÇÃO
USE MASTER
DROP DATABASE TESTE_GO
```

,

Para popular as tabelas, utilize. Para inserir registros

```
INSERT TB_ALUNO
(CPF, NOME)
VALUES
(100, 'LUCAS') -- CAMPOS DE TEXTO EM ASPAS SIMPLES
```

```
INSERT TB_ALUNO
(CPF, NOME)
VALUES
(150, 'LAURA')
```

OU, COM VÁRIOS VALORES AO MESMO TEMPO.

```
INSERT TB_ALUNO
(CPF, NOME)
VALUES
(200, 'TONY'),
(400, 'LIA'),
(500, 'MARIA')
```

SELECT

É onde vamos buscar os registros.

```
SELECT *
FROM TB_ALUNO
```

	CPF	NOME
1	100	LUCAS
2	150	LAURA
3	200	TONY
4	400	LIA
5	500	MARIA

Exercício:

Criar e Popular a TABELA de NOTAS: COM CPF, ID E NOTA

Lembre-se da criação da tabela de Notas que foi:

```
CREATE TABLE NOTAS
(
    ID INT,
    CPF_ALUNO INT,
    NOTA INT
)
```

INSERT SEM ERRO

Cuidado com o tamanho dos campos que foram criados, dessa forma abaixo, não terá erro.

```
INSERT NOTAS
(ID, CPF_ALUNO, NOTA)
VALUES
(1, 11111, 8),
(2, 22222, 10),
(3, 33333, 5)
```

	ID	CPF_ALUNO	NOTA
1	1	11111	8
2	2	22222	10
3	3	33333	5

Se fizer dessa forma..

```
INSERT NOTAS
(ID, CPF_ALUNO, NOTA)
VALUES
(1, 111111111111, 8),
(2, 222222222222, 10),
(3, 333333333333, 5)
```

Veja que vai apresentar um erro.

INSERT COM SELECT

```
INSERT INTO table2  
SELECT * FROM table1  
WHERE condition;
```

OU

```
INSERT INTO table2 (column1, column2, column3, ...)  
SELECT column1, column2, column3, ...  
FROM table1  
WHERE condition;
```

Em Colunas Específicas

```
INSERT INTO Customers (CustomerName, City, Country)  
SELECT SupplierName, City, Country FROM Suppliers;
```

Estouro Aritmético ao Converter

Mensagem 8115, Nível 16, Estado 2, Linha 43

Erro de estouro aritmético ao converter expression no tipo de dados int.
A instrução foi finalizada.

Horário de conclusão: 2021-08-10T20:57:52.3605555-03:00

Comandos DML

INSERT INTO

INSERT INTO + NOME TABELA + VALUES

>>>>> *INSERT INTO PESSOAS VALUES*

(DEFAULT, 'Lucas', '1994-04-14', 'M', '82.1', '1.83', 'BR');

date: Sempre: year - month - day

Sempre que você tiver um auto increment, pode deixar um default para o id)

Tabela auxiliar:

Table Cursos

idcurso	Nome	Carga	Ano
1	HTML4	40	2014
4	PGP	40	2010
5	JARVA	10	2000

UPDATE

UPDATE + NOME DA TABELA + SET = 'NOME DO CAMPO ATUALIZADO' + WHERE

PRIMARYKEY = 'CHAVE IDENTIFICADORA';

Exemplo:

>>>>> *UPDATE CURSOS SET = 'HTML5' WHERE ID CURSO = '1';*

Atualiza a tabela cursos para o nome html5 onde tenha uma chave identificadora do idcurso=1.

>>>>> *UPDATE CURSOS SET NOME = 'PHP', ANO = '2015' WHERE IDCURSO = '4' LIMIT 1;*

Ele vai remover uma linha/tupla inteira da coluna

DELETE FROM + NOME DA TABELA + WHERE CHAVE IDENTIFICADORA;

>>>>> *DELETE FROM CURSOS WHERE ANO = '2018' LIMIT 3;*

[Obs: Edit > Preferences > SQL Editor > Safe Update > Reconnect DTBS]

TRUNCATE

Cuidado: Remove e Deleta TODAS as linhas/tuplas da tabela.

TRUNCATE TABLE + NOMETABELA;

>>>> *TRUNCATE TABLE CURSOS;*

Comandos DDL

CREATE DATABASE

CREATE DATABASE cadastro;

CREATE TABLE

*CREATE TABLE + PESSOAS (
nome varchar(30) NOT NULL,
idade tinyint,
sexo char(1),
peso float,
altura float,
nacionalidade varchar(20) DEFAULT 'Brazil',
Id int NOT NULL AUTO_INCREMENT,
PRIMARY KEY (id)
) DEFAULT CHARSET = utf8;*

DROP DATABASE / DROP TABLE

Cuidado! O comando DROP excluir uma tabela ou uma database inteira!

*Drop Database cadastro
Drop Table + nome;*

ALTER TABLE(add, drop, after, first, modify, change, rename)

ALTER TABLE + ADD COLUMN

ALTER TABLE + NOMETABELA + ADD COLUMN + NOMECOLUNAASERADD + TIPOPRIMITIVO;

>>>> ALTER TABLE PESSOAS ADD COLUMN PROFISSÃO VARCHAR(30);

Obs: [Add Column adiciona sempre na última coluna.]

ALTER TABLE + DROP COLUMN

ALTER TABLE + NOMETABELA + DROP COLUMN + NOMECOLUNAASERREMOVIDA;

>>>> ALTER TABLE PESSOAS DROP COLUMN PROFISSÃO;

ALTER TABLE + AFTER

ALTER TABLE + NOMETABELA + ADD COLUMN + COLUNAASERADICIONADA + TIPOP + AFTER + COLUNAQUEAADICAOOCORRERÁ;

```
>>>>> ALTER TABLE PESSOAS ADD COLUMN PROFISSÃO VARCHAR(30) AFTER NOME;
```

[Obs: Sempre preciso informar onde eu quero adicionar a coluna, após alguma outra.]

ALTER TABLE + FIRST

ALTER TABLE + NOMETABELA + ADD COLUMN + COLUNAASERADICIONADA + TIPOP + FIRST;

```
>>>>> ALTER TABLE PESSOAS ADD COLUMN NACIONALIDADE VARCHAR(30) FIRST;
```

[Obs: Vai adicionar no primeiro lugar, antes de todas as colunas.]

ALTER TABLE + MODIFY

ALTER TABLE + NOMETABELA + MODIFY COLUMN + NOME DACOLUNA + TIPOP + CONSTRAINT;

```
>>>>> ALTER TABLE PESSOAS MODIFY COLUMN PROFISSÃO VARCHAR(20) NOT NULL;
```

[Obs: O modify apenas modifica o tipo da coluna, não o nome.]

ALTER TABLE + CHANGE

ALTER TABLE + NOMETABELA + CHANGE COLUMN + COLUNAANTIGA + COLUNANOVA + TIPOPRIMITIVO;

```
>>>>> ALTER TABLE PESSOAS CHANGE COLUMN PROFISSÃO PROFS VARCHAR(20);
```

[Obs: O antigo nome da coluna era 'profissao' agora é 'profs'. O change altera o nome da coluna!]

ALTER TABLE + RENAME

ALTER TABLE + NOMETABELA + RENAME TO + NOVO NOMETABELA;

```
>>>>> ALTER TABLE PESSOAS RENAME TO HUMANOS;
```

[Obs: O alter table rename, é para mudar o nome da tabela!]

ALTER TABLE e ALTER COLUMN - Alterando o tipo de Dado da Coluna

Veja que a coluna foi criada incorretamente com o tipo de dado INT.

```
5      CD_EMPRESA      INT      NOT NULL
6      , CD_TP_MATERIAL INT      NOT NULL
7      , DS_TP_MATERIAL INT      NOT NULL
8
9
10     CONSTRAINT PK_TB_TP_MATERIAL_CD_EMP
11     , CONSTRAINT FK_TB_TP_MATERIAL_CD_EMP
12 )
13
14 SP HFIP TP MATERIAI
```

110 %

Resultados		Mensagens	
Name	Owner	Type	Created_datetime
TP_MATERIAL	dbo	user table	2022-01-18 20:10:23.040

Column_name	Type	Computed	Length	Prec	Scale	Nullable
CD_EMPRESA	int	no	4	10	0	r
CD_TP_MATERIAL	int	no	4	10	0	r
DS_TP_MATERIAL	int	no	4	10	0	r

Para alterar, utilize:

```
ALTER TABLE TP_MATERIAL ALTER COLUMN DS_TP_MATERIAL VARCHAR(20) NOT NULL
```

```
15
16 ALTER TABLE TP_MATERIAL ALTER COLUMN DS_TP_MATERIAL VARCHAR(20) NOT NULL
```

0 %

Resultados		Mensagens	
Name	Owner	Type	Created_datetime
TP_MATERIAL	dbo	user table	2022-01-18 20:10:23.040

Column_name	Type	Computed	Length	Prec	Scale	Nullable	Trim TrailingBlanks	FixedLenNullInSource	Collation
CD_EMPRESA	int	no	4	10	0	no	(n/a)	(n/a)	NULL
CD_TP_MATERIAL	int	no	4	10	0	no	(n/a)	(n/a)	NULL
DS_TP_MATERIAL	varchar	no	20			no	no	no	Latin1_General_CI_AS

Veja que até a collation que estava errada, pode ser corrigida.

Gerenciando Cópias de Segurança – Back Up de Dados

DUMP

O DUMP serve para gerenciar cópias (backup de dados) para que não se percam caso algum comando manipule as tabelas e a DB.

Export

Server > Data Export > Selecionar as info's > Export to Self Contained File > Include Create Schema > Documents > Dumps

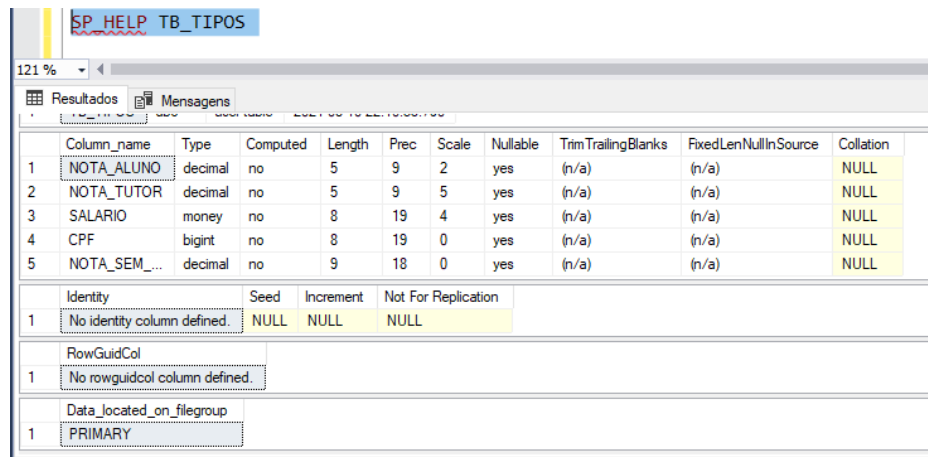
Impor

SP_HELP

Crie a tabela

```
CREATE TABLE TB_TIPOS
(
    NOTA_ALUNO DECIMAL(9,2), -- EXEMPLO: 9999999.99 ~São nove 9, sendo 2 casas
    decimais
    NOTA_TUTOR DECIMAL(9,5), -- EXEMPLO: 99999.99999 ~São nove 9, sendo 5
    casas decimais
    SALARIO MONEY,           -- EXEMPLO: MONEY:
    CPF BIGINT,              -- CPF
    NOTA_SEM_PARAMETRO DECIMAL
)
```

Digite o “SP_HELP” e o nome da tabela para entender.



	Column_name	Type	Computed	Length	Prec	Scale	Nullable	Trim TrailingBlanks	FixedLenNullInSource	Collation
1	NOTA_ALUNO	decimal	no	5	9	2	yes	(n/a)	(n/a)	NULL
2	NOTA_TUTOR	decimal	no	5	9	5	yes	(n/a)	(n/a)	NULL
3	SALARIO	money	no	8	19	4	yes	(n/a)	(n/a)	NULL
4	CPF	bigint	no	8	19	0	yes	(n/a)	(n/a)	NULL
5	NOTA_SEM...	decimal	no	9	18	0	yes	(n/a)	(n/a)	NULL

	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

SP_HELP em Tabela Temporária

Precisa primeiro acessar a base temporária

```
exec tempdb..sp_help #temp
```

SP_HELP em Schema diferente do .dbo

```
EXEC SP_HELP 'nomeschema.nomedatabela'
```

Tipos de Dados

Data type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,767	2
int	2 ³¹ (-2,147,483,648) to 2 ³¹ -1 (2,147,483,647)	4
Bigint	-2 ⁶³ - 2 ⁶³ -1 (+/- 9 quintillion)	8
bit	1, 0 or NULL	1
decimal/numeric	- 10 ³⁸ +1 through 10 ³⁸ – 1 when maximum precision is used	5-17
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
smallmoney	- 214,748.3648 to 214,748.3647	4

Data Type	Range	Storage (bytes)
float(n)	- 1.79E+308 to -2.23E-308, 0 and 2.23E-308 to 1.79E+308	Depends on value of n, 4 or 8 bytes
real	- 3.40E + 38 to -1.18E - 38, 0 and 1.18E - 38 to 3.40E + 38	4

Data Type	Range	Storage
CHAR(n)	1-8000 characters	n bytes
VARCHAR(n)	1-8000 characters	n+2 bytes
VARCHAR(MAX)	1-2 ³¹ -1 characters	Actual length + 2

Data Type	Range	Storage
NCHAR(n)	1-4000 characters	2*n bytes
NVARCHAR(n)	1-4000 characters	(2*n) +2 bytes
NVARCHAR(MAX)	1-2 ³¹ -1 characters	(Actual length) * 2 + 2

Data Type	Storage (bytes)	Date Range	Accuracy	Recommended Entry Format
DATETIME	8	January 1, 1753 to December 31, 9999	3-1/3 milliseconds	'YYMMDD hh:mm:ss:nnn'
SMALLDATETIME	4	January 1, 1900 to June 6, 2079	1 minute	'YYMMDD hh:mm:ss'
DATETIME2	6 to 8	January 1, 0001 to December 31, 9999	100 nanoseconds	'YYMMDD hh:mm:ss.nnnnnnn'
DATE	3	January 1, 0001 to December 31, 9999	1 day	'YYYY-MM-DD'
TIME	3 to 5		100 nanoseconds	'hh:mm:ss.nnnnnnn'
DATETIMEOFFSET	8 to 10	January 1, 0001 to December 31, 9999	100 nanoseconds	'YY-MM-DD hh:mm:ss:nnnnnnn [+ -]hh:mm'

Data Type	Language-Neutral Formats	Examples
DATETIME	'YYYYMMDD hh:mm:ss.nnn' 'YYYY-MM-DDThh:mm:ss.nnn' 'YYYYMMDD'	'20120212 12:30:15.123' '2012-02-12T12:30:15.123' '20120212'
SMALLDATETIME	'YYYYMMDD hh:mm' 'YYYY-MM-DDThh:mm' 'YYYYMMDD'	'20120212 12:30' '2012-02-12T12:30' '20120212'
DATETIME2	'YYYY-MM-DD' 'YYYYMMDD hh:mm:ss.nnnnnnn' 'YYYY-MM-DD hh:mm:ss.nnnnnnn' 'YYYY-MM-DDThh:mm:ss.nnnnnnn' 'YYYYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567' '2012-02-12 12:30:15.1234567' '2012-02-12T12:30:15.1234567' '20120212' '2012-02-12'
DATE	'YYYYMMDD' 'YYYY-MM-DD'	'20120212' '2012-02-12'
TIME	'hh:mm:ss.nnnnnnn'	'12:30:15.1234567'
DATETIMEOFFSET	'YYYYMMDD hh:mm:ss.nnnnnnn [+ -]hh:mm' 'YYYY-MM-DD hh:mm:ss.nnnnnnn [+ -]hh:mm' 'YYYYMMDD' 'YYYY-MM-DD'	'20120212 12:30:15.1234567 +02:00' '2012-02-12 12:30:15.1234567 +02:00' '20120212' '2012-02-12'

Data Type	Range	Storage (bytes)
binary(n)	1-8000 bytes	n bytes
varbinary(n)	1-8000 bytes	n bytes + 2
varbinary(MAX)	1-2.1 billion (approx) bytes	actual length + 2

Númericos

CRIANDO a tabela que conterá os CAMPOS as COLUNAS que serão populadas.

```
CREATE TABLE TB_TIPOS
(
    NOTA_ALUNO DECIMAL(9,2), --EXEMPLO: 9999999.99 ~São nove 9, sendo 2 casas decimais
    NOTA_TUTOR DECIMAL(9,5), --EXEMPLO: 99999.99999 ~São nove 9, sendo 5 casas decimais
    SALÁRIO MONEY,
    CPF BIGINT,
    NOTA_SEM_PARAMETRO DECIMAL
)
```

INSERINDO OS VALORES NA TABELA DE “TB_TIPO”

```
INSERT TB_TIPOS
(NOTA_ALUNO, NOTA_TUTOR, SALARIO, CPF, NOTA_SEM_PARAMETRO)
VALUES
(9999999.99, 9999.11234, 15000.50, 43826699910, 100.99)
```

Data type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,767	2
int	2 ³¹ (-2,147,483,648) to 2 ³¹ -1 (2,147,483,647)	4
Bigint	-2 ⁶³ - 2 ⁶³ -1 (+/- 9 quintillion)	8
bit	1, 0 or NULL	1
decimal/numeric	- 10 ³⁸ +1 through 10 ³⁸ – 1 when maximum precision is used	5-17
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
smallmoney	- 214,748.3648 to 214,748.3647	4

Observações do VARCHAR

```
--          123 45678
-- VARCHAR(20) ANA SILVA O VARCHAR DESPREZA OS OUTROS 12 CARACTERES
--          123 45678910111213..
-- CHAR(20) ANA SILVA O CHAR ARMAZENA OS 20
```

NOT NULL

Quando está dessa forma, tem que declarar obrigatoriamente o CPF e NOME, se não, vai dar erro.

```
CREATE TABLE TB_FUNC
(
    CPF BIGINT NOT NULL,
    NOME VARCHAR(50) NOT NULL,
    SALARIO MONEY,
    DATA_NASC DATE -- Recomendando: 'YYYY-MM-DD'
)

INSERT TB_FUNC
( CPF, NOME, SALARIO, DATA_NASC )
VALUES
( 101, 'LUCAS', 1000, '1994-04-14' )
```

O código abaixo da erro:

```
INSERT TB_FUNC
(SALARIO, DATA_NASC)
VALUES
(1000, '1994-04-14')
```

IDENTITY/AUTO-INCREMENT

```
CREATE TABLE TB_FUNC
(
    ID_FUNC          INT IDENTITY,
    CPF              BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'
)
```

E populo

```
INSERT TB_FUNC
( CPF, NOME, SALARIO, DATA_NASC )
VALUES
( 101, 'LUCAS', 1000, '1994-04-14' )
```

O resultado é:

	ID_FUNC	CPF	NOME	SALARIO	DATA_NASC
1	1	101	LUCAS	1000,00	1994-04-14

INCREMENTO DE X EM X

Fazendo o seed dessa forma:

```
CREATE TABLE TB_FUNC
(
    ID_FUNC          INT IDENTITY(10,5), --inicio e o seed
    CPF              BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'
)
```

	ID_FUNC	CPF	NOME	SALARIO	DATA_NASC
1	10	101	LUCAS	1000,00	1994-04-14

Vai dar o ID de início em 10 e de 5 em 5 ele vai somando,

```
CREATE TABLE TB_DEPENDENTES
(
    ID_DEPENDENTE      INT IDENTITY (1000,2),
    CPF                 BIGINT                        NOT
NULL,
    CPF_RESP            BIGINT                        NOT NULL,
    NOME                VARCHAR(50)                  NOT NULL,
    DATA_NASC          DATE
)
```

NÃO PODEMOS INSERIR EM VALORES O CAMPO DO ID_DEPENDENTE, VAI DAR ERRO.

```
INSERT TB_DEPENDENTES
( ID_DEPENDENTE, CPF, CPF_RESP, NOME, DATA_NASC )
VALUES
(2000, 1001, 101, 'LUCAS', '2021-08-24')
```

Mensagens

Mensagem 544, Nível 16, Estado 1, Linha 48
 Não é possível inserir um valor explícito para a coluna de identidade na tabela 'TB_DEPENDENTES' quando IDENTITY_INSERT

Portanto, tem que ser assim:

```
INSERT TB_DEPENDENTES
(CPF, CPF_RESP, NOME, DATA_NASC )
VALUES
(1001, 101, 'LUCAS', '2021-08-24')
```

	ID_DEPENDENTE	CPF	CPF_RESP	NOME	DATA_NASC
1	1000	1001	101	LUCAS	2021-08-24

Criando uma Primary Key

Uma PK nunca vai ser nula.

Sintaxe: `CONSTRAINT <NOME_DA_PK> PRIMARY KEY (CAMPO_PK)`

```
CREATE TABLE TB_FUNC
(
    ID_FUNC          INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF              BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'

    CONSTRAINT PK_TB_FUNC_CPF PRIMARY KEY (CPF)
)
```

Veja:

SP_HELP TB_FUNC

110 %

Resultados		Mensagens	
Name	Owner	Type	Created_datetime
1 TB_FUNC	dbo	usertable	2021-08-24 20:33:56.850

Column_name	Type	Computed	Length	Prec	Scale	Nullable	Trim TrailingBlanks	FixedLenNullInSource	Collation
1 ID_FUNC	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2 CPF	bigint	no	8	19	0	no	(n/a)	(n/a)	NULL
3 NOME	var...	no	50			no	no	no	Latin1_General_CI_AS
4 SALARIO	mo...	no	8	19	4	yes	(n/a)	(n/a)	NULL
5 DATA_NASC	date	no	3	10	0	yes	(n/a)	(n/a)	NULL

Identity	Seed	Increment	Not For Replication
1 ID_FUNC	10	5	0

RowGuidCol
1 No rowguidcol column defined.

Data_located_on_filegroup
1 PRIMARY

index_name	index_description	index_keys
1 PK_TB_FUNC_CPF	clustered, unique, primary key located on PRIMARY	CPF

constraint_type	constraint_name	delete_action	update_action	status_enabled	status_for_replication	constraint_keys
1 PRIMARY KEY (clustered)	PK_TB_FUNC_CPF	(n/a)	(n/a)	(n/a)	(n/a)	CPF

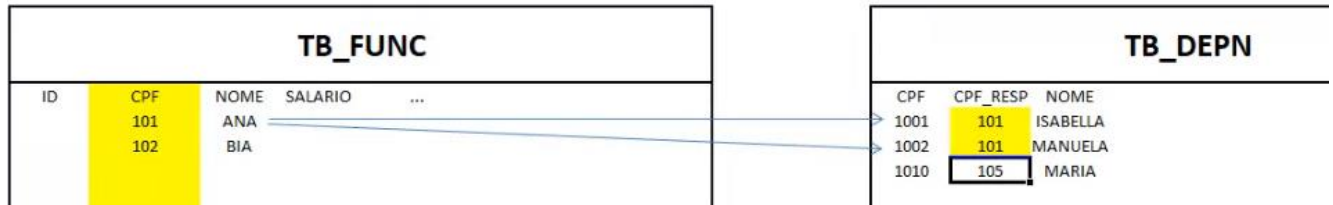
Índice Clusterizado

Significa que o índice está ordenado, logo vai aumentar a performance do Banco de Dados, ele cria um índice exato e preciso, que a pesquisa vai diretamente sem passar por outros índices.

Criando uma Foreign Key

Cria uma relação de dependência de uma tabela com outra tabela. É uma relação Hierárquica, (Mãe > Filha) | (Master > Slave)

Sintaxe: `CONSTRAINT <NOME_DA_FK> FOREIGN KEY (CAMPO_FK) REFERENCES TABELA_MAE(CAMPO_MAE)`



```
CREATE TABLE TB_DEPENDENTES
(
    ID_DEPENDENTE          INT IDENTITY (1000,2),
    CPF                    BIGINT NOT
NULL,
    CPF_RESP                BIGINT NOT NULL,
    NOME                    VARCHAR(50) NOT NULL,
    DATA_NASC              DATE
CONSTRAINT FK_TB_DEPENDENTE_CPF_RESP FOREIGN KEY (CPF_RESP) REFERENCES
TB_FUNC(CPF)
)
```


A Relação entre PK e FK

```
CREATE TABLE TB_FUNC
(
    ID_FUNC          INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF              BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'

    CONSTRAINT PK_TB_FUNC_CPF PRIMARY KEY (CPF)
)
```

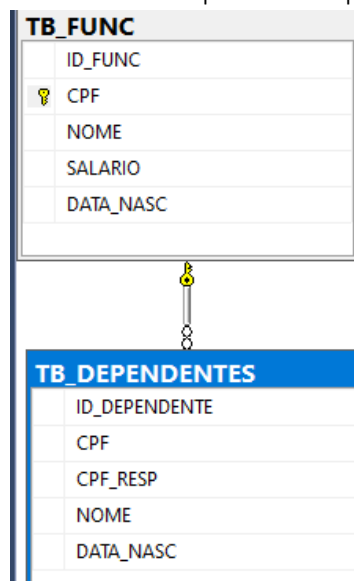
```
CREATE TABLE TB_DEPENDENTES
(
    ID_DEPENDENTE    INT IDENTITY (1000,2),
    CPF              BIGINT NOT NULL,
    CPF_RESP         BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    DATA_NASC       DATE

    CONSTRAINT FK_TB_DEPENDENTE_CPF_RESP FOREIGN KEY (CPF_RESP) REFERENCES TB_FUNC(CPF)
)
```

	ID_FUNC	CPF	NOME	SALARIO	DATA_NASC
1	10	101	LUCAS	1000,00	1994-04-14

	ID_DEPENDENTE	CPF	CPF_RESP	NOME	DATA_NASC
1	1000	1001	101	TONY	2018-05-21

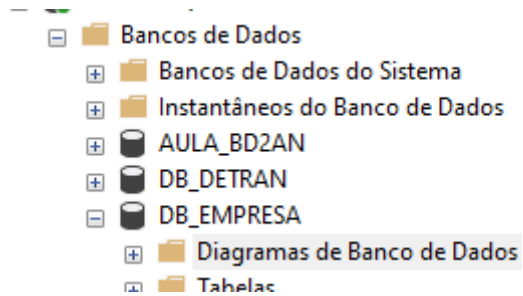
Só foi possível criar o relacionamento porque existe um CPF "101" lá na TABELA DE FUNCIONARIOS, se não houvesse um cpf 101, ia dar erro na hora de criar o DEPENDENTE que tem o cpf 1001, é necessário que tivesse o CPF 101 criado.



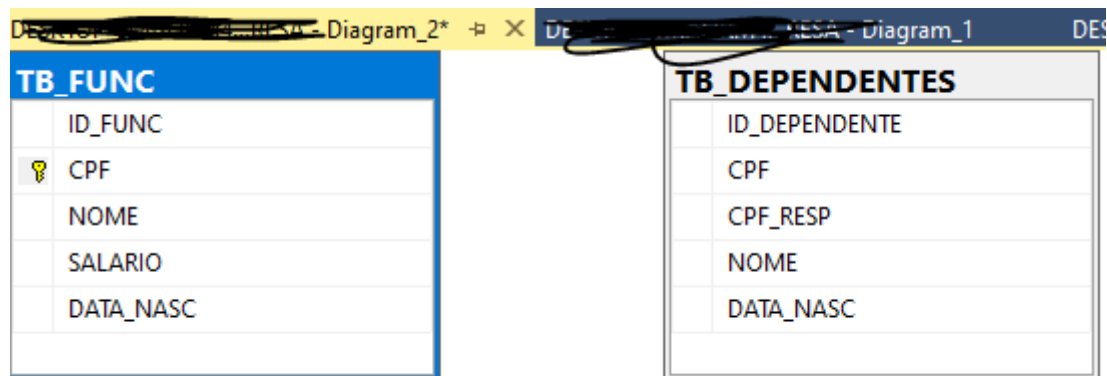
A chave significa que a TB_FUNC é a MÃE

Diagrama de Banco de Dados

Clique com o botão direito no



E crie um novo



Exercício Proposto

Script exemplo:

```
CREATE DATABASE DB_EMPRESA
```

```
USE DB_EMPRESA
```

```
CREATE TABLE TB_FUNC
```

```
(  
    ID_FUNC          INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO  
    CPF              BIGINT                NOT NULL,  
    NOME              VARCHAR(50)           NOT NULL,  
    SALARIO           MONEY,  
    DATA_NASC        DATE -- Recomendando: 'YYYY-MM-DD'  
  
    CONSTRAINT PK_TB_FUNC_CPF PRIMARY KEY (CPF)  
)
```

```
DROP TABLE TB_FUNC
```

```
INSERT TB_FUNC
```

```
( CPF, NOME, SALARIO, DATA_NASC )
```

```
VALUES
```

```
( 101, 'LUCAS', 1000, '1994-04-14' )
```

```
SELECT * FROM TB_FUNC
```

```
SP_HELP TB_FUNC
```

```
--CRIAR TABELA DE DEPENDENTES COM ID AUTO NUMERACAO (QUE INICEI EM 1000 E CONTE  
2)
```

```
--CPFS E NOMES NÃO ACEITAM NULOS.
```

```
CREATE TABLE TB_DEPENDENTES
```

```
(  
    ID_DEPENDENTE    INT IDENTITY (1000,2),  
    CPF              BIGINT                NOT NULL,  
    CPF_RESP          BIGINT                NOT NULL,  
    NOME              VARCHAR(50)           NOT NULL,  
    DATA_NASC        DATE  
  
    CONSTRAINT FK_TB_DEPENDENTE_CPF_RESP FOREIGN KEY (CPF_RESP) REFERENCES  
TB_FUNC(CPF)  
)
```

```
DROP TABLE TB_DEPENDENTES
```

```
INSERT TB_DEPENDENTES
```

```
( CPF, CPF_RESP, NOME, DATA_NASC )
```

```
VALUES
```

```
(1001, 101, 'TONY', '2018-05-21' )
```

```
CREATE TABLE TB_PRODUTOS
```

```
(  
    COD                                INT IDENTITY(100, 10),  
    NOME                                VARCHAR(50) NOT NULL  
  
    CONSTRAINT PK_TB_PRODUTOS_COD PRIMARY KEY (COD)  
)
```

```
CREATE TABLE TB_FORNECEDORES
```

```
(  
    COD_FORNECEDOR                    INT IDENTITY,  
    NOME                                VARCHAR(50) NOT NULL,  
    COD_PRODUTO                        INT NOT NULL  
  
    CONSTRAINT FK_TB_FORNECEDORES_COD_PRODUTOS FOREIGN KEY (COD_PRODUTO)  
REFERENCES TB_PRODUTOS(COD)  
)
```

```
INSERT TB_PRODUTOS
```

```
(NOME)
```

```
VALUES
```

```
( 'CELULAR' ),  
( 'TELEVISAO' ),  
( 'COMPUTADOR' )
```

```
INSERT TB_FORNECEDORES
```

```
(NOME, COD_PRODUTO)
```

```
VALUES
```

```
( 'APPLE' , 100 ),  
( 'SAMSUNG' , 110 ),  
( 'SAMSUNG' , 120 )
```

```
SELECT * FROM TB_PRODUTOS
```

```
SELECT * FROM TB_FORNECEDORES
```

Correção:

TB_PRODUTO	
COD	NOME
9001	LIQUIDIFICADOR
9002	TELEVISÃO

TB_FORNECEDOR		
COD	COD_PROD	NOME
10001	9002	SONY
10002	9001	ARNO
10001	9001	SONY

Código da Aula

```
CREATE DATABASE DB_EMPRESA
```

```
USE DB_EMPRESA
```

```
-- COM SINTAXE COMPLETA
```

```
CREATE TABLE TB_FUNC
```

```
(
    ID_FUNC          INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF              BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'

    CONSTRAINT PK_TB_FUNC_CPF PRIMARY KEY (CPF)
)
```

```
-- COM SINTAXE RESUMIDA DA PK
```

```
CREATE TABLE TB_FUNC
```

```
(
    ID_FUNC          INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF              BIGINT NOT NULL PRIMARY KEY,
    NOME             VARCHAR(50) NOT NULL,
    SALARIO          MONEY,
    DATA_NASC       DATE -- Recomendando: 'YYYY-MM-DD'
)
```

```
DROP TABLE TB_FUNC
```

```
INSERT TB_FUNC
```

```
( CPF, NOME, SALARIO, DATA_NASC )
```

```
VALUES
```

```
( 101, 'LUCAS', 1000, '1994-04-14' )
```

```
SELECT * FROM TB_FUNC
```

```
SP_HELP TB_FUNC
```

```
-- COM SINTAXE COMPLETA
```

```
CREATE TABLE TB_DEPENDENTES
```

```
(
    ID_DEPENDENTE    INT IDENTITY (1000,2),
    CPF              BIGINT NOT NULL,
    CPF_RESP         BIGINT NOT NULL,
    NOME             VARCHAR(50) NOT NULL,
    DATA_NASC       DATE

    CONSTRAINT PK_TB_DEPENDENTE_CPF PRIMARY KEY (CPF),
    CONSTRAINT FK_TB_DEPENDENTE_CPF_RESP FOREIGN KEY (CPF_RESP) REFERENCES
TB_FUNC(CPF)
)
```

```
-- COM SINTAXE RESUMIDA DA FK
CREATE TABLE TB_DEPENDENTES
(
    ID_DEPENDENTE      INT IDENTITY (1000,2),
    CPF                BIGINT          NOT NULL,
    CPF_RESP           BIGINT NOT NULL FOREIGN KEY REFERENCES TB_FUNC(CPF),
    NOME               VARCHAR(50)     NOT NULL,
    DATA_NASC         DATE
)
```

```
DROP TABLE TB_DEPENDENTES
```

```
INSERT TB_DEPENDENTES
(CPF, CPF_RESP, NOME, DATA_NASC )
VALUES
(1001, 101, 'TONY', '2018-05-21')
```

```
SELECT * FROM TB_FUNC
SELECT * FROM TB_DEPENDENTES
```

```
-----

CREATE TABLE TB_PRODUTOS
(
    COD                INT IDENTITY(100, 10),
    NOME               VARCHAR(50) NOT NULL

    CONSTRAINT PK_TB_PRODUTOS_COD PRIMARY KEY (COD)
)
```

```
CREATE TABLE TB_FORNECEDORES
(
    COD_FORNECEDOR     INT IDENTITY,
    NOME               VARCHAR(50) NOT NULL,
    COD_PRODUTO        INT NOT NULL

    CONSTRAINT FK_TB_FORNECEDORES_COD_PRODUTOS FOREIGN KEY (COD_PRODUTO)
REFERENCES TB_PRODUTOS(COD)
)
```

```
INSERT TB_PRODUTOS
(NOME)
VALUES
('CELULAR'),
('TELEVISAO'),
('COMPUTADOR')
```

```
INSERT TB_FORNECEDORES
(NOME, COD_PRODUTO)
VALUES
('APPLE', 100),
('SAMSUNG', 110),
('SAMSUNG', 120)
```

Codigo Aula 2

```
-- NOT NULL não é uma constraint
--Schema: É a forma que o usuário é visto, quais as permissões, acessos,
visualizações das bases.
-- DBO: Data Base Owner
-- DBU: Data Base User

-- HOT KEY DO SP_HELP: DIGITA O NOME DA TABELA: TB_FUNC e ALT + F1

-- Primary Key(PK): 1º Não aceita valores duplicados,
--                  2º aumenta a performance da tabela
--                  3º Abre portas para relacionamentos

-- Após a identificação dos campos.
-- Sintaxe: CONSTRAINT <NOME_DA_PK> PRIMARY KEY (CAMPO_PK)
-- PK Resumida:

--Foreign Key(FK):
-- Sintaxe: CONSTRAINT <NOME_DA_FK> FOREIGN KEY (CAMPO_FK) REFERENCES
TABELA_MAE(CAMPO_MAE)

CREATE DATABASE DB_EMPRESA

USE DB_EMPRESA

-- COM SINTAXE COMPLETA
CREATE TABLE TB_FUNC
(
    ID_FUNC                INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF                    BIGINT NOT NULL,
    NOME                    VARCHAR(50) NOT NULL,
    SALARIO                 MONEY,
    DATA_NASC              DATE -- Recomendando: 'YYYY-MM-DD'

    CONSTRAINT PK_TB_FUNC_CPF PRIMARY KEY (CPF)
)

-- COM SINTAXE RESUMIDA DA PK
CREATE TABLE TB_FUNC
(
    ID_FUNC                INT IDENTITY(10,5), -- CAMPO DE AUTO NUMERAÇÃO
    CPF                    BIGINT NOT NULL PRIMARY KEY,
    NOME                    VARCHAR(50) NOT NULL,
    SALARIO                 MONEY,
    DATA_NASC              DATE -- Recomendando: 'YYYY-MM-DD'
)

DROP TABLE TB_FUNC

INSERT TB_FUNC
( CPF, NOME, SALARIO, DATA_NASC )
VALUES
( 101, 'LUCAS', 1000, '1994-04-14' )

SELECT * FROM TB_FUNC
```


SP_HELP TB_FUNC

-- COM SINTAXE COMPLETA

CREATE TABLE TB_DEPENDENTES

```
(
    ID_DEPENDENTE      INT IDENTITY (1000,2),
    CPF                BIGINT                        NOT
NULL,
    CPF_RESP           BIGINT                        NOT NULL,
    NOME               VARCHAR(50)                  NOT NULL,
    DATA_NASC         DATE
    CONSTRAINT PK_TB_DEPENDENTE_CPF                PRIMARY KEY (CPF),
    CONSTRAINT FK_TB_DEPENDENTE_CPF_RESP           FOREIGN KEY (CPF_RESP)
REFERENCES TB_FUNC(CPF)
)
```

-- COM SINTAXE RESUMIDA DA FK

CREATE TABLE TB_DEPENDENTES

```
(
    ID_DEPENDENTE      INT IDENTITY (1000,2),
    CPF                BIGINT                        NOT
NULL,
    CPF_RESP           BIGINT                        NOT NULL
FOREIGN KEY REFERENCES TB_FUNC(CPF),
    NOME               VARCHAR(50)                  NOT NULL,
    DATA_NASC         DATE
)
```

DROP TABLE TB_DEPENDENTES

INSERT TB_DEPENDENTES

(CPF, CPF_RESP, NOME, DATA_NASC)

VALUES

(1001, 101, 'TONY', '2018-05-21')

SELECT * FROM TB_FUNC

SELECT * FROM TB_DEPENDENTES

```

CREATE TABLE TB_PRODUTOS
(
    COD                INT IDENTITY(100, 10),
    NOME                VARCHAR(50) NOT NULL

    CONSTRAINT PK_TB_PRODUTOS_COD PRIMARY KEY (COD)
)

CREATE TABLE TB_FORNECEDORES
(
    COD_FORNECEDOR      INT IDENTITY,
    NOME                VARCHAR(50) NOT NULL,
    COD_PRODUTO         INT NOT NULL

    CONSTRAINT FK_TB_FORNECEDORES_COD_PRODUTOS FOREIGN KEY (COD_PRODUTO)
REFERENCES TB_PRODUTOS(COD)
)

INSERT TB_PRODUTOS
(NOME)
VALUES
('CELULAR'),
('TELEVISAO'),
('COMPUTADOR')

INSERT TB_FORNECEDORES
(NOME, COD_PRODUTO)
VALUES
('APPLE', 100),
('SAMSUNG', 110),
('SAMSUNG', 120)

SELECT * FROM TB_PRODUTOS
SELECT * FROM TB_FORNECEDORES

-- EMPREGADO: COD_FUNC, NOME, SALARIO, COD_DEPARTAMENTO --> Empregados em
departamento não existe (aqui tem a FK)
-- DEPARTAMENTO: COD, NOME_DEPARTAMENTO, SALARIO_INIC --> Departamento sem
empregado, pode existir (tabela mae)

-- TABELA~MÃE DE EMPREGADO
CREATE TABLE TB_DEPARTAMENTO
(
    COD                INT,
    NOME_DEPARTAMENTO  VARCHAR(50),
    SALARIO_INIC       MONEY

    CONSTRAINT PK_TB_DEPARTAMENTO_COD PRIMARY KEY (COD)
)

DROP TABLE TB_DEPARTAMENTO

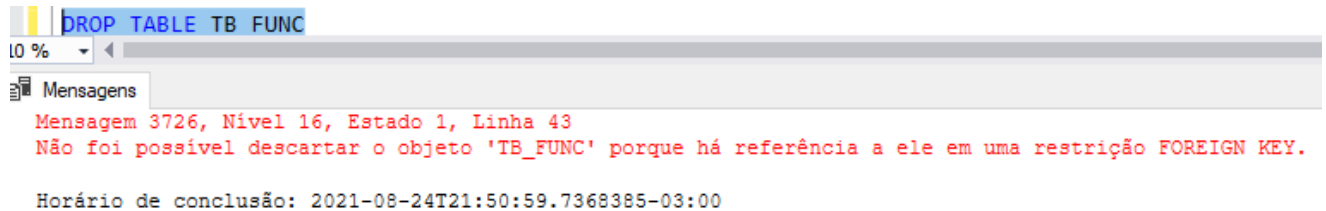
```

```
--TABELA FILHA DA DEPARTAMENTO
CREATE TABLE TB_EMPREGADO
(
    COD_FUNC          INT,
    NOME              VARCHAR(30),
    SALARIO            MONEY,
    COD_DEPARTAMENTO   INT

CONSTRAINT PK_TB_EMPREGADO_COD_FUNC          PRIMARY KEY (COD_FUNC),
CONSTRAINT FK_TB_EMPREGADO_COD_DEPARTAMENTO FOREIGN KEY (COD_DEPARTAMENTO)
REFERENCES TB_DEPARTAMENTO(COD)
)
```

Integridade

Não é possível deletar a mãe se houver uma filha vinculada.



Condicional (CHECK)

```
-- Valida uma informação antes de realizar o Insert
-- Sintaxe1: {COLUMN DEFINITION} CHECK (CONDIÇÃO)
-- Sintaxe2: CONSTRAINT <NOME_DA_CHECK> CHECK (CAMPO+CONDICAO)

CREATE TABLE TB_VENDAS
(
    PRODUTO                VARCHAR(100)
    ,
    QUANTIDADE              INT                CHECK(QUANTIDADE < 5001)
)

INSERT TB_VENDAS (PRODUTO, QUANTIDADE ) VALUES ('ASPIRADOR DE PÓ', 5500)

SELECT * FROM TB_VENDAS
```

Se rodar esse código , vai gerar um erro porque excedeu a condição.

```
Mensagens
Mensagem 547, Nível 16, Estado 0, Linha 17
A instrução INSERT conflitou com a restrição do CHECK "CK_TB_VENDAS_QUANT__16644E42". O conflito ocorreu no
A instrução foi finalizada.

Horário de conclusão: 2021-08-31T20:29:37.6693895-03:00
|
```

. O conflito ocorreu no banco de dados "master", tabela "dbo.TB_VENDAS", column 'QUANTIDADE'.

Para que esse erro não ocorra, é necessário que no INSERT respeite o valor máximo.

Exemplo de Check

```
CREATE TABLE TB_CLIENTES
(
    ID_CLIENTE          INT IDENTITY
    , NOME              VARCHAR(50) DEFAULT 'NOME NÃO FORNECIDO'
    , CNPJ              BIGINT NOT NULL
    , ENDERECO          VARCHAR(50) DEFAULT 'ENDEREÇO NÃO FORNECIDO'
    , DATA_NASCIMENTO  DATE CHECK (DATA_NASCIMENTO < '2003-08-31')
)
```

```
INSERT TB_CLIENTES
(NOME, CNPJ, ENDERECO, DATA_NASCIMENTO)
VALUES
('LUCAS', 123456789000140, 'RUA DOS BOBOS NUMERO ZERO', '1994-04-01')
```

```
SELECT *
FROM TB_CLIENTES
```

```
DROP TABLE TB_CLIENTES
```

ID_CLIENTE	NOME	CNPJ	ENDERECO	DATA_NASCIMENTO
1	LUCAS	123456789000140	RUA DOS BOBOS NUMERO ZERO	1994-04-01

Check com getDate

```
CREATE TABLE TB_CLIENTES
(
    ID_CLIENTE          INT IDENTITY
    , NOME              VARCHAR(50) DEFAULT 'NOME NÃO FORNECIDO'
    , CNPJ              BIGINT NOT NULL
    , ENDERECO          VARCHAR(50) DEFAULT 'ENDEREÇO NÃO FORNECIDO'
    , DATA_NASCIMENTO  DATE CHECK (DATA_NASCIMENTO < getDate())
)
```

```
INSERT TB_CLIENTES
(NOME, CNPJ, ENDERECO, DATA_NASCIMENTO)
VALUES
('LUCAS', 123456789000140, 'RUA DOS BOBOS NUMERO ZERO', '1994-04-01')
```

```
SELECT *
FROM TB_CLIENTES
```

```
DROP TABLE TB_CLIENTES
```

Erro na Sintaxe Completa

Coloque uma vírgula antes de declarar a constraint, é necessário colocar uma vírgula na coluna final antes da constraint.

-- Check com a Sintaxe Completa:

```
CREATE TABLE TB_VENDAS
```

```
(
```

```
    PRODUTO
```

```
    VARCHAR(100)
```

```
    QUANTIDADE
```

```
    INT
```

```
,
```

```
    CONSTRAINT CK_TB_VENDAS_QUANTIDADE CHECK (QUANTIDADE < 5500)
```

```
)
```

Importação de um Banco de Dados Completo

Baixe os arquivos disponibilizados:

DADOS (D:) > workspace > IMPACTA > sem2_Linguagem_SQL > importacao			
Nome	Data de modificação	Tipo	Tamanho
PEDIDOS_TABELAS.MDF	14/09/2021 20:23	SQL Server Databa...	11.008 KB
PEDIDOS_LOG.LDF	14/09/2021 20:23	SQL Server Databa...	1.024 KB
PEDIDOS_INDICES.NDF	14/09/2021 20:23	SQL Server Databa...	2.048 KB

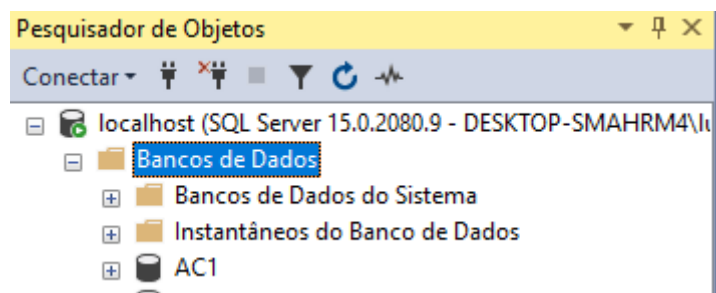
Agora, será necessário colocar no seguinte local:

C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA

Disco Local (C:) > Arquivos de Programas > Microsoft SQL Server > MSSQL15.MSSQLSERVER > MSSQL > DATA			
Nome	Data de modificação	Tipo	Tamanho
model.mdf	14/09/2021 20:27	SQL Server Databa...	8.192 KB
model_msdbdata.mdf	24/09/2019 15:09	SQL Server Databa...	13.696 KB
model_msdblog.ldf	24/09/2019 15:09	SQL Server Databa...	512 KB
model_replicatedmaster.ldf	24/09/2019 15:09	SQL Server Databa...	512 KB
model_replicatedmaster.mdf	24/09/2019 15:09	SQL Server Databa...	4.544 KB
modellog.ldf	14/09/2021 20:28	SQL Server Databa...	8.192 KB
MS_AgentSigningCertificate.cer	15/04/2021 23:36	Certificado de Seg...	1 KB
MS_AgentSigningCertificate2B67E015-8B...	06/08/2021 19:10	Certificado de Seg...	1 KB
MSDBData.mdf	14/09/2021 05:58	SQL Server Databa...	18.304 KB
MSDBLog.ldf	14/09/2021 20:28	SQL Server Databa...	29.504 KB
PEDIDOS_INDICES.NDF	14/09/2021 20:23	SQL Server Databa...	2.048 KB
PEDIDOS_LOG.LDF	14/09/2021 20:29	SQL Server Databa...	2.304 KB
PEDIDOS_TABELAS.MDF	14/09/2021 20:29	SQL Server Databa...	21.248 KB

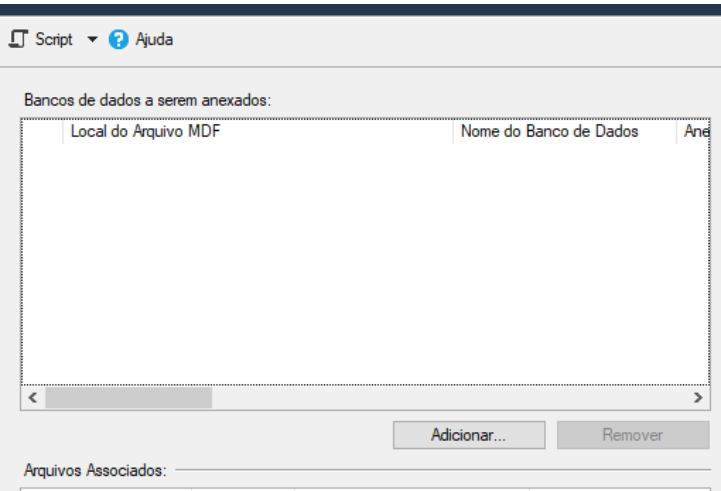
Adicione os 3 arquivos no DATA.

Volte ao SSMS e

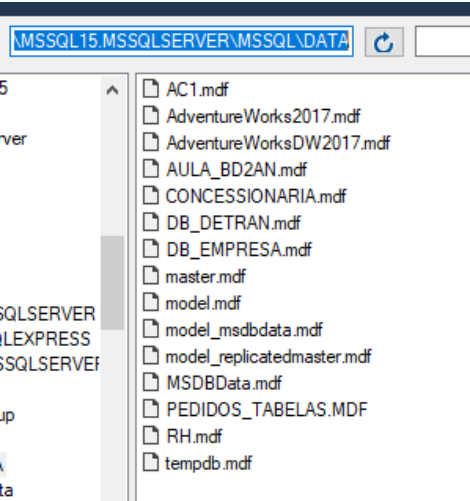


Anexar no Wizard do SQL

Com o botão direito: Banco de Dados > Anexar > Adicionar >

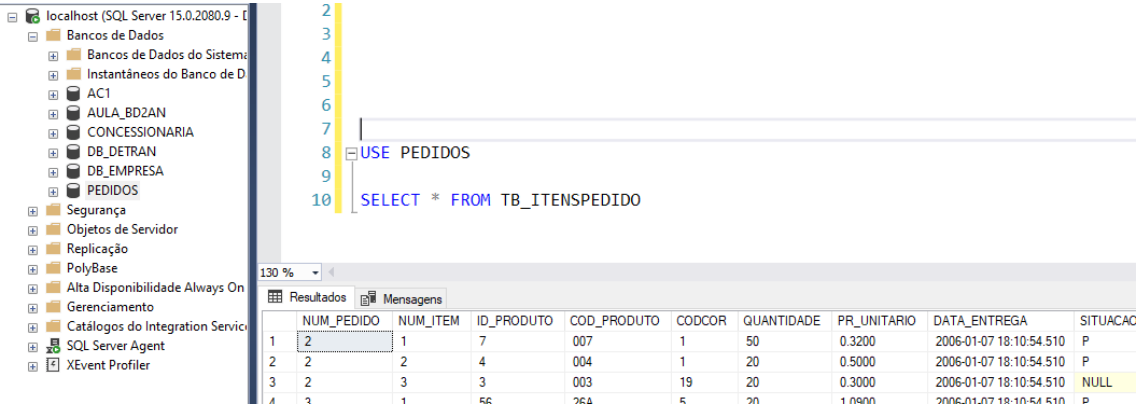


Clique agora no arquivo que seja a extensão: **.MDF**



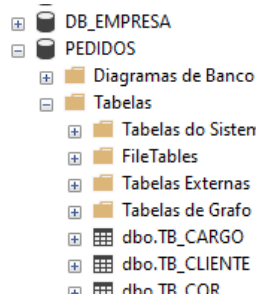
Nesse caso, o 'PEDIDOS_TABELA.MDF', e clique no 'OK'.

Lembre-se de atualizar no 'Pesquisador de Objetos' e utilizar a base:



Comandos do SELECT

Troque para o banco de dados da PEDIDOS



```
SELECT NOME, CNPJ, ESTADO, GETDATE() AS DATA_ATUAL
FROM TB_CLIENTE
```

	NOME	CNPJ	ESTADO	DATA_ATUAL
1	AUGUSTO'S FOLHINHAS LTDA	55230056000124	SP	2021-09-14 22:02:54.613
2	ASSIS BRINDES COMERCIO INDUSTRIA LTDA	47578117000110	SP	2021-09-14 22:02:54.613
3	AVEL APOLIMARIO VEICULOS S.A	59148452000168	SP	2021-09-14 22:02:54.613
4	ANTONIO M.DE SOUZA	NULL	PR	2021-09-14 22:02:54.613
5	ARISTON SERIGRAFIA ESTRUT.ART.PLAST.LTDA.-ME	12068136000112	PI	2021-09-14 22:02:54.613
6	APLIKE SERIGRAFIA LTDA	72095870000100	RS	2021-09-14 22:02:54.613
7	ANDRE DE CASTRO A.MALAQUIAS	02153405742	RJ	2021-09-14 22:02:54.613
8	ANDRADE E SILVA BRINDES PROMOCIONAIS	30000920830	SP	2021-09-14 22:02:54.613
9	ATILA CANGANI	06949735804	SP	2021-09-14 22:02:54.613
10	APARECIDO ALVES ME LTDA	82848432853	SP	2021-09-14 22:02:54.613

Traga todos os empregados que sejam sindicalizados

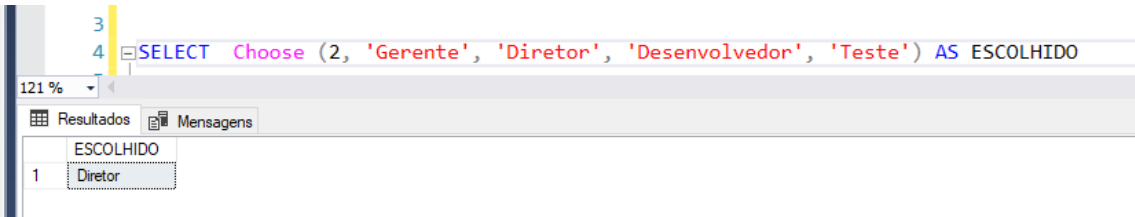
```
SP_HELP TB_EMPREGADO
```

```
SELECT NOME
FROM TB_EMPREGADO
WHERE SINDICALIZADO = 'S'
```

Choose

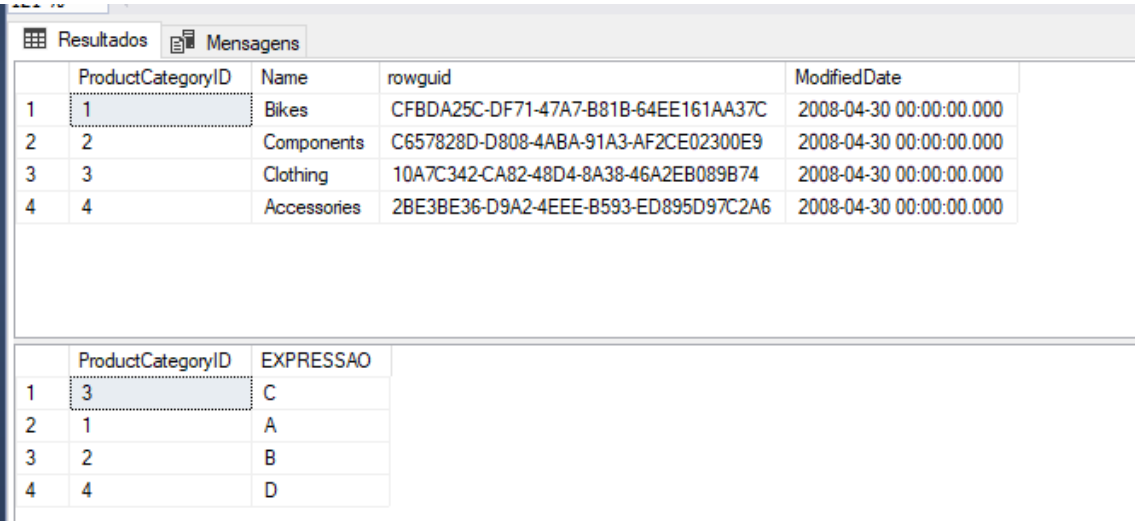
Escolhe dentro de uma “array” o valor pela opção

```
SELECT Choose (2, 'Gerente', 'Diretor', 'Desenvolvedor', 'Teste') AS ESCOLHIDO
```

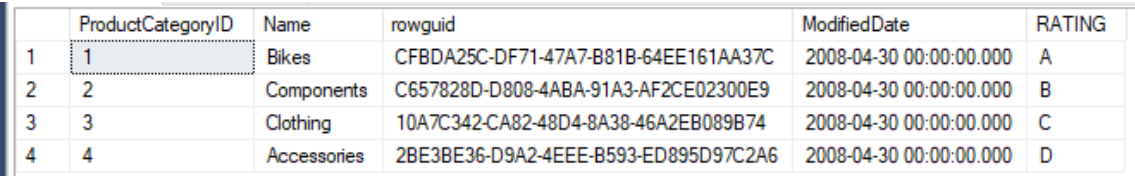


```
SELECT *
FROM Production.ProductCategory

SELECT ProductCategoryID,
CHOOSE (ProductCategoryID, 'A', 'B', 'C', 'D', 'E') AS EXPRESSAO
FROM Production.ProductCategory
```



```
SELECT *,
CHOOSE (ProductCategoryID, 'A', 'B', 'C', 'D', 'E') AS RATING
FROM Production.ProductCategory
```



Choose adicionando novas colunas

```
SELECT
    JobTitle,
    HireDate,
    MONTH(HireDate) mes,
    CHOOSE (MONTH(HireDate), 'Verão', 'Verão', 'Outono',
'Outono', 'Outono',
'Inverno',
'Inverno', 'Inverno', 'Primavera',
'Primavera',
'Primavera', 'Verão') Estacao_Contratado,
    CHOOSE (MONTH(HireDate), 'Janeiro', 'Fevereiro',
'Março', 'Abril',
'Maio', 'Junho',
'Julho', 'Agosto',
'Setembro',
'Outubro', 'Novembro', 'Dezembro')
    MES_CONTRATACAO
FROM HumanResources.Employee
WHERE YEAR(HireDate) > 2005
ORDER BY YEAR(HireDate)
```

	JobTitle	HireDate	mes	Estacao_Contratado	MES_CONTRATACAO
1	Production Technician - WC60	2006-06-30	6	Inverno	Junho
2	Production Supervisor - WC60	2007-12-26	12	Verão	Dezembro
3	Engineering Manager	2007-11-11	11	Primavera	Novembro
4	Senior Tool Designer	2007-12-05	12	Verão	Dezembro
5	Tool Designer	2007-12-11	12	Verão	Dezembro
6	Marketing Manager	2007-12-20	12	Verão	Dezembro
7	Marketing Assistant	2007-01-26	1	Verão	Janeiro
8	Design Engineer	2008-01-06	1	Verão	Janeiro
9	Design Engineer	2008-01-24	1	Verão	Janeiro
10	Research and Development ...	2008-12-29	12	Verão	Dezembro

IIF

Retornará um dos dois valores (True or False), dependendo da expressão que foi proposta e será avaliada:

IIF (expressão, valor_se_verdadeiro, valor_se_falso)

```
DECLARE          @a INT = 45,  
                  @b INT = 40  
  
SELECT           @a VALOR_A,  
                  @b VALOR_B,  
                  IIF (@a > @b, 'Maior', 'Menor') as RESULTADO
```

	VALOR_A	VALOR_B	RESULTADO
1	45	40	Maior

IIF Com Instrução Nulas

Passando como parâmetro para True or False os resultados o NULL, gera erro.

```
42  
43 SELECT IIF (45 > 30, NULL, NULL) AS RESULTADO
```

110 %

Mensagens

Mensagem 8133, Nível 16, Estado 1, Linha 43
Pelo menos uma das expressões resultantes em uma especificação CASE deve ser uma expressão diferente da constante NULL.

Horário de conclusão: 2021-12-09T15:11:15.5031575-03:00

IIF Com Parâmetros Nulos

Diferente da instrução, passando como um parâmetro o Nulo, não gera erro.

```
DECLARE          @p INT = NULL,  
                  @s INT = NULL  
  
SELECT IIF(45 > 30, @p, @s) AS RESULTADO_NULO
```

	RESULTADO_NULO
1	NULL

Funções de Classificação

Funcao	Campos
RANK	As funções de classificação retornam um valor de classificação para cada linha em uma partição. Dependendo da função usada, algumas linhas podem receber o mesmo valor que outras. As funções de classificação são não determinísticas
NTILE	Distribui as linhas de uma partição ordenada em um número de grupos especificado. Os grupos são numerados, iniciando em um. Para cada linha, NTILE retorna o número do grupo ao qual a linha pertence.
DENSE_RANK	Retorna a classificação de linhas dentro da partição de um conjunto de resultados, sem qualquer lacuna na classificação. A classificação de uma linha é um mais o número de classificações distintas que vêm antes da linha em questão.
ROW_NUMBER	Retorna o número sequencial de uma linha em uma partição de um conjunto de resultados, iniciando em 1 para a primeira linha de cada partição.

Funções Matemáticas

Funcao	Campos
ABS	Uma função matemática que retorna o valor absoluto (positivo) da expressão numérica especificada.
RAND	Retorna um valor float pseudoaleatório de 0 a 1, exclusivo.
ROUND	Retorna um valor numérico, arredondado, para o comprimento ou precisão especificados.
POWER	Retorna o valor da expressão especificada elevada à potência especificada.
SQRT	Retorna a raiz quadrada do valor flutuante especificado.

ABS

Transforma em valores absolutos e negativos em absolutos positivos.

SELECT

ABS(-1.0), ABS(0.0), ABS(-9.4), ABS(9.456), ABS(1.3)

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	1.0	0.0	9.4	9.456	1.3

RAND

Gera um número pseudo aleatório de 0 a 1.

SELECT

RAND(), RAND(), RAND()

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	0.467763592665771	0.894482048860037	0.480945875472888

ROUND

Arredonda valores após a casa decimal.

```
SELECT      ROUND(123.9994, 3), -- NA 3º CASA DECIMAL
            ROUND(123.9995, 3)  -- NA 3º CASA DECIMAL
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	123.9990	124.0000

```
SELECT      ROUND(123.9994, 2), -- NA 2º CASA DECIMAL
            ROUND(123.9995, 2), -- NA 2º CASA DECIMAL
            ROUND(139.997, 1)
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	124.0000	124.0000	140.000

Exatamente após a vírgula

```
SELECT      ROUND(150.75, 0) -- Arredonda exatamente após a vírgula.
SELECT      ROUND(150.45, 0 )
SELECT      ROUND(150.75, 0, 1)
```

	(Nenhum nome de coluna)
1	151.00

	(Nenhum nome de coluna)
1	150.00

	(Nenhum nome de coluna)
1	150.00

Arredondamento da Casa Decimal na Dezena e na Centena

```
SELECT ROUND(123.45, -2) -- MAIS PRÓXIMO DO 100 (CENTENA)
SELECT ROUND(193.45, -2) -- MAIS PRÓXIMO DO 200 (CENTENA)
SELECT ROUND(149.45, -2) -- 100
SELECT ROUND(151.45, -2) -- 200
SELECT ROUND(1141.45, -3) -- (3 - MILHAR)
```

	(Nenhum nome de coluna)	
1	100.00	
	(Nenhum nome de coluna)	
1	200.00	
	(Nenhum nome de coluna)	
1	100.00	
	(Nenhum nome de coluna)	
1	200.00	
	(Nenhum nome de coluna)	
1	1000.00	

POWER - Potência

Necessário dois parâmetros para elevar a potência do primeiro.

```
SELECT
    POWER(2,2),
    POWER(2,3),
    POWER(5,2)
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	4	8	25

Raíz Quadrada (SQRT)

```
SELECT SQRT(81),
    SQRT(25),
    SQRT(27),
    SQRT(8)
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	9	5	5,19615242270663	2,82842712474619

Funções de Limite

Funcao	Campos
TOP	A cláusula SELECT TOP é útil em grandes tabelas com milhares de registros. Atenção! Retornar uma grande quantidade de registros pode afetar o desempenho.

```
SELECT TOP 10 *  
FROM campeonato  
ORDER BY pontos ASC -- OS ÚLTIMOS COLADOS APENAS
```

	nometime	pontos
1	Atlético-GO	22
2	Vitória	26
3	Bahia	27
4	São Paulo	27
5	Coritiba	27
6	Ponte Preta	28
7	Chapecoense	28
8	Sport	29
9	Avaí	29
10	Vasco	31

Limite com Rank

```
SELECT TOP 10 *  
FROM , RANK() OVER(ORDER BY populacao DESC) as posicao  
ORDER BY populacao DESC
```

	uf	cod_uf	cod_mun	nome_mun	populacao	posicao
1	SP	35	50308	São Paulo *	10886518	1
2	RJ	33	04557	Rio de Janeiro *	6093472	2
3	BA	29	27408	Salvador *	2892625	3
4	DF	53	00108	Brasília *	2455903	4
5	CE	23	04400	Fortaleza *	2431415	5

Funções de Conversão

Funções que convertem dados.

Só é possível fazer a conversão, desde que a conversão da saída possa ser retornada e convertida novamente no seu estado original.

Funcao	Campos
CAST	Converte uma expressão de um tipo de dados em outro
CONVERT	
PARSE	Retorna o resultado de uma expressão, convertida no tipo de dados solicitado no SQL Server.
TRY_CAST	Retorna uma conversão de valor ao tipo de dados especificado se a conversão for bem-sucedida; caso contrário, retorna nulo
TRY_CONVERT	Retorna uma conversão de valor ao tipo de dados especificado se a conversão for bem-sucedida; caso contrário, retorna nulo.
TRY_PARSE	Retorna o resultado de uma expressão, convertido no tipo de dados solicitado, ou nulo se a conversão falha no SQL Server. Use TRY_PARSE somente para converter da cadeia de caracteres em data/hora e tipos numéricos.

```
DECLARE @valor DECIMAL(5, 2) -- 5 DIGITOS[== COM 2 CASAS DECIMAIS
SET @valor = 193.57 -- PASSANDO O VALOR DA VARIÁVEL
```

```
-- Usando o CAST
-- CAST(valor as tipo_a_converter)
SELECT CAST(@valor AS varbinary(20))
SELECT CAST(CAST(@valor AS varbinary(20)) AS DECIMAL(5,2))

-- Usando o CONVERT
-- CONVERT(tipo_a_converter, valor).
SELECT CONVERT(VARBINARY(20), @valor)
SELECT CONVERT(DECIMAL(5,2), CONVERT(VARBINARY(20), @valor))
```

	(Nenhum nome de coluna)
1	0x050200019D4B0000

	(Nenhum nome de coluna)
1	193.57

	(Nenhum nome de coluna)
1	0x050200019D4B0000

	(Nenhum nome de coluna)
1	193.57

```

SELECT          CAST(ROUND(SalesYTD / CommissionPct, 0) as INT) as FORMATADO,
                (SalesYTD / CommissionPct)                    as nao_formatado
FROM            sales.SalesPerson
WHERE           CommissionPct <> 0

```

	FORMATADO	nao_formatado
1	313598182	313598181,5583
2	283424570	283424569,98
3	212627891	212627891,08
4	145371947	145371946,53
5	231518561	231518561,10
6	135257713	135257713,25

Cast com Concatenação

Ao tentar concatenar um tipo String/Texto com um tipo de dado de Money, gera erro, por isso, transforma(Cast) para um texto.

```

SELECT          DISTINCT 'A lista de preços é: ' + CAST(listprice AS VARCHAR(12))
                as LISTA_PREÇO
FROM            Production.Product
WHERE           ListPrice BETWEEN 350.00 AND 400.00

```

	LISTA_PREÇO
1	A lista de preços é: 357.06
2	A lista de preços é: 364.09

Cast simplificando palavras

Servirá para suprimir os textos.

```

SELECT          DISTINCT p.NAME,
                CAST(p.NAME AS CHAR(10)) AS NOME_TRATADO,
                s.unitprice
FROM            sales.SalesOrderDetail AS s
JOIN            Production.Product    AS p
ON              s.ProductID = p.ProductID
WHERE           p.Name LIKE '%Long-Sleeve Logo Jersey%'

```

	NAME	NOME_TRATADO	unitprice
1	Long-Sleeve Logo Jersey, L	Long-Sleev	24,0337
2	Long-Sleeve Logo Jersey, L	Long-Sleev	26,437
3	Long-Sleeve Logo Jersey, L	Long-Sleev	27,4945
4	Long-Sleeve Logo Jersey, L	Long-Sleev	27,879
5	Long-Sleeve Logo Jersey, L	Long-Sleev	28,8404
6	Long-Sleeve Logo Jersey, L	Long-Sleev	28,9942

Try Cast ou Try_Convert

Ela retorna um resultado dando certo ou não.

Vale lembrar que quando o **CAST** falhar, ele retorna um valor **NULO**.

```
SELECT      CASE
              WHEN TRY_CAST('TESTE' AS FLOAT) IS NULL THEN 'Cast Falhou!'
              ELSE 'Cast Ok, Funcionou'
            END AS RESULTADO_TRY_CAST
```

Vai falhar porque não é possível converter STRING em FLOAT.

	RESULTADO_TRY_CAST
1	Cast Falhou!

```
SELECT
              WHEN TRY_CAST(52 AS FLOAT) IS NOT NULL THEN 'CAST Ok, funcionou'
              ELSE 'Cast Falhou!'
            END AS RESULTADO_TRY_CAST
```

	RESULTADO_TRY_CAST
1	CAST Ok, funcionou

```
SELECT
      CASE
        WHEN TRY_CONVERT(FLOAT, 65) IS NOT NULL THEN 'TRY_CONVERT SUCESSO'
        ELSE 'TRY_CONVERT FALHOU'
      END AS RESULTADO
```

	RESULTADO
1	TRY_CONVERT SUCESSO

Parse

Tratamento de dados para enviar o tipo de dado corretamente para uma base.

Exemplo, foi recebido uma data em formato de texto. Convertemos para 'datetime' na **cultura** específica que precisa ser enviada.

```
SELECT PARSE('Monday, 13 December 2021' AS datetime USING 'en-US') AS  
RESULTADO  
SELECT PARSE('Monday, 13 December 2021' AS datetime USING 'pt-BR') AS  
RESULTADO  
SELECT PARSE('Segunda-feira, 13 Dezembro 2021' AS datetime USING 'pt-BR')  
AS RESULTADO
```

	RESULTADO
1	2021-12-13 00:00:00.000

	RESULTADO
1	2021-12-13 00:00:00.000

	RESULTADO
1	2021-12-13 00:00:00.000

Parse com money

Serve para **remover** o CÍFRÃO, por exemplo.

```
SELECT PARSE('R$345,98' AS MONEY USING 'pt-BR') AS RESULTADO_SEM_CIFRAO  
SELECT PARSE('$345.98' AS MONEY USING 'en-US') AS RESULTADO_SEM_CIFRAO
```

	RESULTADO_SEM_CIFRAO
1	345,98

	RESULTADO_SEM_CIFRAO
1	345,98

Parse com Datas

```
SET LANGUAGE 'English'  
SELECT      PARSE('12/20/2021' AS datetime) AS RESULTADO
```

```
SET LANGUAGE 'Português'  
SELECT      PARSE('20/12/2021' AS datetime) AS RESULTADO
```

	RESULTADO
1	2021-12-20 00:00:00.000

	RESULTADO
1	2021-12-20 00:00:00.000

Funções de Caracteres

São elas:

Função	Significado
ASCII	Retorna o valor do código ASCII do caractere mais à esquerda de uma expressão de caractere
LTRIM	Retorna uma expressão de caractere depois de remover espaços em branco à esquerda
RTRIM	Retorna uma cadeia de caracteres depois de truncar todos os espaços em branco à direita.
STR	Retorna dados de caractere convertidos de dados numéricos
CONCAT	Retorna uma cadeia de caracteres que é o resultado da concatenação de dois ou mais valores.
CONCAT_WS	Separa e retornar valores de cadeia de caracteres concatenados com o delimitador especificado no argumento da primeira função.
REPLACE	Substitui todas as ocorrências de um valor da cadeia de caracteres especificado por outro valor de cadeia de caracteres.
REPLICATE	Repete um valor da cadeia de caracteres um número especificado de vezes.
LEFT	Retorna a parte da esquerda de uma cadeia de caracteres com o número de caracteres especificado.
RIGHT	Retorna a parte da direita de uma cadeia de caracteres com o número de caracteres especificado.
UPPER	Retorna uma expressão de caractere com dados de caractere em minúsculas convertidos em maiúsculas.
LOWER	Retorna uma expressão de caractere com dados de caractere em maiúsculas convertidos em minúsculas.
REVERSE	Retorna a ordem inversa de um valor da cadeia de caracteres.
LEN	Retorna o número de caracteres da expressão da cadeia de caracteres especificada, excluindo espaços em branco à direita.
DATALENGHT	Retorna o número de bytes usado para representar qualquer expressão

ASC

Retorna o valor ASCII de um único caractere.

```
SELECT      ASCII('S'), ASCII('Q'), ASCII('L')
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	83	81	76

LTRIM

Remove quaisquer espaços a esquerda.

```
DECLARE @string_to_trim VARCHAR(60)
SET      @string_to_trim = '      Cinco espaços no início'

SELECT 'Texto sem espaço:' + LTRIM(@string_to_trim)
SELECT 'Texto com espaço:' + @string_to_trim
```

	(Nenhum nome de coluna)
1	Texto sem espaço:Cinco espaços no início

	(Nenhum nome de coluna)
1	Texto com espaço: Cinco espaços no início

CONCAT

```
SELECT CONCAT(CURRENT_USER,
              'SEU SALDO É R$ ',
              11.00,
              ' EM ',
              DAY(GETDATE()), '/',
              MONTH(GETDATE()), '/',
              YEAR(GETDATE())) AS RESULTADO
```

	RESULTADO
1	dboSEU SALDO É R\$ 11.00 EM 10/12/2021

CONCAT_WS

Serve para realizar uma concatenação com um delimitador entre os campos

```
SELECT      CONCAT_WS('|', a.primeiro_nome, a.ultimo_nome, a.email,  
a.nascimento, a.sexo)  
FROM        cliente a
```

	(Nenhum nome de coluna)
1	Channa Kelway ckelway0@cocolog-nifty.com 1971-08-0...
2	Conny Tilliards ctilliards1@forbes.com 2004-07-07 Female
3	Blondie Boulstridge bboulstridge2@webden.co.uk 2008...
4	Ewen Fulton efulton3@go.com 1960-03-15 Male
5	Jasun Heathom jheathom4@wikia.com 1984-09-14 Male

REPLACE

Dada uma cadeia de caracteres, selecione quais serão substituídas por outra.

```
SELECT      'abcdefghij' DE,  
            REPLACE ('abcdefghij', 'cde', 'xxx') PARA
```

	DE	PARA
1	abcdefghij	abxxfghij

REPLICATE

Serve muito para adicionar zeros (0) em números.

Sintaxe: **REPLICATE**('alguma_coisa', quantas_vezes)

```
SELECT      name,  
            productline,  
            REPLICATE('0', 4) + productline AS 'CODIGO'  
FROM        Production.Product  
WHERE       productline = 'T'
```

	name	productline	CODIGO
1	Touring Front Wheel	T	0000T
2	Touring Rear Wheel	T	0000T
3	Touring-Panniers, Large	T	0000T
4	HL Touring Frame - Yellow, 60	T	0000T
5	LL Touring Frame - Yellow, 62	T	0000T
6	HL Touring Frame - Yellow, 46	T	0000T

LEFT ou RIGHT

Admite a “obtenção” de alguns caracteres

```
SELECT      NAME,  
            LEFT(NAME, 5) as apenas_cinco  
FROM        Production.Product
```

	NAME	apenas_cinco
1	Adjustable Race	Adjus
2	All-Purpose Bike Stand	All-P
3	AWC Logo Cap	AWC L
4	BB Ball Bearing	BB Ba
5	Bearing Ball	Beari
6	Bike Wash - Dissolver	Bike

UPPER

Receberá uma cadeia de caracteres e os colocará em maiúsculo.

```
SELECT      NAME,  
            UPPER(NAME)  
FROM        Production.Product
```

	NAME	(Nenhum nome de coluna)
1	Adjustable Race	ADJUSTABLE RACE
2	All-Purpose Bike Stand	ALL-PURPOSE BIKE STAND
3	AWC Logo Cap	AWC LOGO CAP
4	BB Ball Bearing	BB BALL BEARING

SUBSTRING

Receberá uma cadeia de caracteres, e com base nela, inicie na posição X e vá até mais a posição Y.

```
SELECT      lastname AS nome,  
            SUBSTRING(lastname, 1, 3) AS LASTNAME1,  
            SUBSTRING(lastname, 4, 10) AS LASTNAME2  
FROM        Person.Person  
ORDER BY   LastName DESC
```

3	Zukowski	Zuk	owski
4	Zugelder	Zug	elder
5	Zubaty	Zub	aty
6	Zubaty	Zub	aty
7	Zimprich	Zim	prich
8	Zimprich	Zim	prich
9	Zimmernan	Zim	meman
10	Zimmernan	Zim	meman

Reverse

Reverte uma ordem de caractere.

```
SELECT      firstname,  
            REVERSE(firstname) AS REVERTIDO  
FROM        Person.Person  
WHERE       BusinessEntityID < 5
```

	firstname	REVERTIDO
1	Ken	neK
2	Teri	ireT
3	Roberto	otreboR
4	Rob	boR

LEN

Retorna o tamanho de caracteres da expressão.

```
SELECT      firstname,  
            LEN(firstname) AS TAMANHO_NAME  
FROM        Sales.vIndividualCustomer  
WHERE       CountryRegionName = 'AUSTRALIA'
```

	firstname	TAMANHO_NAME
1	Amber	5
2	Devin	5
3	Isabella	8
4	Savannah	8
5	Sydney	6
6	Bob	3
7	Jamie	5

DATALENGTH

Retorna o número de bytes para representar uma expressão. Quantos BYTES são armazenados em cada um. Quantos BYTES uma coluna está armazenando.

```
SELECT      NAME,  
            DATALENGTH(NAME) AS DATAD  
FROM        Production.Product
```

	NAME	DATAD
1	Adjustable Race	30
2	All-Purpose Bike Stand	44
3	AWC Logo Cap	24
4	BB Ball Bearing	30
5	Ball Bearing	24

Funções de Data e Hora

Funções que retornam a data e hora onde a instância está instalado.

Função	Sintaxe	Descrição	Retorno
<u>SYSDATETIME</u>	<u>SYSDATETIME ()</u>	Retorna um valor de datetime2(7) que contém a data e a hora do computador no qual a instância do SQL Server está sendo executada. O deslocamento de fuso horário não está incluído.	datetime2(7)
SYSDATETIMEOFFSET	SYSDATETIMEOFFSET ()	Retorna um valor datetimeoffset(7) que contém a data e a hora do computador no qual a instância de SQL Server está sendo executada. O deslocamento de fuso horário está incluído.	datetimeoffset(7)
SYSUTCDATETIME	SYSUTCDATETIME ()	Retorna um valor datetime2(7) que contém a data e a hora do computador no qual a instância de SQL Server está sendo executada. A data e hora é retornada como hora UTC (Tempo Universal Coordenado).	datetime2(7)
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP	Retorna um valor datetime que contém a data e a hora do computador no qual a instância de SQL Server está sendo executada. O deslocamento de fuso horário não está incluído.	Datetime
GETDATE	GETDATE ()	Retorna um valor datetime que contém a data e a hora do computador no qual a instância de SQL Server está sendo executada. O deslocamento de fuso horário não está incluído.	datetime
GETUTCDATE	GETUTCDATE ()	Retorna um valor datetime que contém a data e a hora do computador no qual a instância de SQL Server está sendo executada. A data e hora é retornada como hora UTC (Tempo Universal Coordenado).	datetime

Data e Hora

```
SELECT      SYSDATETIME ( ) ExSYSDATETIME ,
            SYSDATETIMEOFFSET ( ) ExSYSDATETIMEOFFSET ,
            SYSUTCDATETIME ( ) ExSYSUTCDATETIME ,
            CURRENT_TIMESTAMP ExCURRENT_TIMESTAMP ,
            GETDATE ( ) ExGetDATE ,
            GETUTCDATE ( ) ExGETUTCDATE
```

	ExSYSDATETIME	ExSYSDATETIMEOFFSET	ExSYSUTCDATETIME	ExCURRENT_TIMESTAMP	ExGetDATE	ExGETUTCDATE
1	2021-12-13 10:46:45.0911922	2021-12-13 10:46:45.0911922 -03:00	2021-12-13 13:46:45.0911922	2021-12-13 10:46:45.090	2021-12-13 10:46:45.090	2021-12-13 13:46:45.090

Hora do Sistema:



DATEPART, DATENAME, DATETIME, DATETIMEFROMPART

Funções que obtêm parte dos valores de data e hora do sistema

Datepart { year, month, day, hour, minute, seconds, milliseconds }

Função	Sintaxe	Descrição	Retorno
DATENAME	DATENAME (datepart , date)	Retorna uma cadeia de caracteres que representa o datepart específico da data especificada.	nvarchar
DATEPART	DATEPART (datepart , date)	Retorna um inteiro que representa a datepart especificada da date especificada.	int
DAY	DAY (date)	Retorna um inteiro que representa a parte do dia da date especificada.	int
MONTH	MONTH (date)	Retorna um inteiro que representa a parte do mês de uma date especificada.	int
YEAR	YEAR (date)	Retorna um inteiro que representa a parte do ano da date especificada.	Int
DATETIMEFROMPARTS	DATETIMEFROMPARTS (year, month, day, hour, minute, seconds, milliseconds)	Retorna um valor datetime para a data e a hora especificadas	datetime

```
SELECT      GETDATE() DATA_HORA,
            YEAR(GETDATE()) GET_DATE_YEAR,  --UTILIZAR ESSA FUNÇÃO SEMPRE
            DATENAME (YEAR, GETDATE()) DATENAME_YEAR
```

	DATA_HORA	GET_DATE_YEAR	DATENAME_YEAR
1	2021-12-13 10:53:48.447	2021	2021

```
SELECT      DATETIMEFROMPARTS(2017, 11, 30, 3, 45, 59, 1) HORA
```

	HORA
1	2017-11-30 03:45:59.000

SELECT

```
YEAR(GETDATE()), -- RETORNA O SISTEMA  
DATENAME(YEAR, a.nascimento), -- MAIS CODIGO  
YEAR(a.nascimento) -- MAIS EFICIENTE, DATA DO NASCIMENTO
```

FROM

cliente a

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	2021	1971	1971
2	2021	2004	2004
3	2021	2008	2008
4	2021	1960	1960
5	2021	1984	1984
6	2021	1987	1987

Exemplo de Agrupamento em Datas

SELECT

```
MONTH(a.nascimento),  
count(1) AS QTD
```

FROM

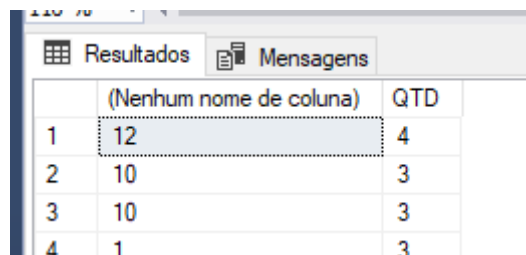
cliente a

GROUP BY

nascimento

ORDER BY

QTD DESC



	(Nenhum nome de coluna)	QTD
1	12	4
2	10	3
3	10	3
4	1	3

DATEDIFF

Funções que obtêm a diferença de data e hora do sistema

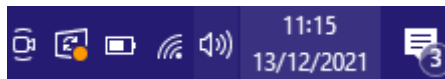
Função	Sintaxe	Descrição	Retorno
DATEDIFF	DATEDIFF (datepart ,startdate , enddate)	Retorna o número de limites de datepart de data ou hora entre duas datas especificadas.	O tipo de dados do argumento date.

Funções que Adicionam intervalos de data e hora do sistema

Função	Sintaxe	Descrição	Retorno
DATEADD	DATEADD (datepart , number , date)	Retorna um novo valor datetime adicionando um intervalo à datepart especificada da date especificada..	O tipo de dados do argumento date.

DATEDIFF é a diferença, quantidade de dias, meses e anos de diferença entre duas datas.

Data do Sistema:



```
SELECT      DATEDIFF(day, '2021-12-11', '2021-12-13') AS DIAS,
            DATEDIFF(month, '2021-07-11', '2021-12-13') AS MESES,
            DATEDIFF(year, '2020-12-11', '2021-12-13') AS ANOS
```

	DIAS	MESES	ANOS
1	2	5	1

```
SELECT      DATEDIFF (DAY, '1994-04-14', GETDATE())
```

DATEADD

A partir de uma data específica, adicione x dias a mais.

```
SELECT          GETDATE() AGORA,  
                DATEADD(day, 90, GETDATE()) AS DAQUI_90
```

	Resultados	Mensagens
	AGORA	DAQUI_90
1	2021-12-13 11:57:15.253	2022-03-13 11:57:15.253

EXEMPLO PRÁTICO

Primeiro, crie uma tabela.

```
CREATE TABLE CONTAS
(
    id_conta          INT IDENTITY (1,1) NOT NULL PRIMARY KEY,
    data_conta        DATE                NOT NULL,
    dias              INT,
    data_vencimento    DATE,
    valor             DECIMAL (10, 2),
    situacao          VARCHAR(1)          DEFAULT ('A')
)
```

Insira alguns valores para popular a tabela.

```
INSERT INTO      CONTAS (data_conta, dias, valor)
VALUES           ('2018-05-24', 10, 25.75),
                 ('2018-05-25', 30, 40.75),
                 ('2018-05-24', 60, 30.75)
```

Veja como está, queremos descobrir o VENCIMENTO.

```
SELECT          *
FROM            CONTAS
```

	id_conta	data_conta	dias	data_vencimento	valor	situacao
1	1	2018-05-24	10	NULL	25.75	A
2	2	2018-05-25	30	NULL	40.75	A
3	3	2018-05-24	60	NULL	30.75	A

Atráves de um SELECT para verificarmos se a aplicação do DATEADD será feita corretamente.

```
SELECT          id_conta,
                data_conta,
                dias,
                DATEADD(day,dias, data_conta) vencito,
                valor
FROM            CONTAS
```

	id_conta	data_conta	dias	vencito	valor
1	1	2018-05-24	10	2018-06-03	25.75
2	2	2018-05-25	30	2018-06-24	40.75
3	3	2018-05-24	60	2018-07-23	30.75

Uma vez que está correto, é possível adicionar essa informação na tabela através um .

UPDATE CONTAS

SET data_vencimento = **DATEADD**(day, dias, data_conta)

WHERE data_vencimento IS NULL

	id_conta	data_conta	dias	data_vencimento	valor	situacao
1	1	2018-05-24	10	2018-06-03	25.75	A
2	2	2018-05-25	30	2018-06-24	40.75	A
3	3	2018-05-24	60	2018-07-23	30.75	A

Formatação de Datas

SQL Statement	Saida
SELECT SUBSTRING(CONVERT(VARCHAR(10), GETDATE(), 120), 3, 8) AS [YY-MM-DD]	99-01-24
SELECT REPLACE(CONVERT(VARCHAR(8), GETDATE(), 11), '/', '-') AS [YY-MM-DD]	
SELECT CONVERT(VARCHAR(10), GETDATE(), 120) AS [YYYY-MM-DD]	1999-01-24
SELECT REPLACE(CONVERT(VARCHAR(10), GETDATE(), 111), '/', '-') AS [YYYY-MM-DD]	
SELECT RIGHT(CONVERT(VARCHAR(8), GETDATE(), 3), 5) AS [MM/YY]	08/99
SELECT SUBSTRING(CONVERT(VARCHAR(8), GETDATE(), 3), 4, 5) AS [MM/YY]	
SELECT RIGHT(CONVERT(VARCHAR(10), GETDATE(), 103), 7) AS [MM/YYYY]	12/2005
SELECT CONVERT(VARCHAR(5), GETDATE(), 11) AS [YY/MM]	99/08
SELECT CONVERT(VARCHAR(7), GETDATE(), 111) AS [YYYY/MM]	2005/12

Utiliza um “padrão” x de formatação, utiliza um padrão 120, por exemplo 99-01-24 (yy-mm-dd).

```

SELECT      CONVERT(VARCHAR(10), GETDATE(), 103)
SELECT      CONVERT(VARCHAR(5), GETDATE(), 103)
SELECT      CONVERT(VARCHAR(10), GETDATE(), 108)
SELECT      CONVERT(VARCHAR(5), GETDATE(), 108)
SELECT      CONVERT(VARCHAR(20), GETDATE(), 100)
-- SELECT    CONVERT(GETDATE(), 100) ERRADO
SELECT      CONVERT(VARCHAR(8), GETDATE(), 1)

```

	(Nenhum nome de coluna)
1	13/12/2021
	(Nenhum nome de coluna)
1	13/12
	(Nenhum nome de coluna)
1	13:25:49
	(Nenhum nome de coluna)
1	13:25
	(Nenhum nome de coluna)
1	Dez 13 2021 1:25PM
	(Nenhum nome de coluna)
1	12/13/21

```

SELECT      CAST(DAY(GETDATE()) AS VARCHAR(2)) + '/' +
            DATENAME(MM, GETDATE()) AS [Dia e Mes]

```

	Dia e Mes
1	13/Dezembro

Exemplos de Padrões Aplicados nas Tabelas

```
SELECT      a.nascimento,
            --      Padrão 120: YYYY-MM-DD
            CONVERT(VARCHAR(10), a.nascimento, 120) AS PADRÃO_120,
            CONVERT(VARCHAR(10), a.nascimento, 103) AS PADRÃO_103,
            CONVERT(VARCHAR(20), a.nascimento, 100) AS PADRÃO_100,
            CONVERT(VARCHAR(11), a.nascimento, 100) AS PADRÃO_100,
            CONVERT(VARCHAR(8), a.nascimento, 1) AS PADRÃO_1
FROM cliente a
```

	nascimento	PADRÃO_120	PADRÃO_103	PADRÃO_100	PADRÃO_100	PADRÃO_1
1	1971-08-07	1971-08-07	07/08/1971	Ago 7 1971	Ago 7 1971	08/07/71
2	2004-07-07	2004-07-07	07/07/2004	Jul 7 2004	Jul 7 2004	07/07/04
3	2008-12-31	2008-12-31	31/12/2008	Dez 31 2008	Dez 31 2008	12/31/08
4	1960-03-15	1960-03-15	15/03/1960	Mar 15 1960	Mar 15 1960	03/15/60
5	1984-09-14	1984-09-14	14/09/1984	Set 14 1984	Set 14 1984	09/14/84
6	1987-06-14	1987-06-14	14/06/1987	Jun 14 1987	Jun 14 1987	06/14/87
7	1961-02-23	1961-02-23	23/02/1961	Fev 23 1961	Fev 23 1961	02/23/61

Expressão CASE

CASE

Avalia uma lista de condições e retorna uma das várias expressões de resultado possíveis.

A expressão CASE tem dois formatos:

- A expressão CASE simples compara uma expressão com um conjunto de expressões simples para determinar o resultado.
 - A expressão CASE pesquisada avalia um conjunto de expressões booleanas para determinar o resultado.
- Os dois formatos dão suporte a um argumento ELSE opcional.

```
DECLARE @data DATETIME
SET @data = GETDATE() - 1

SELECT @data,
CASE
    WHEN @data = GETDATE() THEN 'Hoje'
    WHEN @data < GETDATE() THEN 'Ontem'
    WHEN @data > GETDATE() THEN 'Amanha'
    ELSE 'Não Foi Possível Definir a Data'
END AS DIA
```

	(Nenhum nome de coluna)	DIA
1	2021-12-13 14:00:55.303	Hoje

```
DECLARE @data DATETIME
SET @data = GETDATE() + 1
```

	(Nenhum nome de coluna)	DIA
1	2021-12-14 14:02:28.133	Amanha

Criando uma Nova Coluna com o CASE WHEN

```
SELECT      ProductNumber,
            ProductLine,
            CATEGORIA = CASE ProductLine -- Cria a nova coluna
                        WHEN 'R' THEN 'ROAD'
                        WHEN 'M' THEN 'MOUTAIN'
                        WHEN 'T' THEN 'TOURING'
                        WHEN 'S' THEN 'OTHER SALES ITEMS'
                        ELSE 'NOT FOR SALE'
            END,
            NAME
FROM        Production.Product
ORDER BY    ProductNumber
```

OU

```
SELECT      productnumber,
            productline,
            CASE
                WHEN productline = 'R' THEN 'ROAD'
                WHEN productline = 'M' THEN 'MOUNTAIN'
                WHEN productline = 'T' THEN 'TOURING'
                WHEN productline = 'S' THEN 'OTHER SALES ITEMS'
                ELSE 'NOTE FOR SALE'
            END CATEGORIA
FROM        Production.Product
ORDER BY    productnumber
```

	ProductNumber	ProductLine	CATEGORIA	NAME
1	AR-5381	NULL	NOT FOR SALE	Adjustable Race
2	BA-8327	NULL	NOT FOR SALE	Bearing Ball
3	BB-7421	NULL	NOT FOR SALE	LL Bottom Bracket
4	BB-8107	NULL	NOT FOR SALE	ML Bottom Bracket
5	BB-9108	NULL	NOT FOR SALE	HL Bottom Bracket
6	BC-M005	M	MOUTAIN	Mountain Bottle Cage
7	BC-R205	R	ROAD	Road Bottle Cage
8	BE-2349	NULL	NOT FOR SALE	BB Ball Bearing
9	BE-2908	NULL	NOT FOR SALE	Headset Ball Bearings
10	BK-M18B-40	M	MOUTAIN	Mountain-500 Black, 40


```

SELECT
    ProductNumber,
    name,
    ListPrice,
    CASE
        WHEN ListPrice = 0 THEN 'Não está a venda'
        WHEN ListPrice <= 50 THEN 'Abaixo de $ 50'
        WHEN ListPrice > 50 AND ListPrice <= 250 THEN 'Entre $51 e $250'
        WHEN ListPrice > 250 AND ListPrice <= 1000 THEN 'Entre $251 e
$1000'
        ELSE 'Acima de $1000'
    END PRICE_MARGE,
    'PRODUTOS' AS CATEGORIA
FROM Production.Product

```

	ProductNumber	name	ListPrice	PRICE_MARGE	CATEGORIA
1	HL-U509-R	Sport-100 Helmet, Red	34,99	Abaixo de \$ 50	PRODUTOS
2	HL-U509	Sport-100 Helmet, Black	34,99	Abaixo de \$ 50	PRODUTOS
3	SO-B909-M	Mountain Bike Socks, M	9,50	Abaixo de \$ 50	PRODUTOS
4	SO-B909-L	Mountain Bike Socks, L	9,50	Abaixo de \$ 50	PRODUTOS
5	HL-U509-B	Sport-100 Helmet, Blue	34,99	Abaixo de \$ 50	PRODUTOS
6	CA-1098	AWC Logo Cap	8,99	Abaixo de \$ 50	PRODUTOS

Tratamento de Erros

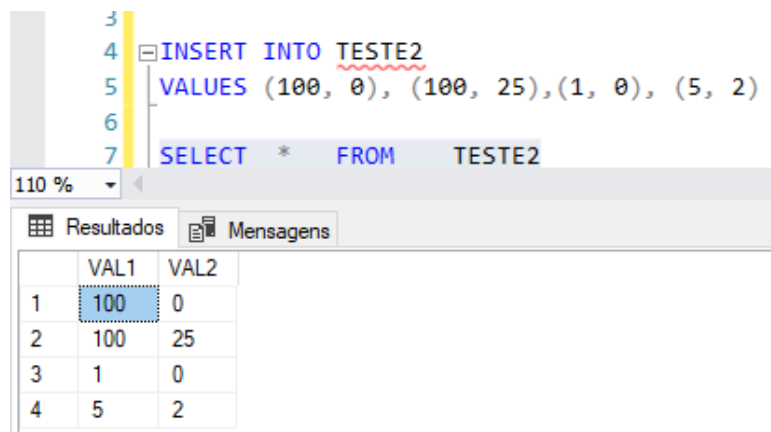
NULLIF

Retorna um valor nulo se as duas expressões especificadas forem iguais.

ISNULL

Retorna um valor especificado se a expressão for nulo.

Foi criado para o exemplo



The screenshot shows a SQL IDE with a query editor and a results pane. The query editor contains the following SQL code:

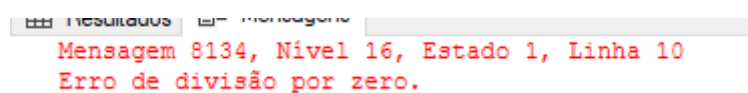
```
3  
4 INSERT INTO TESTE2  
5 VALUES (100, 0), (100, 25), (1, 0), (5, 2)  
6  
7 SELECT * FROM TESTE2
```

The results pane shows a table with two columns, VAL1 and VAL2, and four rows of data:

	VAL1	VAL2
1	100	0
2	100	25
3	1	0
4	5	2

Forçando para um erro

```
SELECT      VAL1,  
            VAL2,  
            VAL1/VAL2 AS EXPRESSAO_ERRO  
FROM        TESTE2
```



The screenshot shows an error message in a SQL IDE. The message is displayed in red text and reads:

```
Mensagem 8134, Nível 16, Estado 1, Linha 10  
Erro de divisão por zero.
```

Tratamento

NULLIF

Mal apresentado pelo professor do curso.

Sem exemplos.

ISNULL

```
SELECT DISTINCT a.id_aluno,
                a.nome,
                b.periodo as periodo
FROM alunos a
LEFT JOIN matricula b
ON a.id_aluno = b.id_aluno
```

Veja que existe um valor nulo.

	id_aluno	nome	periodo
1	1	Joao	Matutino
2	1	Joao	Noturno
3	1	Joao	Vespertino
4	2	Maria	Noturno
5	3	Pedro	Noturno
6	4	Tiago	NULL
7	5	Henrique	Matutino
8	5	Henrique	Noturno

Para isso, a sintaxe: "ISNULL (coluna, caso_seja_nulo)"

```
SELECT DISTINCT a.id_aluno,
                a.nome,
                ISNULL(b.periodo, 'Perd.Vazio') as periodo
FROM alunos a
LEFT JOIN matricula b
ON a.id_aluno = b.id_aluno
```

	id_aluno	nome	periodo
1	1	Joao	Matutino
2	1	Joao	Noturno
3	1	Joao	Vespertino
4	2	Maria	Noturno
5	3	Pedro	Noturno
6	4	Tiago	Perd.Vazio
7	5	Henrique	Matutino
8	5	Henrique	Noturno
9	5	Henrique	Vespertino

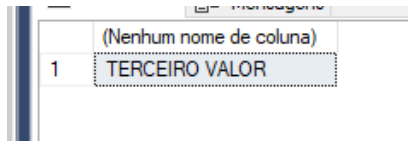
COALESCE

COALESCE

Avalia os argumentos na ordem e retorna o valor atual da primeira expressão que não é avaliada como NULL inicialmente.

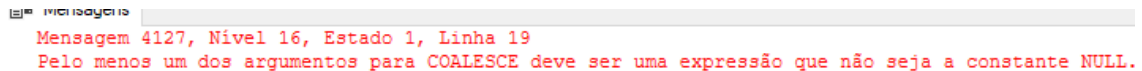
Se todos os argumentos forem NULL, COALESCE retornará NULL. Pelo menos um dos valores nulos precisa ser do tipo NULL.

```
-- Retorna o primeiro valor não nulo de uma constante  
SELECT COALESCE(NULL, NULL, 'TERCEIRO VALOR', 'QUARTO VALOR')
```



	(Nenhum nome de coluna)
1	TERCEIRO VALOR

```
SELECT COALESCE(NULL, NULL, NULL, NULL)
```



```
Mensagem 4127, Nível 16, Estado 1, Linha 19  
Pelo menos um dos argumentos para COALESCE deve ser uma expressão que não seja a constante NULL.
```

Exemplo prático

Criando uma tabela.

```
--drop table tab_salario;  
CREATE TABLE tab_salario  
(  
    matricula      int      identity,  
    salario_hora   decimal   NULL,  
    salario        decimal   NULL,  
    comissao       decimal   NULL,  
    vendas         int       NULL  
);
```

Populando a tabela com Inserts

```
INSERT tab_salario (salario_hora, salario, comissao, vendas)
VALUES
  (10.00, NULL, NULL, NULL),
  (20.00, NULL, NULL, NULL),
  (30.00, NULL, NULL, NULL),
  (40.00, NULL, NULL, NULL),
  (NULL, 10000.00, NULL, NULL),
  (NULL, 20000.00, NULL, NULL),
  (NULL, 30000.00, NULL, NULL),
  (NULL, 40000.00, NULL, NULL),
  (NULL, NULL, 15000, 3),
  (NULL, NULL, 25000, 2),
  (NULL, NULL, 20000, 6),
  (NULL, NULL, 14000, 4);
```

Isso significa que alguns funcionários recebem salário por hora, outros, apenas o salário e outros recebem apenas pagamento por comissão venda.

Como achar o salário de cada um?

```
SELECT matricula,
       salario_hora, salario, comissao, vendas
       -- CAST(COALESCE(salario_hora * 40 * 52, salario, comissao * vendas) AS money)
AS 'Total Salario'
FROM tab_salario
ORDER BY matricula;
```

	matricula	salario_hora	salario	comissao	vendas
1	1	10	NULL	NULL	NULL
2	2	20	NULL	NULL	NULL
3	3	30	NULL	NULL	NULL
4	4	40	NULL	NULL	NULL
5	5	NULL	10000	NULL	NULL
6	6	NULL	20000	NULL	NULL
7	7	NULL	30000	NULL	NULL
8	8	NULL	40000	NULL	NULL
9	9	NULL	NULL	15000	3
10	10	NULL	NULL	25000	2
11	11	NULL	NULL	20000	6
12	12	NULL	NULL	14000	4

Da forma acima, não é possível. Porém, adicionando o COALESCE, podemos criar uma coluna que fará a leitura dessas informações.

```
SELECT matricula,  
       salario_hora, salario, comissao, vendas,  
       CAST(COALESCE(salario_hora * 40 * 52, salario, comissao * vendas) AS  
money) AS 'Total Salario'  
FROM tab_salario  
ORDER BY matricula;
```

	matricula	salario_hora	salario	comissao	vendas	Total Salario
1	1	10	NULL	NULL	NULL	20800,00
2	2	20	NULL	NULL	NULL	41600,00
3	3	30	NULL	NULL	NULL	62400,00
4	4	40	NULL	NULL	NULL	83200,00
5	5	NULL	10000	NULL	NULL	10000,00
6	6	NULL	20000	NULL	NULL	20000,00
7	7	NULL	30000	NULL	NULL	30000,00
8	8	NULL	40000	NULL	NULL	40000,00
9	9	NULL	NULL	15000	3	45000,00
10	10	NULL	NULL	25000	2	50000,00
11	11	NULL	NULL	20000	6	120000,00
12	12	NULL	NULL	14000	4	56000,00

Views

Views servem para reutilização de Queries complexas que podem ser acionadas várias vezes, evitando retrabalho. Uma boa prática é ao criar a views, é adicionar um “v_nome_da_view”. Importando o AS após o nome.

Criando a View

```
use AdventureWorks2017;
CREATE VIEW v_data_contratacao _
AS _
    SELECT p.firstname, _
           p.lastname, _
           e.businessentityid, _
           e.hiredate _
    FROM   humanresources.employee e _
    JOIN   person.person AS p _
ON e.businessentityid = p.businessentityid;

--select na view

select * from v_data_contratacao
where year(hiredate)='2009';
```

Alterando a view

Adicione apenas o ALTER VIEW.

```
Alter VIEW v_data_contratacao _
AS _
    SELECT p.firstname primeiranome, _
           p.lastname ultimoNome,
           p.MiddleName nome_meio, --Campo adicionado_
           e.businessentityid id, _
           e.hiredate _admissao
    FROM   humanresources.employee e _
    JOIN   person.person AS p _
ON e.businessentityid = p.businessentityid;
```

É possível criar views de inserções, de unions, de joins.

Uma view é um procedimento para visualização e reutilização de algo.

Tabelas Temporárias Globais

As tabelas temporárias normais são criadas com um “#” simples e só podem ser acessadas pela conexão local e responsável que foi criada. Ela continuará ativa enquanto a conexão estiver ativa.

As tabelas temporárias GLOBAIS são criadas com dois “##” (duplo hashtag) e podem ser acessadas por diversas sessões. Porém, só continuará ativa durante a conexão.

Transact SQL

Permite a criação de scripts e procedimentos de armazenagem de dados/scripts (PROCEDURES).

EXTENSÃO SQL (T-SQL)

As extensões SQL permitem a criação de “scripts” poderosos e procedimentos armazenados (scripts armazenados no servidor e que podem ser reutilizados).

Há diversas restrições relacionadas à criação de um script, exemplo:

Instruções de definição de dados CREATE VIEW, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER e CREATE DEFAULT devem ser cada uma a única instrução em um script.

Variáveis locais

As variáveis são sempre identificadas em um batch através de um prefixo @.

A atribuição de valores é feita usando-se:

- ✓ a forma especial da instrução **SELECT**.
- ✓ A instrução **SET**.

Cada bloco começa com uma instrução BEGIN e termina com o END

BEGIN

Instrução 1

Instrução 2

...

END

Estrutura IF

```
IF 1 = 1
    BEGIN
        PRINT 'CORRETO, É TRUE'
    END
ELSE
    PRINT 'ERRADO, É FALSE'
```

Mensagens

CORRETO, É TRUE

```
IF 1 = 1 AND 2 <> 2
    BEGIN
        PRINT 'CORRETO, É TRUE'
    END
ELSE
    PRINT 'ERRADO, É FALSE'
```

Mensagens

ERRADO, É FALSE

```
DECLARE @id_aluno int;
SET @id_aluno='1';
IF (SELECT Count(*)
    FROM matricula _ WHERE id_aluno = @id_aluno ) = 0
    BEGIN
        PRINT 'Aluno não Matriculado'
    END
ELSE
    BEGIN
        PRINT 'Disciplina Matriculadas';
        SELECT b.nome_disc,a.periodo from matricula a
        inner join disciplina b
        on a.id_disciplina=b.id_disciplina
        and a.id_aluno=@id_aluno
    END ;
```

	nome_disc	periodo
1	Fisica	Noturno
2	Quimica	Vespertino
3	Matematica	Matutino

Estrutura WHILE

```
DECLARE @cont int
SET @cont=10
WHILE (select getdate()-@cont) <=getdate()
BEGIN
    Print getdate()-@cont
    SET @cont=@cont-1
    IF (getdate()-@cont)>=getdate()

        BREAK

ELSE
    CONTINUE
END;
```

Mensagens			
Dez	7	2021	1:29AM
Dez	8	2021	1:29AM
Dez	9	2021	1:29AM
Dez	10	2021	1:29AM
Dez	11	2021	1:29AM
Dez	12	2021	1:29AM
Dez	13	2021	1:29AM
Dez	14	2021	1:29AM
Dez	15	2021	1:29AM
Dez	16	2021	1:29AM

Tratamento de Erro com Try – Catch

Erros possíveis que são identificados. Serve para debugar erro.

Recuperando informações de erro

No escopo de um bloco CATCH, as seguintes funções de sistema podem ser usadas para obter informações sobre o erro que causou a execução do bloco CATCH.

- ERROR_NUMBER() retorna o número do erro.
- ERROR_SEVERITY() retorna a severidade.
- ERROR_STATE() retorna o número do estado do erro.
- ERROR_PROCEDURE() retorna o nome do procedimento armazenado ou do gatilho no qual ocorreu o erro.
- ERROR_LINE() retorna o número de linha dentro da rotina que causou o erro.
- ERROR_MESSAGE() retorna o texto completo da mensagem de erro. O texto inclui os valores fornecidos para qualquer parâmetro substituível, como cumprimentos, nomes de objeto ou horas.

A estrutura básica

BEGIN TRY

algo

END TRY

BEGIN CATCH

se algo der errado

END CATCH

```
BEGIN TRY
    SELECT      1/0 -- DIVISÃO POR ZERO APRESENTA ERRO
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER(),
        ERROR_SEVERITY(),
        ERROR_STATE(),
        ERROR_PROCEDURE(),
        ERROR_LINE(),
        ERROR_MESSAGE()
END CATCH
```

	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)	(Nenhum nome de coluna)
1	8134	16	1	NULL	2	Erro de divisão por zero.

Funções

Criação de funções próprias.

Funções Escalares

São definidas pelo usuário e são utilizadas o “return”.

O corpo da função possui um bloco BEGIN e END com o bloco do comando pertencente ao código.

Funções com Valores de Tabela

Não existe um corpo de função, o resultado é uma tabela proveniente de um select.

Funções em linha (In line)

Não requer o BEGIN e END para utilização. O retorno será uma tabela, mas com esquema no momento da criação.

Criação de Função

Alterando uma Função

Deletando uma Função

CURSORES

Cursores são áreas compostas por linhas e colunas em memória que servem para armazenar o resultado de uma consulta (Select) que retorna 0 ou nenhuma linha como resultado. Pode ser criado em qualquer momento do código.

A sintaxe da criação é:

DECLARE CURSOR

Para criar um cursor utilizamos a instrução DECLARE CURSOR.

SINTAXE:

```
DECLARE cursor_name CURSOR [ LOCAL | GLOBAL ]  
    [ FORWARD_ONLY | SCROLL ]  
    [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]  
    [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]  
    [ TYPE_WARNING ]  
    FOR select_statement  
    [ FOR UPDATE [ OF column_name [ ,...n ] ] ]  
[;]
```

LOCAL / GLOBAL. É usado para definir o escopo do cursor assim como funciona em tabelas temporárias (@local ou @@global).

FORWARD_ONLY / SCROLL. Indica a rolagem a ser definida para o cursor e aceita as palavras-chaves: FORWARD_ONLY e SCROLL.

STATIC, KEYSET, DYNAMIC e FAST_FORWARD. Usado para definir o tipo do cursor a ser criado: STATIC, KEYSET, DYNAMIC e FAST_FORWARD.

READ_ONLY, SCROLL_LOCKS e OPTIMISTIC. Indica o tipo de bloqueio, se as linhas poderão ser atualizadas pelo cursor e, se assim for, se outros usuários também poderão atualizá-los.

TYPE_WARNING. Este parâmetro instrui o SQL Server para enviar uma mensagem de aviso para o cliente se um cursor for convertido do tipo especificado em outro tipo.

FOR SQL_STATEMENT. Especifica às linhas a serem incluídas no conjunto do cursor.

FOR UPDATE.... Este parâmetro é opcional, por padrão os cursores são atualizáveis e não ser que o parâmetro de bloqueio seja READ_ONLY. Neste parâmetro podem-se especificar as linhas que permitem a atualização. Se forem omitidas todas as colunas na instrução serão atualizáveis.

Passo a Passo

Criando cursores

- 1° um DECLARE @MinhaVariavel, que irá receber os dados da consulta;
- 2° um DECLARE meu_cursor CURSOR LOCAL FOR, que é nosso cursor;
- 3° um OPEN meu_cursor, responsável por abrir nosso cursor;
- 4° um FETCH NEXT FROM meu_cursor e um INTO @MinhaVariavel, responsáveis por percorrer o cursor e setar o valor para a variável @MinhaVariavel;
- 5° um WHILE, responsável por realizar o loop em nossa consulta;
- 6° um PRINT, para simplesmente exibir o valor;
- 7° um novo FETCH NEXT FROM meu_cursor e um INTO @MinhaVariavel, que fará o mesmo processo anterior, passar para o próximo registro da consulta;
- 8° um CLOSE e um DEALLOCATE, responsáveis por fechar nosso cursor e remover sua referência.

PROCEDURES

A procedure é para tarefas repetitivas.

Selecione a sua Database.

Use a sintaxe:

```
CREATE PROCEDURE nome_procedure
```

```
AS
```

```
BEGIN
```

O que a procedure fará

```
END
```

```
USE CURSO
```

```
CREATE PROCEDURE PROC_OLA
```

```
AS
```

```
BEGIN
```

```
    SELECT 'OLÁ, MUNDO!'
```

```
END
```

Para executar uma procedure:

```
EXECUTE PROC_OLA
```

```
EXEC PROC_OLA
```

Resultados		Mensagens
	(Nenhum nome de coluna)	
1	OLÁ, MUNDO!	
	(Nenhum nome de coluna)	
1	OLÁ, MUNDO!	

Mais exemplos

```
CREATE PROCEDURE aluno_10
AS
BEGIN
    SELECT *
    FROM alunos
    WHERE id_aluno < 10
END
```

EXECUTE aluno_10

	id_aluno	nome
1	1	Joao
2	2	Maria
3	3	Pedro
4	4	Tiago
5	5	Henrique

TRIGGERS

Um bloco de comando que é executado automaticamente após um comando de INSERT, DELETE ou UPDATE ser executado.

Triggers

Um Trigger é bloco de comandos Transact-SQL que é automaticamente executado quando um comando **INSERT**, **DELETE** ou **UPDATE** for executado em uma tabela do banco de dados.

Os Triggers são usados para realizar tarefas relacionadas com validações, restrições de acesso, rotinas de segurança e consistência de dados; desta forma estes controles deixam de ser executados pela aplicação e passam a ser executados pelos Triggers em determinadas situações.

Sintaxe:

```
CREATE TRIGGER [NOME DO TRIGGER]
ON [NOME DA TABELA]
[FOR/AFTER/INSTEAD OF] [INSERT/UPDATE/DELETE]
AS
--CORPO DO TRIGGER
```

Exemplo de aplicação de uma trigger.

Uma tabela de funcionários.

```
CREATE TABLE FUNC
```

```
(
  MATRICULA          INT      IDENTITY(1,1) PRIMARY KEY
, NOME                VARCHAR(30)          NOT NULL
, SOBRENOME           VARCHAR(30)          NOT NULL
, ENDEREÇO            VARCHAR(30)          NOT NULL
, CIDADE              VARCHAR(30)          NOT NULL
, ESTADO              VARCHAR(30)          NOT NULL
, DATA_NASC          DATETIME
)
```

Populada:

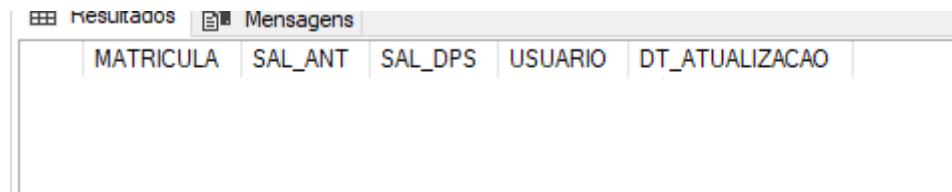
```
insert into func values ('Steve', 'Morse', 'Rua 13', 'JUNDIAI', 'SP', '1977-11-05')
insert into func values ('Joao', 'Pedro', 'Rua 14', 'SÃO PAULO', 'SP', '1980-27-10')
insert into func values ('Maria', 'Clara', 'Rua 15', 'RIBERA0 PRETO', 'SP', '1985-05-05')
insert into func values ('Pedro', 'Luiz', 'Rua 16', 'CAMPINAS', 'SP', '1990-12-09')
```

	MATRICULA	NOME	SOBRENOME	ENDEREÇO	CIDADE	ESTADO	DATA_NASC
1	1	Steve	Morse	Rua 13	JUNDIAI	SP	1977-05-11 00:00:00.000
2	2	Joao	Pedro	Rua 14	SÃO PAULO	SP	1980-10-27 00:00:00.000
3	3	Maria	Clara	Rua 15	RIBERA0 PRETO	SP	1985-05-05 00:00:00.000
4	4	Pedro	Luiz	Rua 16	CAMPINAS	SP	1990-09-12 00:00:00.000

Criada a tabela que recebe a TRIGGER (não populada)

```
CREATE TABLE AUDIT_SALARIO
```

```
(  
  MATRICULA          VARCHAR(30)          NOT NULL  
, SAL_ANT            DECIMAL(10, 2)        NOT NULL  
, SAL_DPS            DECIMAL(10, 2)        NOT NULL  
, USUARIO            VARCHAR(20)          NOT NULL  
, DT_ATUALIZACAO     DATETIME              NOT NULL  
)
```



The screenshot shows a database interface with two tabs: 'Resultados' and 'Mensagens'. The 'Resultados' tab is active, displaying the structure of the AUDIT_SALARIO table. The table has five columns: MATRICULA, SAL_ANT, SAL_DPS, USUARIO, and DT_ATUALIZACAO.

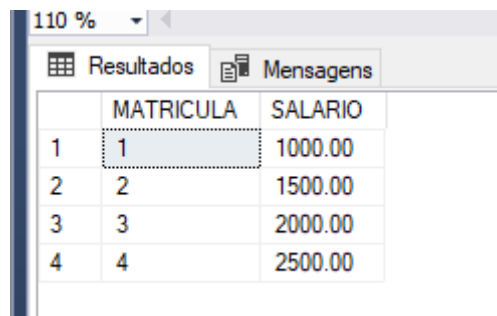
	MATRICULA	SAL_ANT	SAL_DPS	USUARIO	DT_ATUALIZACAO
--	-----------	---------	---------	---------	----------------

Crie a tabela SALARIO.

```
CREATE TABLE SALARIO
```

```
(  
  MATRICULA          INT                  NOT NULL  
, SALARIO            DECIMAL(10,2) NOT NULL  
)
```

```
insert into SALARIO values (1,1000)  
insert into SALARIO values (2,1500)  
insert into SALARIO values (3,2000)  
insert into SALARIO values (4,2500)
```



The screenshot shows a database interface with two tabs: 'Resultados' and 'Mensagens'. The 'Resultados' tab is active, displaying the data in the SALARIO table. The table has two columns: MATRICULA and SALARIO. The data is as follows:

	MATRICULA	SALARIO
1	1	1000.00
2	2	1500.00
3	3	2000.00
4	4	2500.00

Essa tabela alimentará a AUDIT_SALARIO através da TRIGGER.

Criação da Primeira Trigger

```
CREATE TRIGGER TG_aud_sal          -- CREATE TRIGGER    nome_da_trigger
ON    SALARIO                    -- ON                nome_da_tabela
AFTER UPDATE                     -- sempre que ocorrer algo, nesse caso, um UPDATE

AS
BEGIN

    DECLARE

        @sal_antigo  DECIMAL (10,2),
        @sal_novo    DECIMAL (10,2),
        @matricula   INT

        SELECT @matricula =(SELECT matricula FROM inserted)
        SELECT @sal_antigo =(SELECT SALARIO      FROM deleted)
        SELECT @sal_novo   =(SELECT SALARIO      FROM inserted)

    INSERT INTO          AUDIT_SALARIO
    VALUES
    (@matricula, ISNULL(@sal_antigo, 0), @sal_novo, SYSTEM_USER,
GETDATE())

END

SELECT * FROM FUNC
SELECT * FROM SALARIO
SELECT * FROM AUDIT_SALARIO
```

Não existem dados na AUDIT_SALARIO, até que a TRIGGER seja acionada

Resultados							
	MATRICULA	NOME	SOBRENOME	ENDERECO	CIDADE	ESTADO	DATA_NASC
1	1	Steve	Morse	Rua 13	JUNDIAI	SP	1977-05-11 00:00:00.000
2	2	Joao	Pedro	Rua 14	SÃO PAULO	SP	1980-10-27 00:00:00.000
3	3	Maria	Clara	Rua 15	RIBERA0 PRETO	SP	1985-05-05 00:00:00.000
4	4	Pedro	Luiz	Rua 16	CAMPINAS	SP	1990-09-12 00:00:00.000

	MATRICULA	SALARIO
1	1	1000.00
2	2	1500.00
3	3	2000.00
4	4	2500.00

MATRICULA	SAL_ANT	SAL_DPS	USUARIO	DT_ATUALIZACAO
-----------	---------	---------	---------	----------------

Testando a trigger, é necessário um UPDATE da TABELA de SALARIOS, para que então a tabelade AUDIT_SALARIO receba os valores

MERGE

Realiza operações de inserções, atualizações ou exclusões em uma tabela de DESTINO, a partir de uma base de ORIGEM.

MERGE Sintaxe:

INTO <target_table> – Define a tabela ou view que será a fonte de destino.

AS – Define um alias.

USING <table_source> – Define a tabela ou view que será fonte de origem, baseado na condição de merge da tabela destino com a tabela de origem.

ON <merge_search_condition> – Define as condições de “Join” da tabela destino com a tabela origem.

WHEN MATCHED THEN – Define a ação a ser realizada quando há linhas na fonte de destino origem que correspondem com a fonte de origem.

WHEN NOT MATCHED [BY TARGET] THEN – Define a ação a ser realizada quando não há linhas da fonte de origem que correspondem com a fonte de destino.

WHEN NOT MATCHED BY SOURCE THEN – Define a ação a ser realizada quando há linhas na fonte de destino, mas não há linhas correspondentes na fonte de origem.

Exemplificando a criação do MERGE.

Criando uma tabela 1 e populando

```
CREATE TABLE #tabela1
(
  NOME VARCHAR(10),
  CADASTRO DATETIME,
  ALTERACAO DATETIME,
  SITUACAO BIT
)
```

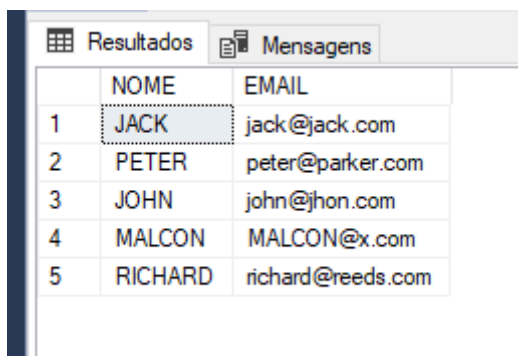
```
INSERT #tabela1
VALUES
('JACK', GETDATE(), NULL, 1),
('PETER', GETDATE(), NULL, 1),
('JOHN', GETDATE(), NULL, 1),
('MALCON', GETDATE(), NULL, 1),
('ARTHUR', GETDATE(), NULL, 1)
```

	NOME	CADASTRO	ALTERACAO	SITUACAO
1	JACK	2022-01-06 13:43:14.193	NULL	1
2	PETER	2022-01-06 13:43:14.193	NULL	1
3	JOHN	2022-01-06 13:43:14.193	NULL	1
4	MALCON	2022-01-06 13:43:14.193	NULL	1
5	ARTHUR	2022-01-06 13:43:14.193	NULL	1

Agora crie a segunda tabela e popule

```
CREATE TABLE #tabela2  
(  
    NOME VARCHAR(100),  
    EMAIL VARCHAR(100)  
)
```

```
INSERT #tabela2  
VALUES  
( 'JACK', 'jack@jack.com' ),  
( 'PETER', 'peter@parker.com' ),  
( 'JOHN', 'john@jhon.com' ),  
( 'MALCON', 'MALCON@x.com' ),  
( 'RICHARD', 'richard@reeds.com' )
```



The screenshot shows a SQL query results window with two tabs: 'Resultados' (Results) and 'Mensagens' (Messages). The 'Resultados' tab is active, displaying a table with two columns: 'NOME' (Name) and 'EMAIL' (Email). The table contains five rows of data, numbered 1 through 5. The first row is highlighted with a blue background.

	NOME	EMAIL
1	JACK	jack@jack.com
2	PETER	peter@parker.com
3	JOHN	john@jhon.com
4	MALCON	MALCON@x.com
5	RICHARD	richard@reeds.com

Observações

Veja que na tabela 1 existe o ARTHUR e na tabela 2 não. Enquanto na tabela 2 existe um RICHARD, que não existe na tabela 1.

Isso altera o comportamento do MERGE.

Iniciando o MERGE

```
MERGE #tabela1 AS Destino
USING #tabela2 AS Origem
ON Destino.NOME = Origem.NOME
```

Quando há registros no Destino, iguais a Origem

-- Há registros no destino, que são iguais a origem? Se sim...

```
WHEN MATCHED
THEN UPDATE SET SITUACAO = 0, ALTERACAO = GETDATE()
```

Quando não há registros no Destino, iguais a Origem

-- Não há registros no destino, que são iguais a Origem.

```
WHEN NOT MATCHED
THEN INSERT(NOME, CADASTRO, ALTERACAO, SITUACAO) VALUES (Origem.NOME, GETDATE(),
GETDATE(), 1)
```

Quando há registro no Destino, mas não há na Origem

-- Quando há registros no Destino, mas não há na Origem

```
WHEN NOT MATCHED BY SOURCE
THEN UPDATE SET SITUACAO = NULL, ALTERACAO = GETDATE()
```


Executando o MERGE

Para executar e valer o MERGE, selecione todo o bloco contido e execute normalmente.

```
MERGE #tabela1 AS Destino
USING #tabela2 AS Origem
ON Destino.NOME = Origem.NOME
-- Há registros no destino, que são iguais a origem? Se sim...
WHEN MATCHED
THEN UPDATE SET SITUACAO = 0, ALTERACAO = GETDATE()

-- Não há registros no destino, que são iguais a Origem.
WHEN NOT MATCHED
THEN INSERT(NOME, CADASTRO, ALTERACAO, SITUACAO) VALUES (Origem.NOME, GETDATE(),
GETDATE(), 1)

-- Quando há registros no Destino, mas não há na Origem
WHEN NOT MATCHED BY SOURCE
THEN UPDATE SET SITUACAO = NULL, ALTERACAO = GETDATE()

-- Mensagem
Mensagem 10713, Nível 15, Estado 1, Linha 70
Uma instrução MERGE deve ser terminada por um ponto-e-vírgula (;).

Horário de conclusão: 2022-01-06T14:01:56.9035686-03:00
```

Não esqueça do ponto-e-vírgula no final da instrução!

ANTES TABELA 1

	NOME	CADASTRO	ALTERACAO	SITUACAO
1	JACK	2022-01-06 13:43:14.193	NULL	1
2	PETER	2022-01-06 13:43:14.193	NULL	1
3	JOHN	2022-01-06 13:43:14.193	NULL	1
4	MALCON	2022-01-06 13:43:14.193	NULL	1
5	ARTHUR	2022-01-06 13:43:14.193	NULL	1

ANTES TABELA 2

Resultados		Mensagens
	NOME	EMAIL
1	JACK	jack@jack.com
2	PETER	peter@parker.com
3	JOHN	john@jhon.com
4	MALCON	MALCON@x.com
5	RICHARD	richard@reeds.com

APÓS O MERGE

	NOME	CADASTRO	ALTERACAO	SITUACAO
1	JACK	2022-01-06 13:43:14.193	2022-01-06 14:02:51.137	0
2	PETER	2022-01-06 13:43:14.193	2022-01-06 14:02:51.137	0
3	JOHN	2022-01-06 13:43:14.193	2022-01-06 14:02:51.137	0
4	MALCON	2022-01-06 13:43:14.193	2022-01-06 14:02:51.137	0
5	ARTHUR	2022-01-06 13:43:14.193	2022-01-06 14:02:51.137	NULL
6	RICHARD	2022-01-06 14:02:51.137	2022-01-06 14:02:51.137	1

	NOME	EMAIL
1	JACK	jack@jack.com
2	PETER	peter@parker.com
3	JOHN	john@jhon.com
4	MALCON	MALCON@x.com
5	RICHARD	richard@reeds.c...

Outro exemplo de MERGE

```
CREATE TABLE PRODUTOS
(
    cod_prod          INT PRIMARY KEY
    , descricao       VARCHAR(100)
    , preco           MONEY
)
```

```
INSERT PRODUTOS
VALUES
(1, 'Chá', 10.00),
(2, 'Café', 20.00),
(3, 'Leite', 30.00),
(4, 'Pão', 40.00)
```

```
CREATE TABLE prod_atual
(
    cod_prod          INT PRIMARY KEY
    , descricao       VARCHAR(100)
    , preco           MONEY
)
```

```
INSERT prod_atual
VALUES
(1, 'Chá', 10.00),
(2, 'Café', 25.00),
(3, 'Leite', 35.00),
(5, 'Peixe', 60.00)
```

Resultados		Mensagens	
	cod_prod	descricao	preco
1	1	Chá	10,00
2	2	Café	20,00
3	3	Leite	30,00
4	4	Pão	40,00

	cod_prod	descricao	preco
1	1	Chá	10,00
2	2	Café	25,00
3	3	Leite	35,00
4	5	Peixe	60,00

```

MERGE PRODUTOS AS destino
USING prod_atual AS origem
ON destino.cod_prod = origem.cod_prod

```

```

WHEN MATCHED

```

```

THEN UPDATE SET destino.preco = origem.preco

```

```

WHEN NOT MATCHED

```

```

THEN INSERT (cod_prod, descricao, preco) VALUES (origem.cod_prod,
origem.descricao, origem.preco);

```

Resultado final:

Resultados		Mensagens	
	cod_prod	descricao	preco
1	1	Chá	10,00
2	2	Café	25,00
3	3	Leite	35,00
4	4	Pão	40,00
5	5	Peixe	60,00

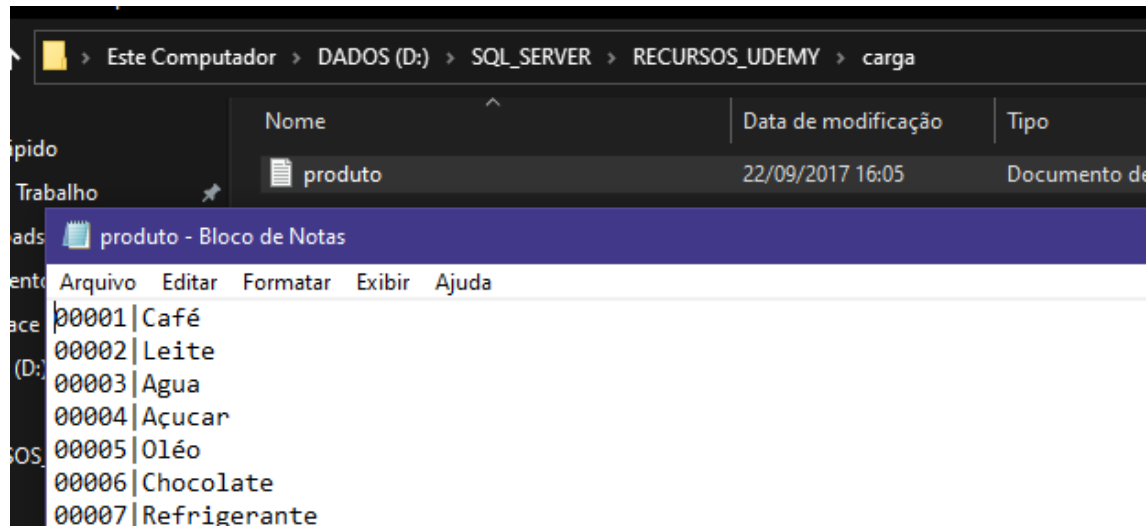
	cod_prod	descricao	preco
1	1	Chá	10,00
2	2	Café	25,00
3	3	Leite	35,00
4	5	Peixe	60,00

Outras Fontes de Dados e Dados em Massa

Bulk Insert

Serve para copiar e importar dados de um arquivo texto ou nativo do SQL (Flat File).

Um arquivo .txt, em um diretório da rede.

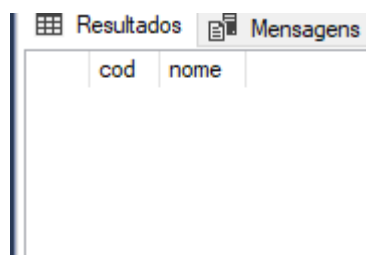


Com informações, com registros e seus dados separados por 'Pipe (|)'.

Esses registros serão armazenados em uma tabela que será criada.

```
CREATE TABLE produtos
(
    cod          NVARCHAR(10),
    nome         NVARCHAR(20)
)
```

```
SELECT *
FROM produtos
```



Porém, precisamos criar os parâmetros que compõem o BULK INSERT.

Parâmetros do BULK INSERT

```
--Bulk insert nome_da_tabela
--FROM local_do_arquivo
BULK INSERT      produtos
FROM             'D:\SQL_SERVER\RECURSOS_UDEMY\carga\produto.txt'
WITH
(
    codepage = 'ACP',
    DATAFILETYPE = 'widechar', DESUSO
    fieldterminator = '|',
    rowterminator = '\n',
    maxerrors = 0,
    fire_triggers,
    firstrow = 1,
    lastrow = 5 -- Nesse exemplo é até a linha 5
);
```

	cod	nome
1	00001	Café
2	00002	Leite
3	00003	Água
4	00004	Açúcar
5	00005	Óleo

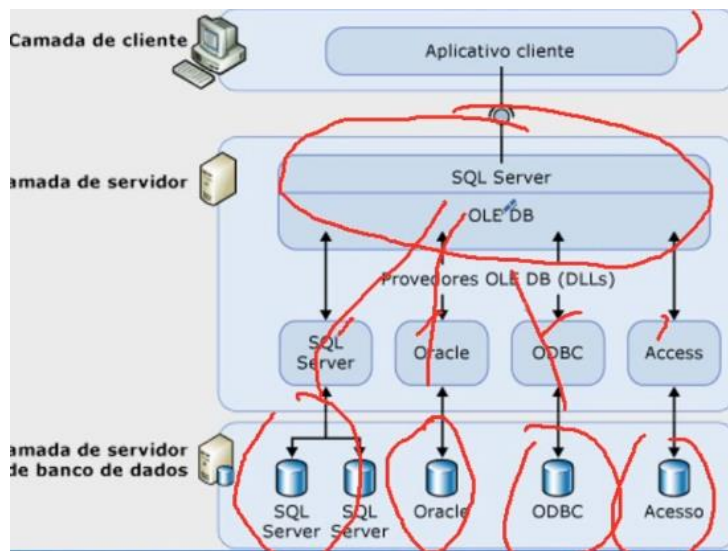
Para ler todos as linhas e registros, basta não colocar o parâmetro da 'lastrow' que fará a leitura de todos os registros.

Existe a OPENROWSET, que é similar ao BULK INSERT.

Linked Server

É uma funcionalidade para estabelecer conexão entre servidores, como uma ponte entre fontes OLE DB, sendo elas: Excel, Access, Oracle, MySQL, Sybase e entre outros.

Facilidade de executar consultas em servidores distintos.



Linked Server Disponível

Verifica se já existe um LINKED SERVER disponível.

```
SELECT * FROM sys.servers
```

A captura de tela mostra uma interface de consulta SQL com o seguinte resultado:

	server_id	name	product	provider	data_source	location	provider_string	catalog	connect_timeout	query_timeout	is_linked
1	0	DESKTOP-SMAHRM4	SQL Server	SQLNCLI	DESKTOP-SMAHRM4	NULL	NULL	NULL	0	0	0

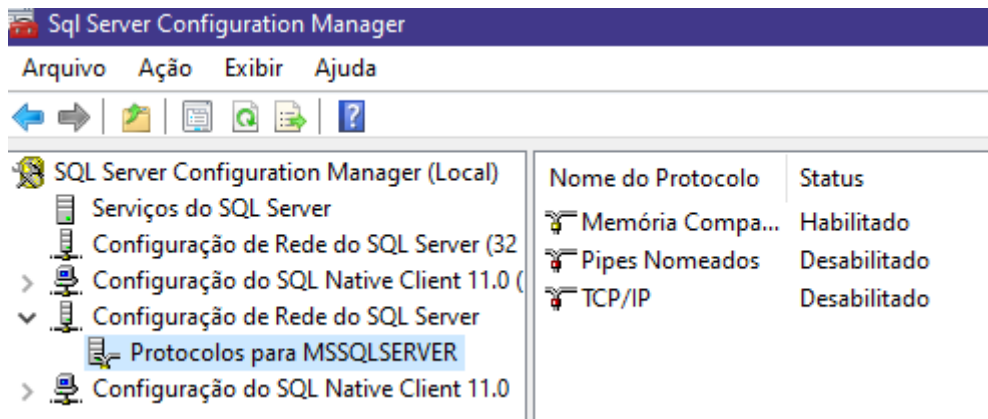
Apenas uma instância está disponível. Para esse exemplo, precisaríamos de duas.

Permissões Necessárias Obrigatórias

LINKED SERVER-PONTOS DE ATENÇÃO.

- ✓ Configuration Manager Pipes Nomeado e TCP/IP -> **Habilitados**
- ✓ Configurações Firewall -> Portas 1433, 1434 -> **Liberadas, Regras.**
- ✓ Conexão Remota na Instância -> **Permitir**
- ✓ Conexão entre as redes -> **Funcionando**

Digite: SQL Configuration Manager



Por exemplo, nesse caso acarretaria um erro. Precisa que todos os serviços de Protocolos estejam habilitados.

Projeto de Mini – ERP Multi Empresas

Para o início do projeto, devemos

Analisar os Requisitos

Identificar os requisitos e necessidades que o banco deverá atender;

Projetar Conceitualmente

Definir em uma visão Macro. Nessa etapa são utilizados os Modelos Entidade Relacionamento.

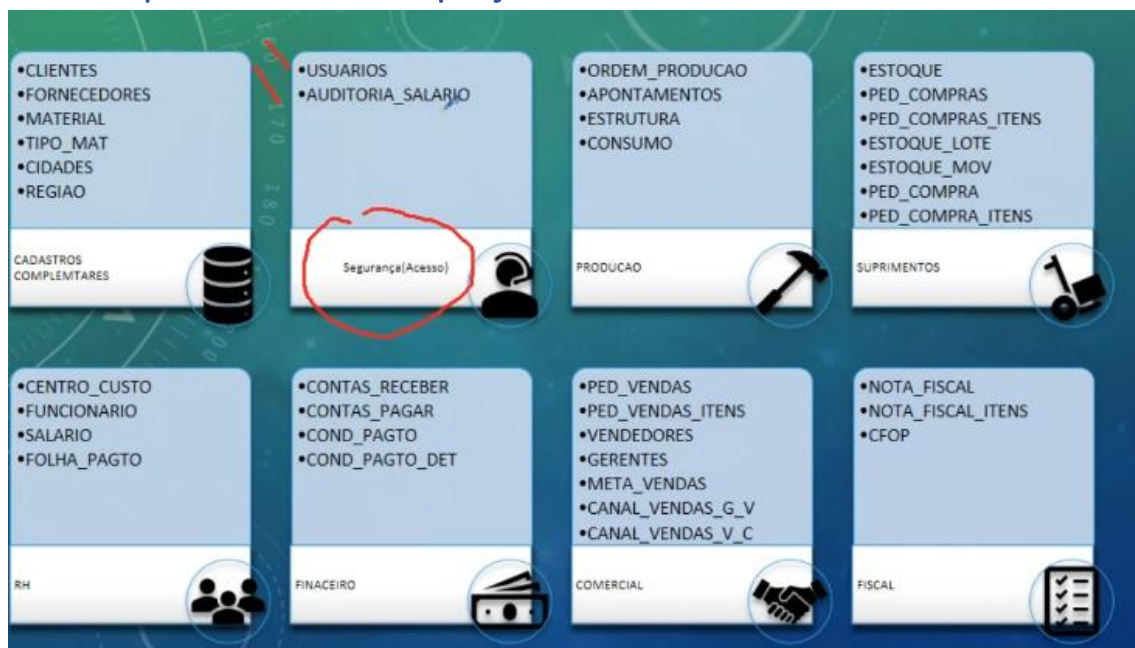
Projeto Lógico

Detalhamento de tabelas, relacionamentos, regras, tipos de dados da coluna(tamanho, tipo,...)

Projeto Físico

Implementação do sistema, forma como os dados serão armazenados, scripts para criação dos objetos, permissões de acessos.

Blocos que atenderam o projeto

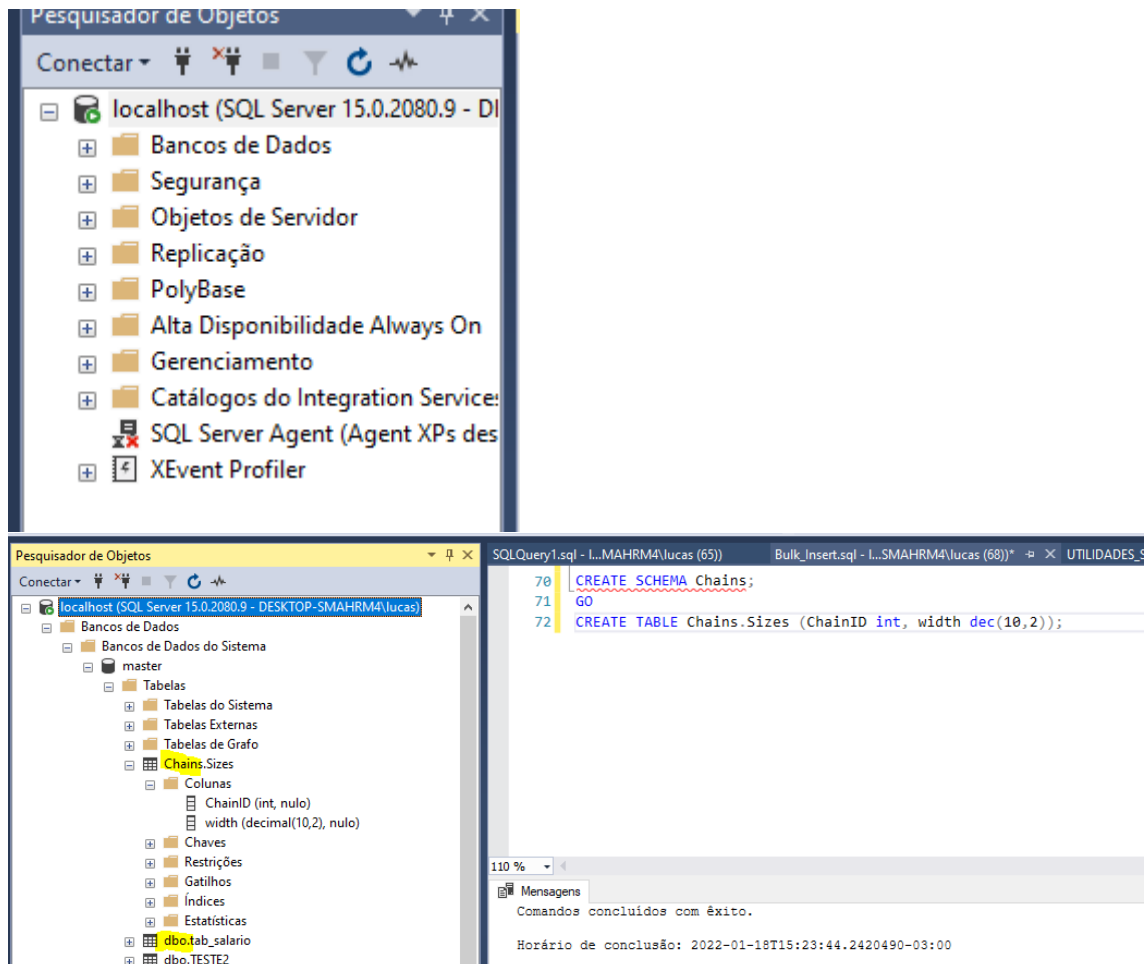


CTE – Common Table Expression ou Subfactory

É quando se cria uma tabela, quase que temporária, só que ela existirá apenas na instância da criação e desenvolvimento do código.

SCHEMA

```
CREATE SCHEMA Chains;  
GO  
CREATE TABLE Chains.Sizes (ChainID int, width dec(10,2));
```

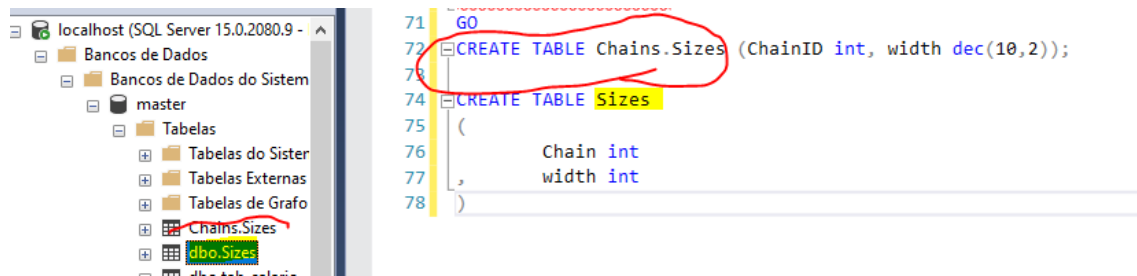


Para consultar os esquemas:

Vá até onde o schema foi criado, em qual Base de Dados.

A partir daí, você pode criar tabelas exclusivamente dentro dessas schemas.

Veja que se não foi criado informado o Schema, vai ser criado dentro(Nesse caso!!!) da MASTER.

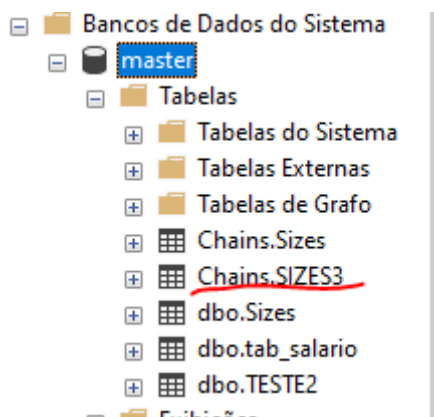


Transferência de Schemas

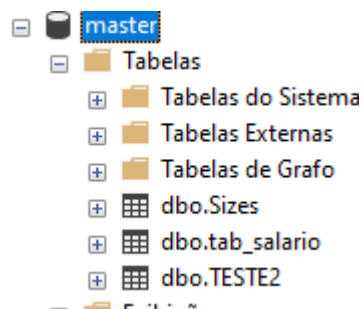
-- SINTAXE:

```
ALTER SCHEMA TargetSchema
    TRANSFER SourceSchema.TableName;
```

```
ALTER SCHEMA Chains
    TRANSFER dbo.SIZES3
```



Agora a tabela que era dbo.SIZES3, agora é parte do Schema do CHAINS.



CHAR, VARCHAR e NCHAR/NVARCHAR

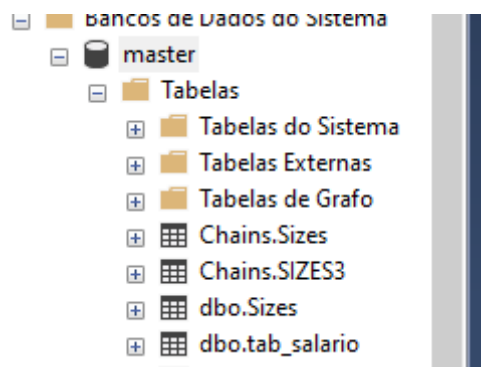
Pode-se utilizar:

O tipo **CHAR** deve ser usado quando **sabemos que todos os dados armazenados** em determinada coluna não são variáveis como, por exemplo, uma coluna que armazena a **sigla do estado** ou o **cep** que sempre terão o **mesmo tamanho sempre**.

Já o **VARCHAR** deve ser utilizado quando **não sabemos o que vamos armazenar**. Um exemplo pode ser o nome do cliente, endereço, o email que sempre **variam de tamanho**.

Dropar um Schema

Para dropar um schema, é preciso antes, dropar todas as tabelas que existam nele. Ou transferir as tabelas para um outro schema.



```
DROP TABLE Chains.Sizes  
DROP TABLE Chains.SIZES3  
DROP SCHEMA Chains
```

Criando um identificador único

Sintaxe para criação:

```
DECLARE @uniid uniqueidentifier = NEWID();
```

```
SELECT @uniid AS RESULT;
```

Resultados		Mensagens	
RESULT			
1	0466D566-C4C3-4E9B-AD26-7A694BD8584D		