Banco de Dados

Módulo Intermediário

TÓPICOS AVANÇADOS
FUNÇÕES PREDEFINIDAS

Data/Hora

- ***Funções** de **Data/Hora***
  - *Permite **armazenar valores** de **data** e **hora** em **vários formatos***
  - ***Embora** seja possível **armazenar valores** de **data** e **hora** em **praticamente qualquer formato**, o melhor é usar as **funções** do **SQL Server** para **retorná-los** em um **formato** que **atenda** as suas **necessidades***
  - *As **funções** que **manipulam data/hora** pode fazer:*
    - ***Retornar valores** de **data** e **hora** de **variadas precisões***
    - ***Retornar partes** de **valores** de **data** e **hora***

Data/Hora

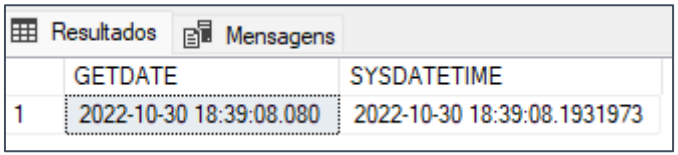
- ***Funções** de **Data/Hora***
  - *As **funções** que **manipulam data/hora** pode fazer:*
    - ***Extrair valores** de **data** e **hora** de **partes** de **datas** e **horas***
    - ***Obter diferenças** de **data** e **hora***
    - ***Modificar valores** de **data** e **hora***
    - ***Validar valores** de **data** e **hora***

Data/Hora

- ***Funções** de **Data/Hora***
  - ***Exemplo (01):***
    - ***Obtendo data** e **hora** do **sistema** com **GETDATE** e **SYSDATETIME**. **GETDATE** **retorna** um **tipo** de **dados** **DATETIME** e a **função** **SYSDATETIME** **retorna** um **datetime**(7)*

```
USE MyAdventureWorks;

SELECT GETDATE() AS GETDATE,
       SYSDATETIME() AS SYSDATETIME;
```



	GETDATE	SYSDATETIME
1	2022-10-30 18:39:08.080	2022-10-30 18:39:08.1931973

Data/Hora

- ***Funções** de **Data/Hora***
  - ***Exemplo (02):***
    - ***Retornando fragmentos** de uma **data/hora***

```
USE MyAdventureWorks;

SELECT
   DAY(GETDATE()) AS DAY,
   MONTH(GETDATE()) AS MONTH,
   YEAR(GETDATE()) AS YEAR,
   DATENAME(WEEKDAY, GETDATE()) AS DATENAMEWeekDay,
   DATEPART(M, GETDATE()) AS DATEPART,
   DATEPART(WEEKDAY, GETDATE()) AS DateParteWeekDay,
   DATENAME(MONTH, GETDATE()) AS DateNameMonth;
```

Data/Hora

- ***Funções** de **Data/Hora***
  - *Os **valores retornados** das **primeiras três funções** – **DAY**, **MONTH** e **YEAR** são **óbvios***
  - *As **duas últimas funções**, **DATENAME** e **DATEPART**, **oferecem** um **pouco mais** de **funcionalidade***
  - *Ao **contrário** das **três primeiras funções**, **DATENAME** e **DATEPART** **aceitam** um **parâmetro adicional** conhecido como **datepart***
  - *Esse **parâmetro** diz a **função** qual **parte** da **data deve ser retornada***

Data/Hora

- ***Funções** de **Data/Hora***

datepart	Abreviações
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms

Tabela 1 – Argumentos *datepart* válidos

Data/Hora

- ***Funções** de **Data/Hora***

datepart	Abreviações
microsecond	mcs
nanosecond	ns
TZoffset	tz
ISO_WEEK	Isowk, isoww

Tabela 1 – Argumentos *datepart* válidos

Data/Hora

- ***Funções** de **Data/Hora***
  - *Normalmente, cada **argumento** **datepart** **retorna** um **valor inteiro***
  - ***Podemos observar** que **valores** de **string** são **incluídos** no **resultado** do **exemplo anterior**, ao **utilizarmos** **month** e **weekday** como **argumento** **datepart** para a **função** **DATENAME***
  - ***Essa** é a **principal diferença entre** as **duas funções** (**DATEPART** **retorna** o **tipo** de **dados** com um **valor inteiro** / **DATENAME** **retorna valores** de **string/nvarchar**)*

Data/Hora

- ***Funções** de **Data/Hora***
  - ***Exemplo (03):***
    - ***Utilizando** a **função** DATEFROMPARTS*

```
USE MyAdventureWorks;

SELECT
    DATEFROMPARTS(1972, 5, 26) AS DATEFROMPARTS,
    DATETIME2FROMPARTS(1972, 5, 26, 7, 14, 16, 10, 3) AS DATETIME2FROMPARTS,
    DATETIMEFROMPARTS(1972, 5, 26, 7, 14, 16, 10) AS DATETIMEFROMPARTS,
    DATETIMEOFFSETFROMPARTS(1972, 5, 26, 7, 14, 16, 10, 12, 0, 3) AS
                                                    DATETIMEOFFSETFROMPARTS,
    SMALLDATETIMEFROMPARTS(1972, 5, 26, 7, 14) AS SMALLDATETIMEFROMPARTS,
    TIMEFROMPARTS(7, 14, 16, 10, 3) AS TIMEFROMPARTS
```

Resultados | Mensagens

	DATEFROMPARTS	DATETIME2FROMPARTS	DATETIMEFROMPARTS	DATETIMEOFFSETFROMPARTS	SMALLDATETIMEFROMPARTS	TIMEFROMPARTS
1	1972-05-26	1972-05-26 07:14:16.010	1972-05-26 07:14:16.010	1972-05-26 07:14:16.010 +12:00	1972-05-26 07:14:00	07:14:16.010

Data/Hora

- ***Funções** de **Data/Hora***
  - ***Além** de **oferecer** as **funções** já mencionadas, o **T-SQL permite efetuar cálculos** com **valores** de **data** e **validar** esses **valores***
  - ***Por exemplo**, é possível **calcular** o **número** de **dias entre duas datas** ou **somar um mês** ou **um ano** a **uma data***
  - *O **SQL Server** introduziu a **função EOMONTH**, a qual **determina** a **última data** de um **mês específico***

Data/Hora

- ***Funções** de **Data/Hora***
  - ***Exemplo (04):***
    - ***Obtendo** a **diferença**, **modificando** e **validando valores** de **data***

```
USE MyAdventureWorks;

SELECT
   DATEDIFF(dd, GETDATE(), '5/26/2025') AS DaysUnitlMyBirthday,
   DATEADD(y, 1, GETDATE()) AS DateAdd,
   EOMONTH(GETDATE()) AS EOMonth, -- New function
   ISDATE(GETDATE()) AS IsValidDate,
   ISDATE('13/1/2122') AS InvalidDate;
```

Resultados | Mensagens

	DaysUnitlMyBirthday	DateAdd	EOMonth	IsValidDate	InvalidDate
1	939	2022-10-31 20:08:49.883	2022-10-31	1	0

Conversão

- ***Funções** de **Conversão***
  - *As **funções** de **conversão** são **divididas** em **duas categorias:***
    - ***CAST***
    - ***CONVERT***
  - *A **principal finalidade** dos **dois tipos** é **mudar um valor** de **um tipo** de **dados** para **outro***
  - ***CONVERT** é **diferente** de **CAST**, pois permite **formatar** a **saída** de **uma conversão***

Conversão

- ***Funções** de **Conversão***
  - *O **SQL Server introduziu quatro novas funções** de **conversão***
    - ***PARSE***
    - ***TRY_PARSE***
    - ***TRY_CAST***
    - ***TRY_CONVERT***

Conversão

- ***Funções** de **Conversão***
  - ***Exemplo (05): - Parte 1***
    - *Utilizando a **função CAST** para **conversão** de **dados***

```
USE MyAdventureWorks;

SELECT TOP(10)
   SalesOrderNumber,
   totalDue,
   CAST(TotalDue AS DECIMAL(10,2)) AS TotalDueCast,
   OrderDate,
   CAST(OrderDate AS DATE) AS OrderDateCast
FROM Sales.SalesOrderHeader;
```

Conversão

- ***Funções** de **Conversão***
  - ***Exemplo (05): - Parte 2***
    - *Utilizando a **função CAST** para **conversão** de **dados***

	SalesOrderNumber	totalDue	TotalDueCast	OrderDate	OrderDateCast
1	SO43659	23153,2339	23153.23	2005-07-01 00:00:00.000	2005-07-01
2	SO43660	1457,3288	1457.33	2005-07-01 00:00:00.000	2005-07-01
3	SO43661	36865,8012	36865.80	2005-07-01 00:00:00.000	2005-07-01
4	SO43662	32474,9324	32474.93	2005-07-01 00:00:00.000	2005-07-01
5	SO43663	472,3108	472.31	2005-07-01 00:00:00.000	2005-07-01
6	SO43664	27510,4109	27510.41	2005-07-01 00:00:00.000	2005-07-01
7	SO43665	16158,6961	16158.70	2005-07-01 00:00:00.000	2005-07-01
8	SO43666	5694,8564	5694.86	2005-07-01 00:00:00.000	2005-07-01
9	SO43667	6876,3649	6876.36	2005-07-01 00:00:00.000	2005-07-01
10	SO43668	40487,7233	40487.72	2005-07-01 00:00:00.000	2005-07-01

Conversão

- ***Funções** de **Conversão***
  - ***Exemplo (06): - Parte 1***
    - ***Embora** as **funções CONVERT** e **CAST** executem a **mesma atividade primária**, a **função CONVERT** é **um pouco mais flexível** do que **CAST**, pois podemos **formatar** a **saída** de seu **conjunto** de **resultados** utilizando o **argumento style***
    - ***É possível aplicar estilos** nos **tipos** de **dados date**, **time**, **real**, **float**, **money**, **xml** e **binary***

```
SELECT
    CONVERT(VARCHAR(20), GETDATE()) AS [Default],
    CONVERT(VARCHAR(20), GETDATE(), 100) AS DefaultWithStyle,
    CONVERT(VARCHAR(10), GETDATE(), 103) AS BritishFrenchStyle,
    CONVERT(VARCHAR(8), GETDATE(), 105) AS ItalianStyle,
    CONVERT(VARCHAR(8), GETDATE(), 112) AS ISOStyle,
    CONVERT(VARCHAR(15), CAST('111111.22' AS MONEY), 1) AS MoneyWithCommas
```

Default	DefaultWithStyle	BritishFrenchStyle	ItalianStyle	ISOStyle	MoneyWithCommas
Oct 30 2022 8:56PM	Oct 30 2022 8:56PM	30/10/2022	30-10-20	20221030	111,111.22

Conversão

- ***Funções** de **Conversão***
  - ***Exemplo (06): - Parte 2***
    - ***Note** que a **função CONVERT** recebe **três argumentos***
    - *O **primeiro** é o **tipo** de **dados** de **destino** ou o **tipo** para o qual **desejamos converter determinado valor***
    - *O **segundo** é o **valor** que será **convertido** e o **último argumento** é o **estilo***
    - *Este **último argumento** é **opcional** e, se **não** for **fornecido**, o **SQL Server** usará **valores padrão***
    - *As **primeiras 5 colunas** representam **conversões** de **data** para **diferentes países***
    - *A **última coluna ilustra** o **uso** da **função CONVERT** para **adicionar vírgulas** a um **valor** com **tipo** de **dado money***

Conversão

- ***Funções** de **Conversão***
  - *O **SQL Server introduziu quatro novas funções** de **conversão PARSE** e **TRY_PARSE** são **totalmente novas** no **SQL Server**, enquanto **TRY_CONVERT** e **TRY_CAST** são **extensões** da **funções CONVERT** e **CAST** já **existentes***
  - *A **função PARSE** só deve ser **usada** ao se **converter** de **strings** para **tipos** de **dados data/hora** e **número***

Conversão

- ***Funções** de **Conversão***
  - ***As outras, prefixadas** com **TRY_**, **adicionam** funcionalidade às **funções CONVERT, CAST** e **PARSE básicas***
  - ***Converter** um **valor** usando uma das **funções mencionadas anteriormente** pode fazer a **instrução inteira falhar***
  - *Ao usar qualquer uma das **versões TRY** da **função**, será **retornado** um **valor NULL***

Conversão

- ***Funções** de **Conversão***
  - ***Exemplo (07):***
    - ***Convertendo datas** com as **novas funções***

```
SELECT
   TRY_CAST('José da Silva' AS INT) TryCast,
   TRY_CONVERT(DATETIME, '13/2/2999', 112) AS TryConvert,
   PARSE('Saturday, 26 May 2012' AS DATETIME USING 'en-US') AS Parse,
   TRY_PARSE('José da Silva BirthDay' AS DATETIME USING 'en-US') AS TryParse
```

Resultados | Mensagens

	TryCast	TryConvert	Parse	TryParse
1	NULL	NULL	2012-05-26 00:00:00.000	NULL

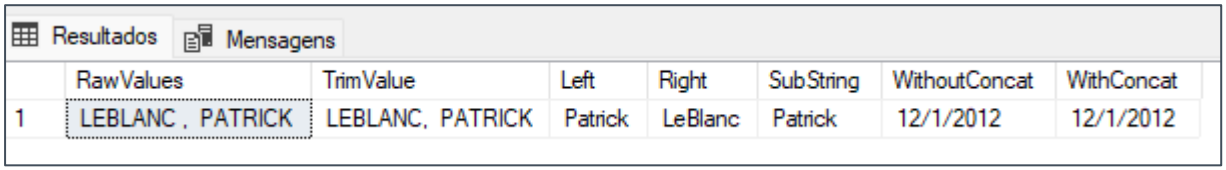
Conversão

- ***Funções** de **String***
  - *O **SQL Server** contém **25 funções** de **string predefinidas**, incluindo **duas novas**:*
    - ***CONCAT***
    - ***FORMAT***
  - ***Cada função** executa **alguma operação** sobre **um valor** de **string** ou **numérico** **fornecido***
  - *Algo a notar é que, podemos **combinar** ou **concatenar** uma **string** e **um valor número**, primeiramente o **valor numérico deverá** ser **convertido** em **uma string***

Conversão

- ***Funções** de **String***
  - ***Exemplo (08):***
    - *Utilizando **diversas funções** de **string***

```
SELECT
   'LEBLANC ' + ', ' + ' PATRICK' RawValues,
   RTRIM('LEBLANC  ') + ',  ' + LTRIM(' PATRICK') TrimValue,
   LEFT('PatrickDTomorr', 7) [Left],
   RIGHT('DTomorrLeBlanc', 7) [Right],
   SUBSTRING('DTomorrPatrick', 8, LEN('DTomorrPatrick')) [SubString],
   '12/' + CAST(1 AS VARCHAR) + '/2012' WithoutConcat,
   CONCAT('12/', 1, '/2012') WithConcat
```


Resultados Mensagens

	RawValues	TrimValue	Left	Right	SubString	WithoutConcat	WithConcat
1	LEBLANC , PATRICK	LEBLANC, PATRICK	Patrick	LeBlanc	Patrick	12/1/2012	12/1/2012

Conversão

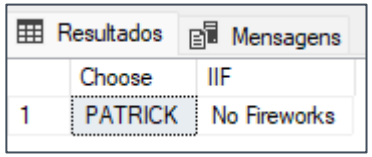
- ***Funções Lógicas***
  - *O **SQL Server** contém **duas novas funções lógicas** que permitem mais **seleção** de **dados** com **pouco código***
    - ***CHOOSE***
    - ***IIF***
  - ***CHOOSE** **retorna um valor** de **uma lista** com **base** em um **índice especificado***
  - ***IIF** **retorna um valor** com **base** na **avaliação** de **uma expressão booleana** como **verdadeira** ou **falsa***

Conversão

- ***Funções Lógicas***
  - ***Exemplo (09):***
    - *Utilizando as **funções lógicas** no exemplo*

```
DECLARE @choosevar INT = 3

SELECT
   CHOOSE(@choosevar, 'ONE', 'TWO', 'PATRICK', 'THREE') [Choose],
   IIF(DATENAME(MONTH, GETDATE()) = 'July',
               'The 4th is this month', 'No Fireworks') AS [IIF]
```



AGREGAÇÃO, VISUALIZAÇÃO E AGRUPAMENTO

T-SQL

- ***Tópicos Avançados***
  - *Normalmente, quando **dados** são **armazenados** em um **BD relacional**, **cada linha representa** um **único valor***
  - ***Por exemplo:***
    - *Em um **BD** de **vendas**, podemos **gravar uma linha** para **cada venda realizada** por **um vendedor***
    - *Em um **BD bancário**, podemos **gravar uma linha** para **cada transação** ocorrida na **conta** de **um cliente***
  - ***Independentemente** dos **dados armazenados**, em algum momento eles **precisarão** ser **agregados** de **alguma maneira***

T-SQL

- ***Tópicos Avançados***
  - ***No exemplo*** *do* ***BD*** *de* ***vendas****, talvez desejamos* ***visualizar todas*** *as* ***vendas*** *de* ***cada vendedor****, ao* ***passo*** *que no* ***exemplo*** *do* ***BD bancário****, talvez* ***desejamos visualizar*** *o* ***saldo*** *da* ***conta*** *de* ***cada cliente***
  - *Por meio da* ***sintaxe T-SQL****, podemos* ***agregar*** *os* ***dados*** *com* ***funções escalares*** *de* ***agregação predefinidas*** *e* ***agrupá-los*** *usando a* ***cláusula GROUP BY*** *para* ***efetuar*** *essas* ***operações***

T-SQL

- ***Agregações***
  - *A **agregação mais comum** é a **adição**, que **soma todos** os **valores** de **determinado conjunto** de **dados***
  - *A **adição** de **dados** é **suportada** por meio da **função SUM** do **T-SQL***
  - ***Sintaxe:***

```
SUM( [ ALL | DISTINCT] expressão)
```

T-SQL

- ***Agregações***
  - ***Sintaxe:***

```
SUM( [ ALL | DISTINCT] expressão)
```

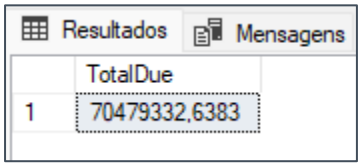
  - *A **palavra-chave ALL aplica** a **agregação** a **cada valor** no **resultado**, enquanto **DISTINCT agrega apenas** os **valores únicos***
  - *A **palavra-chave ALL** é **usada** por **padrão**, portanto, **não** é **necessário especificá-la** como **parte** da **query***
  - *A **expressão fornecida deve ser** um **tipo** de **dados numérico**, que pode ser uma **constante**, **variável** , **coluna** ou **função***

T-SQL

- ***Agregações***
  - ***Exemplo (01):***
    - ***Apresentando** o **total devido** para **todos** os **pedidos** de **compra realizados***

```
USE MyAdventureWorks;

SELECT
   SUM(poh.TotalDue) AS TotalDue
FROM Purchasing.PurchaseOrderHeader poh;
```



T-SQL

- ***Agregações***
  - ***Exemplo (02): - Parte 1***
    - ***Adicionando três novas agregações, obter a média do total devido e contabilizar o número de funcionários de duas maneiras diferentes***

```
USE MyAdventureWorks;

SELECT
   SUM(poh.TotalDue) AS [TotalDue],
   AVG(poh.TotalDue) AS [Average TotalDue],
   COUNT(poh.EmployeeID) [NumberOfEmployees],
   COUNT(DISTINCT(poh.EmployeeID)) [DistinctNumberOfEmployees]
FROM Purchasing.PurchaseOrderHeader poh;
```

Resultados | Mensagens

	TotalDue	Average TotalDue	NumberOfEmployees	DistinctNumberOfEmployees
1	70479332,6383	17567,1317	4012	12

T-SQL

- ***Agregações***
  - ***Exemplo (02): - Parte 2***
    - *Average Total Due utiliza a* ***função AVG*** *para* ***calcular*** *a* ***média*** *no* ***conjunto*** *de* ***resultados inteiro****. Ela* ***soma*** *o* ***total devido****, conta o* ***número*** *de* ***linhas****,* ***divide*** *os* ***dois valores*** *e* ***retorna*** *a* ***média***
    - *Number Of Employees utiliza a* ***função COUNT*** *para* ***cada funcionário*** *no resultado, incluindo* ***valores duplicados*** *da* ***coluna*** *fornecida,* ***EmployeeID***
    - *Distinct Number Of Employees utiliza a* ***função COUNT****, mas* ***inclui*** *a* ***palavra-chave DISTINCT*** *para* ***garantir*** *que os* ***registros duplicados*** *sejam* ***ignorados***

T-SQL

- ***Agregações** com **Agrupamentos***
  - ***Agregar** os **dados holisticamente** dentro de um **resultado** provavelmente **não** é **algo** que **se vê** com **frequência***
  - *Normalmente, os **dados são** **decompostos** ou **agregados** em **categorias** ou **segmentos**, como **ano**, **região**, **país** ou **vendedor***
  - *Com a **cláusula** **GROUP BY** é possível **resumir** o **conjunto** de **resultados selecionado** por **uma** ou **mais colunas** ou **expressões***
  - ***Sintaxe:***

```
SELECT
FROM
GROUP BY <coluna1>, <coluna2>, …
```

T-SQL

- ***Agregações** com **Agrupamentos***
  - ***Sintaxe:***

```
SELECT
FROM
GROUP BY <coluna1>, <coluna2>, …
```

  - ***Qualquer coluna não utilizada** por uma **função** de **agregação listada** na **instrução SELECT** pode ser **incluída** na **cláusula GROUP BY***
  - *Todavia, a **coluna** pode estar **incluída** na **cláusula GROUP BY**, mas **não** na **instrução SELECT***

T-SQL

- ***Agregações** com **Agrupamentos***
  - ***Exemplo (03): - Parte 1***
    - ***Realizando agregações** com a **cláusula GROUP BY***

```
USE MyAdventureWorks;

SELECT
   sm.Name AS ShippingMethod,
   SUM(poh.TotalDue) AS [TotalDue],
   AVG(poh.TotalDue) AS [Average TotalDue],
   COUNT(poh.EmployeeID) [NumberOfEmployees],
   COUNT(DISTINCT(poh.EmployeeID)) [DistinctNumberOfEmployees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
      ON(poh.ShipMethodID = sm.ShipMethodID)
GROUP BY sm.Name;
```

- ***Agregações** com **Agrupamentos***
  - ***Exemplo (03): - Parte 2***

ShippingMethod	TotalDue	Average TotalDue	NumberOfEmployees	DistinctNumberOfEmployees
XRQ - TRUCK GROUND	3330909,2897	5655,194	589	12
ZY - EXPRESS	14874601,7677	22709,3156	655	12
OVERSEAS - DELUXE	8002938,997	50018,3687	160	12
OVERNIGHT J-FAST	11965191,1871	11027,8259	1085	12
CARGO TRANSPORT 5	32305691,3968	21211,8787	1523	12

- *Com a **inclusão** da **cláusula GROUP BY** na **coluna Name** da **tabela ShippingMethod**, a query foi capaz de **fornecer agregações** para **cada método** de **despacho individual** utilizado (é possível agrupar por mais de uma coluna ou expressão)*

T-SQL

- ***Agregações** com **Agrupamentos***
  - ***Exemplo (04): - Parte 1***
    - ***Incluindo** uma **expressão** que **extraia** o **ano** da **coluna** OrderDate na **tabela** PurchaseOrderHeader*

```
USE MyAdventureWorks;

SELECT
   sm.Name AS ShippingMethod,
   YEAR(poh.OrderDate) AS OrderYear,
   SUM(poh.TotalDue) AS [TotalDue],
   AVG(poh.TotalDue) AS [Average TotalDue],
   COUNT(poh.EmployeeID) [NumberOfEmployees],
   COUNT(DISTINCT(poh.EmployeeID)) [DistinctNumberOfEmployees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
      ON(poh.ShipMethodID = sm.ShipMethodID)
GROUP BY sm.Name, YEAR(poh.OrderDate);
```

- ***Agregações** com **Agrupamentos***
  - ***Exemplo (04): - Parte 2***
    - ***Incluindo** uma **expressão** que **extraia** o **ano** da **coluna** OrderDate na **tabela** PurchaseOrderHeader*

	ShippingMethod	OrderYear	TotalDue	Average TotalDue	NumberOfEmployees	DistinctNumberOfEmployees
1	XRQ - TRUCK GROUND	2006	238226,7076	5672,0644	42	12
2	XRQ - TRUCK GROUND	2007	931577,2578	6128,7977	152	12
3	XRQ - TRUCK GROUND	2008	2161105,3243	5471,1527	395	12
4	ZY - EXPRESS	2005	9776,2665	9776,2665	1	1
5	ZY - EXPRESS	2006	1128598,8326	24534,7572	46	12
6	ZY - EXPRESS	2007	3657789,9952	21902,9341	167	12
7	ZY - EXPRESS	2008	10078436,6734	22853,5978	441	12
8	OVERSEAS - DELUXE	2005	81233,7049	27077,9016	3	3
9	OVERSEAS - DELUXE	2006	326145,231	36238,359	9	5
10	OVERSEAS - DELUXE	2007	1333639,9307	38103,998	35	10
11	OVERSEAS - DELUXE	2008	6261920,1304	55415,2223	113	12
12	OVERNIGHT J-FAST	2005	22539,0165	22539,0165	1	1
13	OVERNIGHT J-FAST	2006	750384,0766	10875,1315	69	12
14	OVERNIGHT J-FAST	2007	3295912,2337	11210,5858	294	12
15	OVERNIGHT J-FAST	2008	7896355,8603	10951,9498	721	12
16	CARGO TRANSPORT 5	2005	1255,8943	418,6314	3	3
17	CARGO TRANSPORT 5	2006	2226522,1526	21004,9259	106	12
18	CARGO TRANSPORT 5	2007	8073385,8419	20861,4621	387	12
19	CARGO TRANSPORT 5	2008	22004527,508	21426,0248	1027	12

T-SQL

- ***Visualizando Dados***
  - ***Visualizar dados*** *em* ***janelas*** *usando* ***T-SQL*** *é uma maneira de fornecer* ***uma nova perspectiva*** *deles*
  - *Essa* ***nova perspectiva*** *ou* ***nova janela*** *de* ***dados*** *é* ***criada*** *pela* ***cláusula OVER()***
  - ***Depois*** *que* ***uma nova janela*** *é* ***criada***, *é* ***gerado*** *um* ***valor*** *para* ***cada linha*** *no* ***conjunto*** *de* ***resultados***
  - ***Esses valores*** *são* ***obtidos*** *com as* ***funções*** *de* ***agregação*** *e/ou* ***funções*** *de* ***classificação*** *do* ***T-SQL***

T-SQL

- ***Visualizando Dados***
  - ***Existem 4 funções** de **classificação** básicas:*
    - ***ROW_NUMBER**: **gera inteiros incrementados automaticamente**, de acordo com a **classificação** na **cláusula OVER***
    - ***RANK**: **gera** um **valor baseado** na **classificação visualizada** em **janelas**, como se as **linhas estivessem competindo**. Se **houver empates**, as **duas linhas recebem** o **mesmo valor***
    - ***DENSE_RANK**: **gera** um **valor muito parecido** com a **função RANK**. Entretanto, se houver **empate, somente um valor** (igual) é **usado***
    - ***NTILE**: **organiza** as **linhas** em **algum número** de **grupos, chamados tiles***

T-SQL

- ***Visualizando Dados***
  - ***Exemplo (05): - Parte 1***
    - ***Visualizando dados*** *em* ***janelas*** *utilizando* ***ROW_NUMBER***, ***RANK*** *e* ***DENSE_RANK***

```
USE MyAdventureWorks;

GO
WITH ProductQty AS
(
   SELECT TOP(10)
          p.ProductID,
          SUM(OrderQty) AS OrderQty
   FROM Sales.SalesOrderDetail AS sod
   INNER JOIN Production.Product AS p
      ON(sod.ProductID = p.ProductID)
   GROUP BY p.ProductID
) --continua...
```

T-SQL

- ***Visualizando Dados***
  - ***Exemplo (05): - Parte 2***
    - ***Visualizando dados*** *em* ***janelas*** *utilizando* ***ROW_NUMBER, RANK*** *e* ***DENSE_RANK***

```
SELECT
   p.Name AS ProductionName,
   pq.OrderQty,
   ROW_NUMBER() OVER(ORDER BY pq.OrderQty DESC) RowNumber,
   RANK() OVER(ORDER BY pq.OrderQty DESC) [Rank],
   DENSE_RANK() OVER(ORDER BY pq.OrderQty DESC) [DenseRank]
FROM ProductQty AS pq
INNER JOIN Production.Product AS p
   ON(pq.ProductID = p.ProductID);
```

- ***Visualizando Dados***
  - ***Exemplo (05): - Parte 3***
    - ***Visualizando dados*** *em* ***janelas*** *utilizando* ***ROW_NUMBER, RANK*** *e* ***DENSE_RANK***

	ProductionName	OrderQty	RowNumber	Rank	DenseRank
1	Mountain-200 Silver, 38	2394	1	1	1
2	Men's Bib-Shorts, M	1616	2	2	2
3	LL Mountain Frame - Black, 44	625	3	3	3
4	Road-450 Red, 44	346	4	4	4
5	Touring-2000 Blue, 50	322	5	5	5
6	All-Purpose Bike Stand	249	6	6	6
7	LL Fork	190	7	7	7
8	ML Road Frame - Red, 52	90	8	8	8
9	Mountain Bike Socks, L	90	9	8	8
10	LL Touring Frame - Yellow, 58	36	10	10	9

T-SQL

- ***Visualizando Dados***
  - ***Exemplo (05): - Parte 4***
    - ***ROW_NUMBER gera*** *um* ***número*** *para* ***cada linha*** *em* ***sequência,*** *com* ***base*** *em* ***ORDER BY***
    - *Neste caso, o* ***produto*** *com* ***OrderQty mais alto aparece*** *em* ***primeiro lugar*** *e* ***todas*** *a* ***linhas subsequentes recebem*** *um* ***valor baseado*** *na* ***quantidade***
    - ***RANK*** *e* ***DENSE_RANK*** *o* ***comportamento*** *é* ***semelhante,*** *a* ***menos*** *que* ***encontrem empates***

- ***Visualizando Dados***
  - ***Exemplo (05): - Parte 5***
    - *Se **existirem empates**, **todas** as **linhas receberão** o **mesmo valor**. No entanto, quando **RANK** é **usada**, o **número gerado** para as **linhas** que **estão após** o **empate pula um valor** (dependendo do número de linhas incluídas no empate)*
    - *No **caso** de **DENSE_RANK**, diante de um **empate não** é **pulado** um **valor***

T-SQL

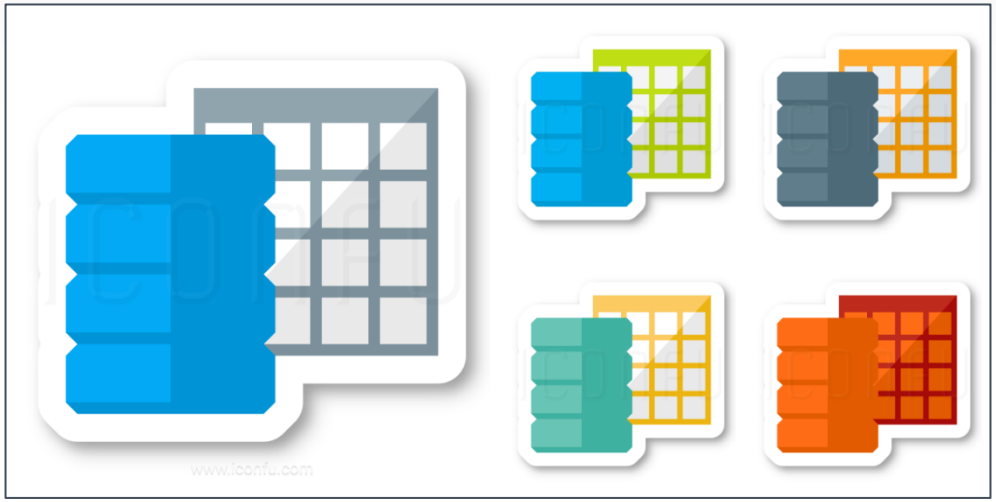
- ***Novos Recursos** de **Visualização***
  - ***Diversos argumentos** para **visualização** em **janelas** que podem ser **usados** com a **cláusula OVER***

Argumentos	Descrição
ROWS/RANGE	Limita as linhas com base na linha atual
CURRENT ROW	Declara o ponto inicial ou final como a linha atual
BETWEEN	É acoplado a ROWS ou RANGE para especificar os pontos iniciais e finais
UNBOUND PRECEDING	Declara onde a janela iniciará
UNBOUND FOLLOWING	Declara onde a janela terminará

Tabela 2 – Novos recursos de visualização em janela

T-SQL

- ***Cláusula HAVING***
  - *Se **comporta** de **modo semelhante** a uma **instrução SELECT**, **usufruindo** da **agregação***
  - ***Podemos utilizá-la** com uma **instrução SELECT** e, normalmente, ela é usada com uma **cláusula GROUP BY***



T-SQL

- ***Cláusula HAVING***
  - ***Exemplo (06): - Parte 1***
    - ***Limitando** as **linhas agregadas** com a **cláusula HAVING***

```
USE MyAdventureWorks;

SELECT
   sm.Name AS ShippingMethod,
   YEAR(poh.OrderDate) AS OrderYear,
   SUM(poh.TotalDue) AS [TotalDue],
   AVG(poh.TotalDue) AS [Average TotalDue],
   COUNT(poh.EmployeeID) [NumberOfEmployees],
   COUNT(DISTINCT(poh.EmployeeID)) [DistinctNumberOfEmployees]
FROM Purchasing.PurchaseOrderHeader poh
INNER JOIN Purchasing.ShipMethod sm
      ON(poh.ShipMethodID = sm.ShipMethodID)
GROUP BY sm.Name, YEAR(poh.OrderDate)
HAVING SUM(poh.TotalDue) > 5000000;
```

- ***Cláusula HAVING***
  - ***Exemplo (06): - Parte 2***
    - ***Limitando** as **linhas agregadas** com a **cláusula HAVING***

Resultados | Mensagens

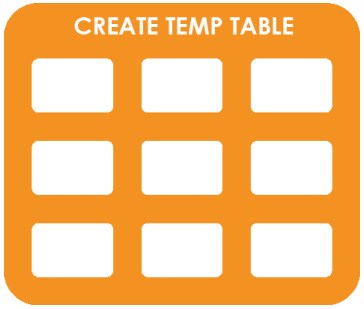
	ShippingMethod	OrderYear	TotalDue	Average TotalDue	NumberOfEmployees	DistinctNumberOfEmployees
1	ZY - EXPRESS	2008	10078436,6734	22853,5978	441	12
2	OVERSEAS - DELUXE	2008	6261920,1304	55415,2223	113	12
3	OVERNIGHT J-FAST	2008	7896355,8603	10951,9498	721	12
4	CARGO TRANSPORT 5	2007	8073385,8419	20861,4621	387	12
5	CARGO TRANSPORT 5	2008	22004527,508	21426,0248	1027	12

T-SQL

- ***Objetos Temporários***
  - *Ao **trabalhar** com **T-SQL**, muitas vezes **precisaremos armazenar um conjunto** de **dados temporariamente** para **uso posterior***
  - ***Considerando** os **exemplos anteriores**, se **desejarmos agregar** por **vendedor**, apresentando **nome**, **nome** do **meio**, **sobrenome** e **endereço**, a **query poderia** se **tornar muito dispendiosa**, **devido** ao **número** de **colunas** na **cláusula GROUP BY** e ao **número** de **JOIN**s **envolvidas***

T-SQL

- ***Objetos Temporários***
  - ***Portanto, em vez** de **unir cada tabela** e **agrupar** em **todas** as **colunas retornadas**, podemos **utilizar** um **objeto temporário** para **armazenar** os **dados agregados** e **unir** as **tabelas adicionais**, retornando as **colunas necessárias***
  - *O **SQL Server** tem **três objetos temporários:***
    - ***Expressões** de **tabela comuns***
    - ***Variáveis** de **tabela***
    - ***Tabelas temporárias***



T-SQL

- ***Expressões** de **Tabela Comuns***
  - *Uma **expressão** de **tabela comum** (**CTE** – common table expression) é um **conjunto** de **resultado temporário definido durante** a **execução** de uma **instrução SELECT**, **INSERT**, **UPDATE**, **DELETE** ou **CREATE VIEW***
  - *A **CTE** apenas está **disponível durante** a **execução** da **query** e **não** é **armazenada** como **outros objetos** no **BD***
  - *AS **CTE**s são **normalmente utilizadas** para **substituir views**, **agrupar dados**, realizar **recursão** e **criar várias referências** para **uma única tabela***

T-SQL

- ***Expressões** de **Tabela Comuns***
  - ***Sintaxe:***

```
WITH <nome_da_expressão> [(nome_da_coluna [,…n])]
AS
(definição_da_query_CTE)
```

  - ***nome_da_empressão** é **como** a **CTE** será **referenciada** na **query**, e é **obrigatório***
  - *A **listagem** de **colunas** é **opcional**, mas **recomendada***
  - ***Se** os **nomes** de **coluna** na **definição** da **query não forem exclusivos**, um **erro** será **reportado** na **execução** da **query***
  - ***Podemos utilizar** a **listagem** de **colunas** para **corrigir** o **problema** ou **corrigi-lo** na **definição** da **query** com **alias***

T-SQL

- ***Expressões** de **Tabela Comuns***
  - ***Imediatamente após** a **definição** da **CTE**, devemos **executar** uma **query** que a **referencie***
  - *Pode ser uma **instrução INSERT**, **UPDATE** ou **DELETE***

T-SQL

- ***Expressões** de **Tabela Comuns***
  - ***Exemplo (07): - Parte 1***
    - ***Criando** e **utilizando** uma **expressão** de **tabela comum***

```
USE MyAdventureWorks;

WITH EmployeePOs (EmployeeID, [Total Due])
AS
(
  SELECT poh.EmployeeID,
         CONVERT(VARCHAR(20), SUM(poh.TotalDue), 1)
  FROM Purchasing.PurchaseOrderHeader poh
  GROUP BY poh.EmployeeID
)
SELECT *
FROM EmployeePOs;
```

- ***Expressões** de **Tabela Comuns***
  - ***Exemplo (07): - Parte 2***
    - *A **agregação** é **definida** na **definição** da **CTE** e agora pode ser usada na **instrução SELECT** que vem **imediatamente** após a **CTE***

Resultados | Mensagens

	EmployeeID	Total Due
1	261	7,239,495.37
2	252	2,978,027.37
3	258	5,556,272.23
4	255	6,305,115.83
5	259	5,186,032.12
6	250	2,501,613.04
7	256	6,552,648.57
8	253	7,423,411.20
9	254	6,578,521.33
10	251	7,426,610.64
11	257	6,942,815.77
12	260	5,788,769.16

T-SQL

- ***Expressões** de **Tabela Comuns***
  - ***Exemplo (08): - Parte 1***
    - ***Adicionando** um **JOIN** à **tabela Person** para **acrescentar** os **nomes** e **sobrenomes** dos **funcionários***

```
USE MyAdventureWorks;

WITH EmployeePOs (EmployeeID, [Total Due])
AS
(
  SELECT poh.EmployeeID,
         CONVERT(VARCHAR(20), SUM(poh.TotalDue), 1)
  FROM Purchasing.PurchaseOrderHeader poh
  GROUP BY poh.EmployeeID
) --continua...
```

T-SQL

- ***Expressões** de **Tabela Comuns***
  - ***Exemplo (08): - Parte 2***
    - ***Adicionando** um **JOIN** à **tabela Person** para **acrescentar** os **nomes** e **sobrenomes** dos **funcionários***

```
SELECT ep.EmployeeID,
       p.FirstName,
       p.LastName,
       ep.[Total Due]
FROM EmployeePOs ep
INNER JOIN Person.Person p
  ON (ep.EmployeeID = p.BusinessEntityID);
```

- ***Expressões** de **Tabela Comuns***
  - ***Exemplo (08): - Parte 3***
    - ***Adicionando** um **JOIN** à **tabela Person** para **acrescentar** os **nomes** e **sobrenomes** dos **funcionários***

	EmployeeID	FirstName	LastName	Total Due
1	261	Reinout	Hillmann	7,239,495.37
2	252	Arvind	Rao	2,978,027.37
3	258	Erin	Hagens	5,556,272.23
4	255	Gordon	Hee	6,305,115.83
5	259	Ben	Miller	5,186,032.12
6	250	Sheela	Word	2,501,613.04
7	256	Frank	Pellow	6,552,648.57
8	253	Linda	Meisner	7,423,411.20
9	254	Fukiko	Ogisu	6,578,521.33
10	251	Mikael	Sandberg	7,426,610.64
11	257	Eric	Kurjan	6,942,815.77
12	260	Annette	Hill	5,788,769.16

T-SQL

- ***Variáveis** de **Tabela***
  - *As **variáveis** de **tabela** têm **comportamento semelhante** às **variáveis locais***
  - ***Normalmente**, elas são **utilizadas** para **armazenar pequenos volumes** de **dados** (menos de 500 linhas) e só **estão disponíveis dentro** do **escopo** do **lote**, **função** ou **stored procedure** no qual são **declaradas***
  - ***Sintaxe:***

```
DECLARE @variavel_local [AS] tabela
(
   [(definição_da_coluna) [,…n]]
)
```

T-SQL

- ***Variáveis** de **Tabela***
  - ***Sintaxe:***

```
DECLARE @variavel_local [AS] tabela
(
   [(definição_da_coluna) [,…n]]
)
```

  - ***Podemos substituir** variável_local pelo **nome** que **desejarmos**, mas deve **prefixá-lo** com uma **arroba** (@)*
  - *Na sequência, podemos **definir cada coluna** da **tabela***
  - ***Cada coluna** será **definida** da mesma **maneira** como se **define colunas** ao **criar** uma **tabela***

T-SQL

- ***Variáveis** de **Tabela***
  - ***Exemplo (09):***
    - ***Declarando** e **utilizando variáveis** de **tabela***

```
DECLARE @EmployeePOs AS TABLE
(
  EmployeeID    INT,
  TotalDue      MONEY
)

INSERT INTO @EmployeePOs
SELECT
  poh.EmployeeID,
  CONVERT(VARCHAR(20), SUM(poh.TotalDue), 1)
FROM Purchasing.PurchaseOrderHeader poh
GROUP BY poh.EmployeeID;
```

T-SQL

- ***Variáveis** de **Tabela***
  - ***Exemplo (10): - Parte 1***
    - ***Adicionando** uma **instrução SELECT** que **referencie** a **variável** de **tabela**, unindo com à **tabela** **Person***

```
DECLARE @EmployeePOs AS TABLE
(
  EmployeeID    INT,
  TotalDue      MONEY
)

INSERT INTO @EmployeePOs
SELECT
  poh.EmployeeID,
  CONVERT(VARCHAR(20), SUM(poh.TotalDue), 1)
FROM Purchasing.PurchaseOrderHeader poh
GROUP BY poh.EmployeeID;
--continua...
```

T-SQL

- ***Variáveis** de **Tabela***
  - ***Exemplo (10): - Parte 2***
    - ***Adicionando** uma **instrução SELECT** que **referencie** a **variável** de **tabela**, unindo com à **tabela** **Person***

```
SELECT ep.EmployeeID,
       p.FirstName,
       p.LastName,
       ep.[TotalDue]
FROM @EmployeePOs ep
INNER JOIN Person.Person p
  ON (ep.EmployeeID = p.BusinessEntityID);
```

Resultados | Mensagens

	EmployeeID	FirstName	LastName	TotalDue
1	261	Reinout	Hillmann	7239495,37
2	252	Arvind	Rao	2978027,37
3	258	Erin	Hagens	5556272,23
4	255	Gordon	Hee	6305115,83
5	259	Ben	Miller	5186032,12
6	250	Sheela	Word	2501613,04
7	256	Frank	Pellow	6552648,57
8	253	Linda	Meisner	7423411,20
9	254	Fukiko	Ogisu	6578521,33
10	251	Mikael	Sandberg	7426610,64
11	257	Eric	Kurjan	6942815,77
12	260	Annette	Hill	5788769,16

T-SQL

- ***Tabelas Temporárias***
  - *É possível* ***criar tabelas temporárias locais*** *e* ***globais***
  - *As* ***tabelas temporárias locais*** *estão* ***disponíveis*** *dentro do* ***escopo*** *da* ***sessão atual*** *e* ***são removidas*** *ao* ***final*** *de* ***uma sessão***
  - *Elas* ***devem ser prefixadas*** *com o* ***símbolo #***
  - *As* ***tabelas temporárias globais*** *estão* ***disponíveis*** *para* ***todas*** *as* ***sessões*** *e* ***são removidas*** *quando a* ***sessão*** *que as* ***criou*** *e* ***todas*** *as* ***sessões*** *que* ***fazem referências*** *a elas* ***são fechadas***
  - *Elas* ***devem*** *ser* ***prefixadas*** *com* ***dois símbolos ##***

T-SQL

- ***Tabelas Temporárias***
  - *A **sintaxe** para **criar uma** ou **outra** é **exatamente** a **mesma** da **criação** de uma **tabela tradicional**, mas devemos **incluir** o **sinal** **#** ou **##***
  - *Ao **contrário** do que **acontece** com os **outros dois objetos temporários**, é **alocado espaço** para **tabelas temporárias***
  - *As **tabelas temporárias** normalmente **não** são **gravadas** no **disco**, mas, em alguns casos, podem **utilizar recursos***

T-SQL

- ***Tabelas Temporárias***
  - ***Exemplo (11): - Parte 1***
    - ***Criando e utilizando tabelas temporárias***

```
USE MyAdventureWorks;

CREATE TABLE #EmployeePOs(
EmployeeID      INT,
TotalDue        MONEY
);

INSERT INTO #EmployeePOs
SELECT poh.EmployeeID,
       CONVERT(VARCHAR(20), SUM(poh.TotalDue), 1)
FROM Purchasing.PurchaseOrderHeader poh
GROUP BY poh.EmployeeID;
```

- ***Tabelas Temporárias***
  - ***Exemplo (11): - Parte 2***
    - ***Criando e utilizando tabelas temporárias***

```
SELECT
  ep.EmployeeID,
  p.FirstName,
  p.LastName,
  ep.[TotalDue]
FROM #EmployeePOs ep
INNER JOIN Person.Person p
  ON (ep.EmployeeID = p.BusinessEntityID);
```

	EmployeeID	FirstName	LastName	TotalDue
1	261	Reinout	Hillmann	7239495,37
2	252	Arvind	Rao	2978027,37
3	258	Erin	Hagens	5556272,23
4	255	Gordon	Hee	6305115,83
5	259	Ben	Miller	5186032,12
6	250	Sheela	Word	2501613,04
7	256	Frank	Pellow	6552648,57
8	253	Linda	Meisner	7423411,20
9	254	Fukiko	Ogisu	6578521,33
10	251	Mikael	Sandberg	7426610,64
11	257	Eric	Kurjan	6942815,77
12	260	Annette	Hill	5788769,16

T-SQL

- ***Tratamento** de **Erros***
  - ***Assim** como **toda linguagem** de **programação**, **T-SQL** fornece **métodos elegantes** para **tratar** (ou **manipular**) **erros** e **exceções** **durante** a **execução***
  - ***T-SQL** usa **TRY**...**CATCH**, **semelhante** ao **C#***
  - *Ao **escrever** em **T-SQL**, posicionamos o **código** no **bloco TRY** e, se **ocorre** um **erro**, o **controle** é **enviado** para o **bloco CATCH***
  - *O **código T-SQL** que **tratará** os **erros** deve ser **incluído** **dentro** do **bloco CATCH***

T-SQL

- ***Tratamento** de **Erros***
  - ***Sintaxe:***

```
BEGIN TRY
   { instrução_sql | bloco_de_instruções }
END TRY
BEGIN CATCH
   [{ instrução_sql | bloco_de instruções }]
END CATCH
```

  - *instrução_sql é uma **única instrução T-SQL** e bloco_de_instruções é **qualquer conjunto** ou **lote** de **instruções T-SQL***
  - ***Isso** se **aplica** a **blocos TRY** e **CATCH***
  - *Os **blocos TRY** e **CATCH** **devem** ser **construídos juntos***

T-SQL

- ***Tratamento** de **Erros***
  - *A **Microsoft introduziu** a **instrução THROW**, que **lança uma exceção** e **transfere** a **execução** para um **bloco CATCH***

```
THROW [{ número_do_erro | @variável_local },
   {mensagem | @variável_local },
   {estado | @variável_local },
] [ ; ]
```

  - *número_do_erro **deve estar entre** 50.000 e 2.147.483.647 e **pode** ser uma **constante** ou **uma variável**, mas é **opcional** ao se **implementar tratamento** de **erro** com **T-SQL***
  - *mensagem **descreve** o **erro** e **pode ser** uma **string** ou **uma variável***

T-SQL

- ***Tratamento** de **Erros***
  - *A **Microsoft introduziu** a **instrução THROW**, que **lança uma exceção** e **transfere** a **execução** para um **bloco CATCH***

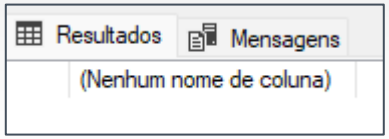
```
THROW [{ número_do_erro | @variável_local },
   {mensagem | @variável_local },
   {estado | @variável_local },
] [ ; ]
```

  - *estado deve **estar entre** 0 e 255 e pode ser **uma constante** ou **uma variável***
  - *A **instrução** que **precede THROW** deve ser **terminada** com um **ponto** e **vírgula** (;)*

- ***Tratamento** de **Erros***
  - ***Exemplo (12):***
    - ***Implementando tratamento** de **erro** utilizando **construções T-SQL***

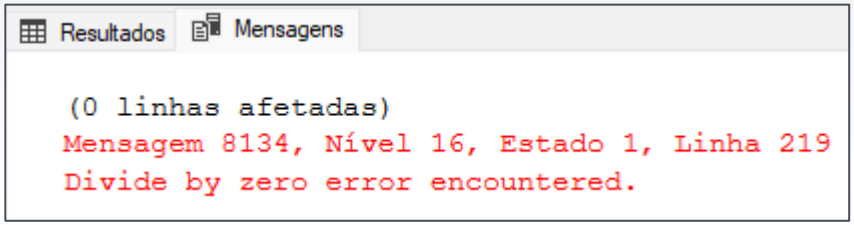
```
BEGIN TRY
  SELECT 1/0;
END TRY
BEGIN CATCH

END CATCH
```



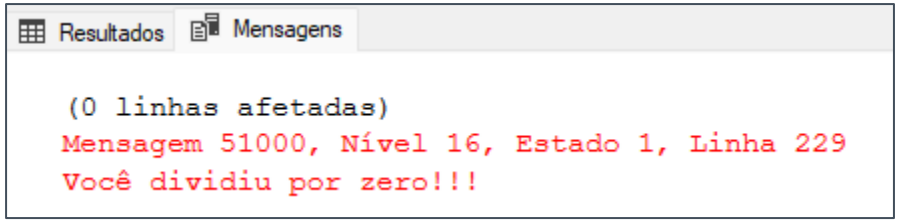
- ***Tratamento** de **Erros***
  - ***Exemplo (13):***
    - ***Adicionando** a **instrução THROW** ao **bloco CATCH**, **obrigando exibir** a **mensagem** de **erro***

```
BEGIN TRY
  SELECT 1/0;
END TRY
BEGIN CATCH
  THROW
END CATCH
```



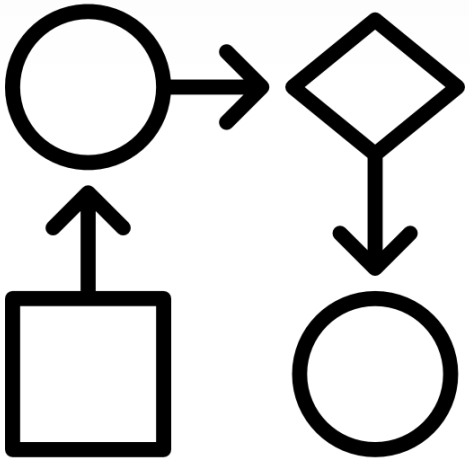
- ***Tratamento** de **Erros***
  - ***Exemplo (14):***
    - ***Personalizando** a **mensagem** de **erro** com o **THROW***

```
BEGIN TRY
  SELECT 1/0;
END TRY
BEGIN CATCH
  THROW 51000, 'Você dividiu por zero!!!', 1;
END CATCH
```



T-SQL

- ***Palavras-chave** de **Controle** de **Fluxo***
  - ***Determinar quando** e **como** o **código deve reagir** ou **funcionar** junto é uma **parte essencial** de **qualquer linguagem** de **programação T-SQL inclui** um **conjunto** de **palavras-chave** que **permitem agrupar** uma **série** de **instruções** e **tomar decisões** em **tempo** de **execução baseadas** na **lógica dentro** do **código***



T-SQL

- ***Palavras-chave** de **Controle** de **Fluxo***
  - *As **palavras-chave** são as **seguintes:***
    - ***BEGIN...END***
    - ***BREAK***
    - ***CONTINUE***
    - ***GOTO***
    - ***IF...ELSE***
    - ***RETURN***
    - ***WAITFOR***
    - ***WHILE***

T-SQL

- ***BEGIN...END***
  - ***Engloba** um **grupo** ou **uma série** de **instruções T-SQL***
  - ***Blocos BEGIN...END** podem ser **aninhados***

```
BEGIN
   instrução_sql | bloco_de_instruções
END
```

T-SQL

- ***BEGIN…END***
  - ***Exemplo (15):***
    - ***Utilizamos*** *uma* ***variável*** *para* ***limitar*** *o* ***resultado*** *a apenas os* ***funcionários*** *cuja* ***data*** *de* ***admissão*** *seja* ***inferior*** *ou* ***igual*** *ao* ***valor atribuído*** *a essa* ***variável***

```
BEGIN
  DECLARE @StartingHireDate DATETIME = '12/31/2001'

  SELECT e.BusinessEntityID, p.FirstName, p.LastName, e.HireDate
  FROM HumanResources.Employee e
  INNER JOIN Person.Person p
    ON (e.BusinessEntityID = p.BusinessEntityID)
  WHERE HireDate <= @StartingHireDate
END
```

T-SQL

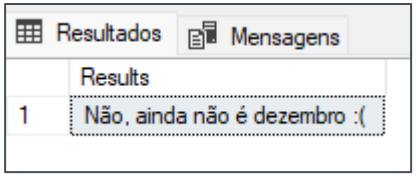
- ***IF...ELSE***
  - *Diz para a **linguagem** de **programação** para que **execute uma instrução** ou um **conjunto** de **instruções T-SQL** se a **condição especificada** for **satisfeita**, ou **outra instrução** ou **um conjunto** de **instruções T-SQL**, **se não for***
  - *O **IF** pode **existir sem** o **ELSE**, mas o **ELSE não** pode **existir sem** o **IF***

```
IF expressão_booleana { instrução_sql | bloco_de_instruções }
  [ ELSE { instrução_sql | bloco_de_instruções } ]
```

T-SQL

- ***IF...ELSE***
  - ***Exemplo (16):***
    - *Como o **mês não** é **dezembro** (December), a **expressão booleana retornou falso***

```
IF (DATENAME(M, GETDATE()) = 'December')
BEGIN
  SELECT 'Chegando o Natal!!!' Results
END
ELSE
BEGIN
  SELECT 'Não, ainda não é dezembro :(' Results
END
```



T-SQL

- ***WHILE***
  - *É um* ***mecanismo*** *de* ***loop baseado*** *em* ***uma expressão booleana***
  - ***Enquanto*** *a* ***expressão*** *for* ***avaliada*** *como* ***verdadeira****, a* ***instrução*** *ou* ***bloco*** *de* ***código T-SQL*** *especificado* ***executará***
  - ***Duas palavras-chave opcionais****,* ***BREAK*** *e* ***CONTINUE****, podem ser* ***incluídas*** *com a* ***palavra-chave WHILE*** *para* ***ajudar*** *na* ***lógica*** *de* ***controle dentro*** *do* ***loop***
  - *Se, em* ***qualquer ponto durante*** *o* ***loop WHILE****, a* ***palavra-chave BREAK*** *fizer a* ***execução*** *da* ***query terminar****, o* ***código T-SQL após*** *a* ***palavra-chave END será executado***

T-SQL

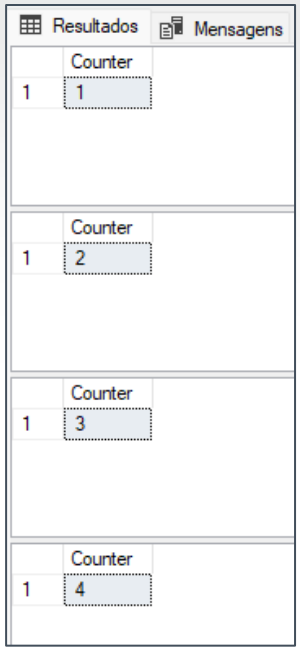
- ***WHILE***
  - *A **palavra-chave** **CONTINUE**, por outro lado, **faz** o **loop** **reiniciar***
  - *As **instruções** **após** a **palavra-chave** **CONTINUE** são **ignorados***

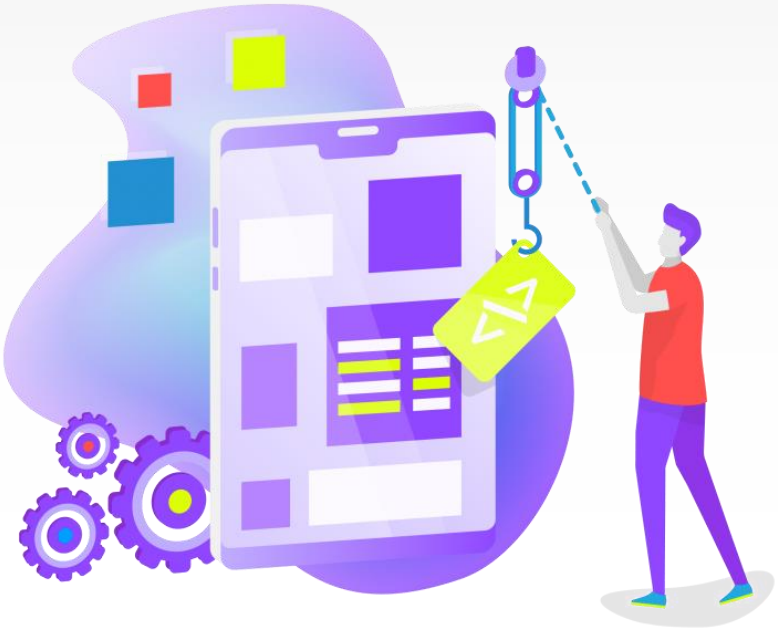
```
WHILE expressão_booleana
  { instrução_sql | bloco_de_instruções | BREAK | CONTINUE }
```

T-SQL

- ***WHILE***
  - ***Exemplo (17):***
    - ***Utilizando** o **loop** **WHILE** com **BREAK** e **CONTINUE***

```
DECLARE @count INT = 0
WHILE (@count < 10)
BEGIN
  SET @count = @count + 1;
    IF (@count < 5)
        BEGIN
          SELECT @count AS Counter
          CONTINUE;
        END
        ELSE
          BREAK;
END
```





EXERCÍCIOS

Referências

Noble, E.; Pro T-SQL 2019 Toward Speed, Scalability, and Standardization for SQL Server Developers. Apress, 2020.

Ben-Gan, I.; Microsoft SQL Server 2012 T-SQL Fundamentals. Pearson Education. 2012.

Lahoud, P.; Lopes, P.; T-SQL Querying: A guide to developing efficient and elegant T-SQL code. Packt Publishing. 2019.