Banco de Dados

Módulo Intermediário

VIEWS

Views

- ***O que são views?***
  - *É um **objeto** ou **tabela virtual** cujo **conjunto** de **resultados** é **extraído** de **uma query***
  - *É muito **parecida** com uma **tabela real**, pois contém **colunas** e **linhas** de **dados***
  - *O **único momento** em que **uma view** é **materializada** (**armazenada** em **disco**) é **quando** é **indexada***

Views

- ***O que são views?***
  - ***Alguns usos típicos** de **views:***
    - ***Filtrar dados** de **tabelas subjacentes***
    - ***Filtrar dados** para **propósito** de **segurança***
    - ***Centralizar dados distribuídos** por **vários servidores***
    - ***Criar** um **conjunto** de **dados reutilizável***
  - *As **views** podem ser **criadas** usando a **interface gráfica** de **usuário** (GUI) do **Microsoft SQL Server Management Studio** (SSMS) ou com **T-SQL***

Views

- ***O que são views?***
  - ***Antes** de **criar views**, é preciso **entender** o **seguinte:***
    - *As **views**, muitas vezes, são **usadas** como uma **camada** de **abstração** por **desenvolvedores** de **BD***
    - *As vezes, também são **usadas** para dar **segurança** aos **dados** de **diversas maneiras***
    - ***Exemplo:** podemos **criar** uma **view** que **exponha somente dados específicos**. Dessa forma, em vez de **conceder** aos **usuários permissões** para a **tabela subjacente**, podemos **conceder permissões** para a **view** que **expõe algumas colunas***
    - ***Não** se deve **utilizar SELECT *** em **views***

Views

- ***O que são views?***
  - ***Antes** de **criar views**, é preciso **entender** o **seguinte:***
    - *Caso **SELECT *** seja usado, quando os **esquemas** de **tabela** **mudarem**, o mesmo **acontecerá** com a **listagem** de **colunas** das **views***
    - *Ao se **escrever queries**, somente as **colunas exigidas** devem ser **retornadas***
    - ***Não** se deve usar **ORDER BY** em **views**, pois **não** **serão válidas**; só é **válido** quando **usado** com **TOP**. Nesse caso, **ORDER BY** é usado para **determinar** quais **linhas** serão **retornadas***

Views

- ***O que são views?***
  - ***Exemplo (01):***
    - ***Criando** uma **view** em **T-SQL***

```
USE MyAdventureWorks;

CREATE VIEW dbo.vwEmployeeInformation
AS
SELECT p.Title, p.FirstName, p.MiddleName,
       p.LastName, e.JobTitle, e.BirthDate, e.Gender
FROM Person.Person p
INNER JOIN HumanResources.Employee e
   ON (p.BusinessEntityID = e.BusinessEntityID);
```

Views

- ***O que são views?***
  - ***Exemplo (02):***
    - ***Invocando a view*** *vwEmployeeInformation*

```
USE MyAdventureWorks;

SELECT *
FROM dbo.vwEmployeeInformation;
```

	Title	FirstName	MiddleName	LastName	JobTitle	BirthDate	Gender
1	NULL	Ken	J	Sánchez	Chief Executive Officer	1963-03-02	M
2	NULL	Terri	Lee	Duffy	Vice President of Engineering	1965-09-01	F
3	NULL	Roberto	NULL	Tamburello	Engineering Manager	1968-12-13	M
4	NULL	Rob	NULL	Walters	Senior Tool Designer	1969-01-23	M
5	Ms.	Gail	A	Erickson	Design Engineer	1946-10-29	F
6	Mr.	Jossef	H	Goldberg	Design Engineer	1953-04-11	M
7	NULL	Dylan	A	Miller	Research and Development Manager	1981-03-27	M
8	NULL	Diane	L	Margheim	Research and Development Engineer	1980-07-06	F
9	NULL	Gigi	N	Matthew	Research and Development Engineer	1973-02-21	F
10	NULL	Michael	NULL	Raheem	Research and Development Manager	1979-01-01	M

Views

- ***Views Indexadas***
  - *Uma **view indexada** é **diferente** das **outras views**, pois é **materializada** e **armazenada** no **disco** da **mesma maneira** que **uma tabela***
  - *Um **ponto interessante** sobre as **views indexadas** é que o **otimizador** de **queries** pode **referenciar** uma **view** para **aumentar** o **desempenho**, mesmo que ela **não** seja **referenciada** na **query***
  - *Esse **recurso** só está **disponível** na **edição Enterprise** do **SQL Server***

Views

- ***Requisitos** da **Tabela Referenciada***
  - ***Antes** de **criar** uma **view indexada**, precisamos **garantir** que todas as **tabelas referenciadas satisfaçam alguns requisitos***
  - ***Todas** elas **devem estar contidas** no **mesmo BD***
  - *Se quaisquer **colunas calculadas** nas **tabelas** de base **não** forem **determinísticas**, elas **devem ser removidas***
  - *A **definição** de **determinística** é: sempre retorna o mesmo valor ou conjunto de resultados*

Views

- ***Requisitos** da **Tabela Referenciada***
  - *Como um **requisito** de uma **view indexada** é ser **determinística**, **todas** as **colunas** da **tabela** de **base também** devem ser **determinísticas***
  - *Para **determinar** se a **coluna** é **determinística**, podemos usar o **código** a **seguir**, que utiliza a **função escalar** COLUMNPROPERTY*

```
USE MyAdventureWorks;
GO
SELECT COLUMNPROPERTY(OBJECT_ID('Sales.SalesOrderDetail'),
                'LineTotal', 'IsDeterministic') AS 'Column Length';
GO
```

Views

- ***Requisitos** da **Tabela Referenciada***
  - ***Além** de **ser determinística**, a **coluna calculada** também **pode ser marcada** como **PERSISTED***
  - *Isso **depende** do **tipo** de **dado** ser **impreciso***
  - *Todo **tipo** de **dado float** ou **real** é **considerado impreciso** e **não** pode ser a **chave** de um **índice**, a **não** ser que seja **marcado** como **PERSISTED***
  - *Por último, é **necessário** que as **opções ANSI_NULLS** e **QUOTED_IDENTIFIER** tenham sido **definidas** como **true** quando as **tabelas referenciadas** foram **criadas***

Views

- ***Requisitos** de **Views Indexadas***
  - ***Além** dos **requisitos** das **tabelas referenciadas**, precisamos **garantir mais alguns detalhes**, antes de **criar** uma **view** e como **parte** da **criação***
  - ***As seguintes opções** de **SET** devem ser **ON:***
    - ***ANSI_NULLS***
    - ***ANSI_PADDING***
    - ***ANSI_WARNINGS***
    - ***ARITHABORT***
    - ***CONCAT_NULL_YIELDS_NULL***
    - ***QUOTED_IDENTIFIER***

Views

- ***Requisitos** de **Views Indexadas***
  - *A **opção** **NUMERIC_ROUNDABORT** deve ser **definida** como **OFF** (é preciso definir essas opções antes de criar a view)*
  - *Na sequência, **precisamos verificar** se a **view** é **determinística**, isso significa que a **view retornará** os mesmos **valores** sempre que for **consultada***
  - ***Quando** a **view** está sendo **criada** é preciso **utilizar** a **opção** **WITH SCHEMABINDING**, a qual **vincula** a **view** ao **esquema** das **tabelas subjacentes***
  - *Por fim, o **primeiro índice** deve ser **UNIQUE CLUSTERED***

Views

- ***Requisitos** de **Views Indexadas***
  - ***Exemplo (03): - Parte 1***
    - ***Criando** uma **view indexada***

```
USE MyAdventureWorks;

GO
-- Configura as opções para suportar views indexadas
SET NUMERIC_ROUNDABORT OFF;
SET ANSI_PADDING, ANSI_WARNINGS, CONCAT_NULL_YIELDS_NULL,
    ARITHABORT, QUOTED_IDENTIFIER, ANSI_NULLS ON;

GO
-- Verifica se já existe uma view com o mesmo nome
IF (OBJECT_ID('Purchasing.vwPurchaseOrders')) IS NOT NULL
  DROP VIEW Purchasing.vwPurchaseOrders
```

Views

- ***Requisitos** de **Views Indexadas***
  - ***Exemplo (03): - Parte 2***
    - ***Criando** uma **view indexada***

```
GO
-- Cria a view
CREATE VIEW Purchasing.vwPurchaseOrders
WITH SCHEMABINDING
AS
  SELECT
        poh.OrderDate,
        pod.ProductID,
        SUM(poh.TotalDue) TotalDue,
        COUNT_BIG(*) POCount
  FROM Purchasing.PurchaseOrderHeader poh
  INNER JOIN Purchasing.PurchaseOrderDetail pod
    ON (poh.PurchaseOrderID = pod.PurchaseOrderID)
  GROUP BY poh.OrderDate, pod.ProductID;
```

Views

- ***Requisitos** de **Views Indexadas***
  - ***Exemplo (03): - Parte 3***
    - ***Criando** uma **view indexada***

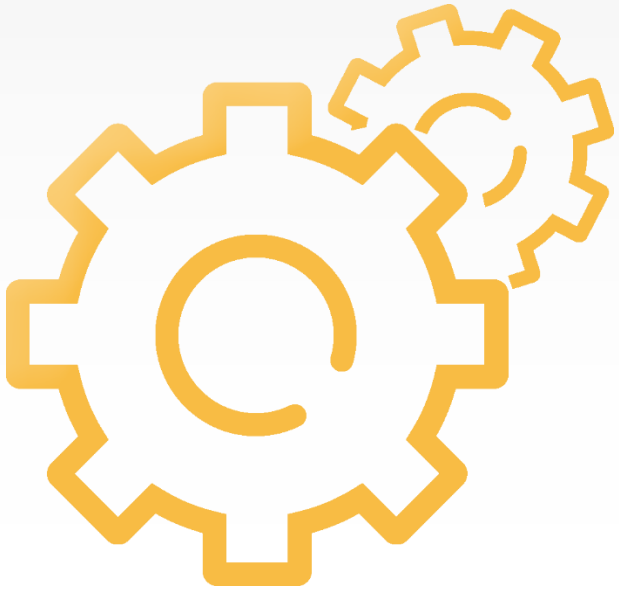
```
GO

-- Adiciona um índice clusterizado único
CREATE UNIQUE CLUSTERED INDEX CIX_vwPurchaseOrders_OrderDateProductID
ON Purchasing.vwPurchaseOrders(OrderDate, ProductID)

SELECT *
FROM Purchasing.vwPurchaseOrders;
```

Views

- ***Requisitos** de **Views Indexadas***
  - ***Observação:***
    - ***Assim** como **acontece** com a **view normal**, agora podemos **escrever queries** para **acessar** esses **dados***
    - *A **vantagem** é que **outras queries** que **não referenciam** a **view diretamente** podem ser **usadas** pelo **otimizador** para **aumentar** o **desempenho***
    - *A **desvantagem** é que **precisamos manter** o **índice** e então essa **view passará** a **consumir espaço** em **disco***
    - *À **medida** que os **dados aumentarem** nas **tabelas subjacentes**, o **mesmo acontecerá** com as **views** (estaremos armazenando várias cópias dos mesmos dados)*



Funções Definidas pelo Usuário

Functions

- ***Funções***
  - *As **funções definidas** pelo **usuário** são **semelhantes** às **funções** de **outras linguagens** de **programação***
  - *O **SQL Server** permite **criar dois tipos** de **funções: escalares** e **table-valued***
  - *Essa **funções possibilitam** um **tipo** de **programação modular**, onde o **código** e a **lógica** podem ser **incluídos dentro** da **função***
  - ***Outros aplicativos**, **rotinas** e **objetos** de **BD** podem **utilizar** a **função** (estratégia que permite determinar padrões e controlar como o código é desenvolvido e distribuído)*

Functions

- ***Funções***
  - ***São códigos*** *em* ***T-SQL*** *que* ***podem aceitar parâmetros, efetuar cálculos lógicos*** *e* ***complexos****, e* ***retornar dados***
  - *As* ***funções escalares*** ***retornam*** *um* ***único valor*** *e as* ***funções table-valued retornam*** *um* ***conjunto*** *de* ***resultados***
  - *As* ***funções*** *podem ser* ***usadas*** *como* ***CHECK CONSTRAINT*** *em* ***tabelas****, por* ***views****, para* ***definir uma coluna****, em uma* ***instrução SELECT*** *e de* ***muitas outras maneiras***

Functions

- ***Funções** versus **Stored Procedures***
  - ***Embora** as **funções** sejam **muito parecidas** com os **stored procedures** (procedimentos armazenados), elas **diferem** de **várias maneiras***
  - *A **diferença mais notável** é que as **funções table-valued podem** ser **utilizadas** em uma **instrução SELECT**, de modo que podem ser **unidas** a **tabelas**, **views** e até **outras funções***
  - *Os **stored procedures não** podem ser **usados dessa maneira***

Functions

- ***Funções Escalares***
  - ***É uma rotina*** *que* ***retorna*** *um* ***único valor***
  - *Normalmente, essa* ***funções*** *são* ***utilizadas*** *para* ***centralizar*** *a* ***lógica*** *de um* ***cálculo complexo*** *que pode ser* ***utilizado*** *por* ***vários outros recursos*** *de* ***BD*** *ou de* ***aplicativo***

Functions

- ***Funções Escalares***
  - ***Sintaxe:***

```
CREATE FUNCTION [ nome_do_esquema. ] nome_da_função
( [ { @nome_do_parâmetro [AS][ nome_do_tipo_do_esquema. ] tipo_de_dado_do_parâmetro
  [ = padrão ] [ READONLY ] }
  [ ,...n ]
  ]
)
RETURNS tipo_de_dado_de_retorno
  [ WITH <opção_de_função> [ ,...n ] ]
  [ AS ]
  BEGIN
    corpo_da_função
    RETURN expressão_escalar
  END
  [ ; ]
```

Functions

- ***Parametrize Funções***
  - *A **parametrização** se **aplica** aos **dois tipos** de **funções***
  - *No **campo** da **programação** de **funções** com **T-SQL**, um **parâmetro** é **um valor** de **entrada** que **pode ser passado** para o **código** a partir da **função chamadora** (**invocadora**)*
  - *Um **parâmetro** pode ser **definido** como uma **constante**, uma **coluna** de **uma tabela**, uma **expressão** e **outros valores***

Functions

- ***Parametrize Funções***
  - *As **funções** podem **conter três tipos** de **parâmetros:***
    - ***Entrada:** é o **valor passado** para o **corpo** da **função***
    - ***Opcional: não** é **exigido** para se **executar** a **função***
    - ***Padrão: indica** quando um **valor** é **atribuído** ao **parâmetro durante** a **criação** (é o valor especificado quando a função é criada)*
  - ***Exemplo:***

```
-- Parâmetro de Entrada
CREATE FUNCTION dbo.Input
@parameter1 INT
...
```

Functions

- ***Parametrize Funções***
  - ***Exemplo:***

```
-- Parâmetro Opcional
CREATE FUNCTION dbo.Optional
@parameter1 INT = NULL
...

-- Parâmetro Padrão
CREATE FUNCTION dbo.Default
@parameter1 INT = 1
...
```

Functions

- ***Parametrize Funções***
  - *Se o **parâmetro** de **entrada não** tem um **valor padrão**, **deve** ser **fornecido** um **valor** quando a **função for chamada***
  - *O **parâmetro opcional** é **definido** como **NULL durante** a **criação**, de modo que, **quando** a **função** for **utilizada**, pode ser **chamada sem** o **fornecimento** de **um valor***
  - *O **parâmetro padrão recebe** um **valor durante** a **criação**, mas quando a **função** é **executada**, é possível **especificar DEFAULT** no **lugar** do **parâmetro***

Functions

- ***Parametrize Funções***
  - ***Exemplo (04): - Parte 1***
    - ***Criando*** *uma* ***função escalar*** *a qual será usada para* ***retornar*** *a* ***idade*** *do* ***empregado***

```
USE MyAdventureWorks;

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

Functions

- ***Parametrize Funções***
  - ***Exemplo (04): - Parte 2***
    - ***Criando** uma **função escalar** a qual será usada para **retornar** a **idade** do **empregado***

```
CREATE FUNCTION dbo.GetEmployeeAge(@BirthDate DATETIME)
  RETURNS INT
AS
BEGIN
  -- Declara a variável de retorno
  DECLARE @Age INT
  -- Adicione as instruções T-SQL para calcular o valor de retorno
  SELECT @Age = DATEDIFF(DAY, @BirthDate, GETDATE())
  -- Retorna o resultado da função
  RETURN @Age
END;
GO
```

Functions

- ***Parametrize Funções***
  - ***Exemplo (04): - Parte 3***
    - ***Invocando a função*** *GetEmployeeAge*

```
USE MyAdventureWorks;

SELECT TOP(10)
        p.FirstName, p.LastName, e.BirthDate,
        dbo.GetEmployeeAge(BirthDate) EmployeeAge
FROM HumanResources.Employee e
INNER JOIN Person.Person p
  ON (e.BusinessEntityID = p.BusinessEntityID);
```

Functions

- ***Parametrize Funções***
  - ***Exemplo (04): - Parte 4***
    - ***Invocando a função*** *GetEmployeeAge*

	FirstName	LastName	BirthDate	EmployeeAge
1	Ken	Sánchez	1963-03-02	21807
2	Terri	Duffy	1965-09-01	20893
3	Roberto	Tamburello	1968-12-13	19694
4	Rob	Walters	1969-01-23	19653
5	Gail	Erickson	1946-10-29	27775
6	Jossef	Goldberg	1953-04-11	25419
7	Dylan	Miller	1981-03-27	15207
8	Diane	Margheim	1980-07-06	15471
9	Gigi	Matthew	1973-02-21	18163
10	Michael	Raheem	1979-01-01	16023

Functions

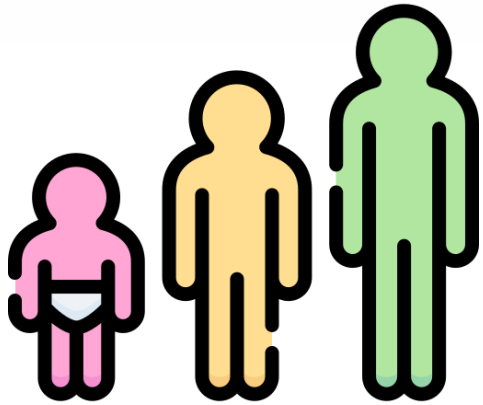
- ***Parametrize Funções***
  - *As **funções escalares** são **utilizadas** com a **mesma sintaxe** das **funções predefinidas***
  - *Se o **parâmetro** é **opcional**, podemos **chamar** a **função sem especificar um valor***
  - *Se um **valor padrão** fosse **atribuído** à **função**, a **chamada** da **função** seria **semelhante** à **sintaxe:***

```
dbo.GetEmployeeAge(DEFAULT)
```

  - *A **palavra-chave DEFAULT** diz ao **SQL Server Engine** para que **use** o **valor atribuído** ao **parâmetro quando foi criado***

Functions

- ***Parametrize Funções***
  - *Uma **vantagem importante** de **usar funções** é que, agora, em vez de **efetuar cálculo**, **uma função** pode ser **utilizada** para **retornar** a **idade** (**age**) – exemplo anterior*
  - ***Essa função** pode ser **reutilizada** por **outros programas**, **oferecendo** um **mecanismo consistente** para **calcular** os **dados***



Functions

- ***Parametrize Funções***
  - ***Exemplo (05): - Parte 1***
    - *Alterando a **função** GetEmployeeAge para **retornar** a **idade** em **anos** e **não** **dias***

```
ALTER FUNCTION [dbo].[GetEmployeeAge](@BirthDate DATETIME)
RETURNS INT
AS
BEGIN
  -- Declara a variável de retorno
  DECLARE @Age INT
  -- Adicione as instruções T-SQL para calcular o valor de retorno
  SELECT @Age = DATEDIFF(Year, @BirthDate, GETDATE())

  -- Retorna o resultado da função
  RETURN @Age
END;
```

Functions

- ***Parametrize Funções***
  - ***Exemplo (05): - Parte 2***
    - ***Invocando a função GetEmployeeAge, após a alteração***

```
USE MyAdventureWorks;

SELECT TOP(10)
        p.FirstName, p.LastName, e.BirthDate,
        dbo.GetEmployeeAge(BirthDate) EmployeeAge
FROM HumanResources.Employee e
INNER JOIN Person.Person p
  ON (e.BusinessEntityID = p.BusinessEntityID);
```

Functions

- ***Parametrize Funções***
  - ***Exemplo (05): - Parte 3***
    - ***Invocando a função GetEmployeeAge, após a alteração***

	FirstName	LastName	BirthDate	EmployeeAge
1	James	Hamilton	1977-02-07	45
2	Peter	Krebs	1976-12-04	46
3	Jo	Brown	1950-11-09	72
4	Guy	Gilbert	1976-05-15	46
5	Jill	Williams	1973-07-19	49
6	Mary	Gibson	1956-10-14	66
7	Sariya	Harnpadoungsataya	1981-06-21	41
8	Terry	Eminhizer	1980-03-07	42
9	Annik	Stahl	1971-01-27	51
10	Suchitra	Mohan	1981-07-11	41

Functions

- ***Parametrize Funções***
  - ***Exemplo (06):***
    - *Para **remover** a **função**, **execute** o **código T-SQL (DDL)***

```
DROP FUNCTION dbo.GetEmployeeAge;
```

Functions

- ***Execute Funções Escalares***
  - *As **funções escalares** podem ser **chamadas** por meio de **dois métodos:***
    - ***Dentro** de **uma instrução SELECT***
    - ***Pelo** uso da **palavra-chave EXECUTE***
  - ***Independentemente** do **método** utilizado para **selecionar** a **saída**, se os **valores** de **parâmetro** forem **consistentes**, os **resultados** de **uma** ou **outra execução serão** os **mesmos***

Functions

- ***Invocando Funções Escalares InLine***
  - *Uma **função escalar** pode ser **incluída** em uma **instrução SELECT***
  - *Os **parâmetros** podem ser uma **coluna**, uma **constante** ou uma **expressão***

```
SELECT dbo.GetEmployeeAge('07/31/1977') Result;
```

  - ***Esse é** o **uso típico** de **uma função escalar***
  - ***Esse método** é **muito simples** e apresenta um **desafio** somente quando **vários tipos** de **parâmetros** (entrada, opcional e padrão) são **especificados***

Functions

- ***Invocando Funções Escalares InLine***
  - ***Se existe apenas*** *um* ***parâmetro*** *de* ***entrada*** *e* ***qualquer combinação*** *de* ***outros tipos*** *de* ***parâmetros****, devemos* ***garantir*** *que a* ***ordem*** *em que eles* ***são passados corresponda*** *à* ***ordem*** *na qual* ***são especificados*** *na* ***função***

Functions

- ***Invocando Funções Escalares InLine***
  - ***Exemplo (07): - Parte 1***
    - ***Utilizando dois tipos** de **parâmetros**, padrão e opcional*

```
USE MyAdventureWorks;
GO
IF (OBJECT_ID('dbo.GetEmployeeAge')) IS NOT NULL
  DROP FUNCTION dbo.GetEmployeeAge
GO
```

Functions

- ***Invocando Funções Escalares InLine***
  - ***Exemplo (07): - Parte 2***
    - ***Utilizando dois tipos** de **parâmetros**, padrão e opcional*

```
CREATE FUNCTION [dbo].[GetEmployeeAge](
        @BirthDate DATETIME = '05/26/1972', -- DEFAULT
        @Temp DATETIME = NULL ) -- OPCIONAL
RETURNS INT
AS
BEGIN
  -- Declara variável de retorno
  DECLARE @Age INT
  -- Adiciona as instruções T-SQL para calcular o valor de retorno
  SELECT @Age = DATEDIFF(Year, @BirthDate, GETDATE())
  -- Retorna o resultado da função
  RETURN @Age
END;
```

Functions

- ***Invocando Funções Escalares InLine***
  - ***Exemplo (08):***
    - ***Invocando a função de diversas maneiras***

```
-- Parâmetro de entrada único
SELECT dbo.GetEmployeeAge(DEFAULT, NULL);
SELECT dbo.GetEmployeeAge('05/26/1972', '');
SELECT dbo.GetEmployeeAge('05/26/1972', NULL);

-- O primeiro parâmetro é padrão e o segundo é de entrada
SELECT dbo.GetEmployeeAge(DEFAULT, '01/10/1972');
SELECT dbo.GetEmployeeAge('05/26/1972', '01/10/1972');
```

Functions

- ***Invocando Funções Escalares** com **EXECUTE***
  - *Uma **função escalar** também pode ser **invocada** com a **palavra-chave EXECUTE***
  - *Por enquanto, é **suficiente** saber que é **possível usar** essa **palavra-chave** para **executar funções escalares***

Functions

- ***Invocando Funções Escalares** com **EXECUTE***
  - ***Exemplo (09): - Parte 1***
    - ***Para obter** a **saída** de uma **função escalar** com a **palavra-chave EXECUTE**, devemos **declarar** a **variável** que **armazenará** a **saída***

```
USE MyAdventureWorks;
GO
DECLARE @Age INT;
EXECUTE @Age = dbo.GetEmployeeAge @BirthDate = '07/31/1977';
SELECT @Age;
```

Functions

- ***Invocando Funções Escalares** com **EXECUTE***
  - ***Exemplo (09): - Parte 2***
    - *É **declarada** a **variável** @Age. A **função** é **invocada** usando a **palavra-chave** EXECUTE, atribuindo o **valor** de **retorno** à **variável**. Por fim, é **executada** uma **instrução** SELECT para **obter** o **resultado***
    - *O **nome** do **parâmetro** **não** é **obrigatório**. Contudo, se **especificarmos vários parâmetros** de **diferentes tipos**, devemos **mencionar** os **nomes explicitamente***
    - *Isso **garante** que o **valor correto** seja **atribuído** ao **parâmetro apropriado***
    - ***Independentemente** dos **tipos** de **parâmetros** e do **método** de **execução**, o **segredo** é **especificar** um **valor** para **cada parâmetro** e **garantir** que a **ordem** dos **valores corresponda** à **ordem** dos **parâmetros***

Functions

- ***Funções Table-Valued***
  - ***Existem dois tipos*** *de* ***funções table-valued:***
    - ***InLine***
    - *De* ***várias instruções***
  - *A* ***função InLine*** *simplesmente* ***retorna*** *um* ***conjunto*** *de* ***resultados*** *e a* ***função*** *de* ***várias instruções*** *oferece a* ***capacidade*** *de* ***incluir lógica*** *dentro do* ***corpo*** *da* ***função***
  - ***Ambas retornam*** *um* ***resultado completo****, semelhante a fazer uma* ***seleção*** *em uma* ***tabela*** *ou* ***view****, mas a* ***função*** *de* ***várias instruções*** *pode* ***efetuar lógica*** *e* ***retornar dados***

Functions

- ***Funções Table-Valued***
  - ***Sintaxe (table-valued inline):***

```
CREATE FUNCTION [ nome_do_esquema. ] nome_da_função
( [ { @nome_do_parâmetro [AS][ nome_do_tipo_do_esquema. ] tipo_de_dado_do_parâmetro
  [ = padrão ] [ READONLY ] }
  [ ,...n ]
  ]
)
RETURNS TABLE tipo_de_dado_de_retorno
  [ WITH <opção_de_função> [ ,...n ] ]
  [ AS ]
  RETURN [ ( ] inst_seleção [ ) ]
[ ; ]
```

Functions

- ***Funções Table-Valued***
  - ***Sintaxe (table-valued** de **várias instruções):***

```
CREATE FUNCTION [ nome_do_esquema. ] nome_da_função
( [ { @nome_do_parâmetro [AS][ nome_do_tipo_do_esquema. ] tipo_de_dado_do_parâmetro
  [ = padrão ] [ READONLY ] }
  [ ,...n ]
  ]
)
RETURNS @variável_de_retorno TABLE <definição_de_tipo_de_tabela>
  [ WITH <opção_de_função> [ ,...n ] ]
  [ AS ]
  BEGIN
    corpo_da_função
    RETURN
  END
  [ ; ]
```

Functions

- ***Funções Table-Valued***
  - *A **maior parte** do **código** é **opcional***
  - ***Note** que **ambas** podem **retornar** uma **tabela***
  - *A **função InLine** **retorna** somente o **resultado** de uma **instrução SELECT** como **conjunto**, enquanto a **função** de **várias instruções** utiliza uma **variável** de **tabela** que pode ser **definida***
  - *As **linhas** de **dados** são **adicionadas** à **tabela** de **acordo** com o **código** e podem ser **manipuladas antes** que os **dados** sejam **retornados***

Functions

- ***Funções Table-Valued***
  - ***Isso** é **diferente** da **função table-valued inline**, onde **qualquer manipulação** ou **filtragem** de **dados** deve ser **feita** na **própria query***

Functions

- ***Funções Table-Valued***
  - ***Exemplo (10): - Parte 1***
    - ***Função responsável** por **retornar** os **itens** de **linha** para **determinado** OrderId*

```
USE MyAdventureWorks;
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (10): - Parte 2***
    - ***Função responsável** por **retornar** os **itens** de **linha** para **determinado** OrderId*

```
CREATE OR ALTER FUNCTION dbo.GetOrderDetails(@SalesID INT)
  RETURNS TABLE
AS
  RETURN (
  SELECT sod.SalesOrderID, sod.SalesOrderDetailID,
         sod.CarrierTrackingNumber, p.Name ProductName, so.Description
  FROM Sales.SalesOrderDetail sod
  INNER JOIN Production.Product p ON (sod.ProductID = p.ProductID)
  INNER JOIN Sales.SpecialOffer so
        ON (sod.SpecialOfferID = so.SpecialOfferID)
  WHERE sod.SalesOrderID = @SalesID
)
GO
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (10): - Parte 3***
    - ***Invocando a função*** *GetOrderDetails*

```
USE MyAdventureWorks;

SELECT *
FROM dbo.GetOrderDetails(43659);
```

Functions

- ***Funções Table-Valued***
  - *Uma **função table-valued** pode ser **usada** da **mesma maneira** que uma **tabela** ou **view** é usada em uma **instrução SELECT***
  - *Podemos fazer uma **união** nessa **função** como se **ela fosse** uma **tabela** ou **inserir** os **resultados** em **uma variável** de **tabela** ou a **uma tabela temporária***
  - *O **uso** e os **tipos** dos **parâmetros** são iguais àqueles usados com **funções escalares** (deve-se ter **cuidado** ao **misturar parâmetros** de **entrada**, **padrão** e **opcionais**)*

```
SELECT *
FROM dbo.GetOrderDetails(43659, DEFAULT, NULL);
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (11): - Parte 1***
    - ***Criando** a **função** GetTotalPedido **responsável** por **calcular** o **valor total** para um **pedido específico***

```
USE [MyAdventureWorks];
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (11): - Parte 2***
    - ***Criando** a **função** GetTotalPedido **responsável** por **calcular** o **valor total** para um **pedido específico***

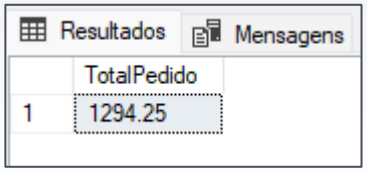
```
CREATE OR ALTER FUNCTION [dbo].[GetTotalPedido](@OrderID INT)
  RETURNS DECIMAL(7,2)
AS
BEGIN
  DECLARE @Total DECIMAL(7,2)
  SELECT @Total = SUM(SalesOrderDetail.UnitPrice * SalesOrderDetail.OrderQty)
  FROM Sales.SalesOrderDetail
  WHERE SalesOrderID = @OrderID;

  RETURN @Total
END;
GO
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (11): - Parte 3***
    - ***Invocando a função*** *GetTotalPedido*

```
SELECT dbo.GetTotalPedido(43660) TotalPedido;
```



Functions

- ***Funções Table-Valued***
  - ***Exemplo (12): - Parte 1***
    - ***Criando** a **função** GetDetalhesPedidoCliente onde **definimos** a **estrutura** da tabela PedidosCliente a qual será **retornada**. Usamos como **parâmetro** o CustomerID*

```
USE [MyAdventureWorks];
GO
CREATE OR ALTER FUNCTION [dbo].[GetDetalhesPedidoCliente]
  (@CustomerID NCHAR(5))
RETURNS @PedidosCliente TABLE(
                CustomerID              NCHAR(5),
                FirstName               NVARCHAR(25),
                MiddleName              NVARCHAR(25),
                LastName                NVARCHAR(25),
                OrderDate               DATETIME
) -- continua…
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (12): - Parte 2***
    - ***Criando** a **função** GetDetalhesPedidoCliente onde **definimos** a **estrutura** da tabela PedidosCliente a qual será **retornada**. Usamos como **parâmetro** o CustomerID*

```
AS
BEGIN
  INSERT INTO @PedidosCliente
  SELECT c.CustomerID, p.FirstName, p.MiddleName, p.LastName,
         c.ModifiedDate
  FROM Sales.Customer c
  INNER JOIN Person.Person p
    ON (c.CustomerID = p.BusinessEntityID)
  WHERE c.CustomerID = @CustomerID;

-- continua…
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (12): - Parte 3***
    - ***Criando** a **função** GetDetalhesPedidoCliente onde **definimos** a **estrutura** da tabela PedidosCliente a qual será **retornada**. Usamos como **parâmetro** o CustomerID*

```
IF (@@ROWCOUNT = 0)
 BEGIN
   INSERT INTO @PedidosCliente
       VALUES
       (NULL, 'Cliente não localizado',
       'Cliente não localizado',
       'Cliente não localizado', GETDATE())
 END
 RETURN
END;
GO
```

Functions

- ***Funções Table-Valued***
  - ***Exemplo (12): - Parte 4***
    - ***Testando a função*** *GetDetalhesPedidoCliente*

```
SELECT *
FROM dbo.GetDetalhesPedidoCliente(19056);
```

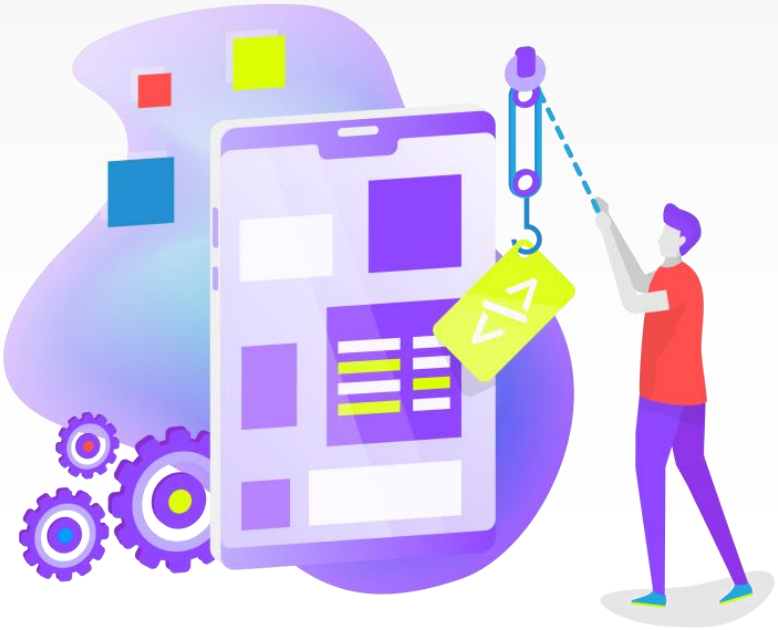
Resultados | Mensagens

	CustomerID	FirstName	MiddleName	LastName	OrderDate
1	19056	Craig	C	Ruiz	2008-10-13 11:15:07.263

```
SELECT *
FROM dbo.GetDetalhesPedidoCliente(99999);
```

Resultados | Mensagens

	CustomerID	FirstName	MiddleName	LastName	OrderDate
1	NULL	Cliente não localizado	Cliente não localizado	Cliente não localizado	2022-11-15 19:42:14.940



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.