Sintaxes estranhas no SQL Server

Leitores, como esta semana temos o Halloween acontecendo, resolvi montar uma coluna somente com alguns aspectos curiosos e estranhos sobre a sintaxe de algumas instruções Transact-SQL. Isso me faz lembrar do que alguém me disse uma vez :

- Computação não é uma ciência exata. É uma ciência exotérica.

Apesar do SQL Server ser um excelente gerenciador de banco de dados relacionais as vezes me deparo com situações muito inusitadas. Por isso resolvi citar algumas 'peculiaridades' do Transact-SQL:

1. Cláusula WITH da função OPENXML()

A função OPEMXML() tem como objetivo retornar em linhas e colunas informações contidas em um documento no formato XML. O que é estranho aqui é a sua sintaxe quando se utiliza a cláusula WITH , que é opcional mas quando utilizada deve ser fornecida depois de se fechar os parênteses de OPENXML(). Exemplo:

```
SELECT * FROM
OPENXML (@idoc, '/PRINCIPAL/AMIGA')
WITH ( NOME varchar(10),
COR_OLHO varchar(20),
IDADE INT,
TAGTOTAL ntext '@mp:xmltext')
```

Para mim mas parece que WITH faz parte do SELECT do que do OPENXML() .....

2. Stored Procedure SP_STORED_PROCEDURES

Esta Stored Procedure retorna todas as Stored Procedures que existem no banco de dados atual. O que parece esquisito aqui é o nome desta função. Puxa logo de cara já dá para saber o que esta Stored Procedure faz ??? Acho que um pouco mais de clareza na definição do nome ajudaria.

3. Stored Procedure SP_SERVEROPTION

Outra Stored Procedure onde o nome não ajuda muito. Esta daqui até que dá para arriscar um palpite: provavelmente deve servir para alterar ou visualizar uma opção do servidor. O curioso é que ela faz isso , mas somente para servidores remotos ou linkados!

4. Funções que começam com ::

É isso mesmo. O SQL Server possui algumas funções que começam com :: indicando que esta é uma função nativa do SQL Server que retorna uma tabela (table valued function). Por exemplo:

```
SELECT * FROM ::fn_helpcollations()
```

Isso viola a regra para a definição de identificadores regulares. Mas podem ser criados objetos começando com :: utilizando identificadores delimitados como no exemplo abaixo:

```
CREATE FUNCTION [::HORA]()
RETURNS DATETIME
AS
BEGIN

RETURN CONVERT(DATETIME,'01/01/2002')

END
```

5. Nome de coluna quando se utiliza FOR XML

Essa daqui é meio misteriosa. Sempre que utilizamos a cláusula FOR XML uma coluna com uma ou mais linhas é retornada contendo o que foi pedido para o banco em formato XML. O fato é que sempre o nome da coluna é o mesmo! Pois é e este nome da coluna é: XML_F52E2B61-18A1-11d1-B105-00805F49916B e independe do sistema operacional ou versão do SQL Server.

Depois que um usuário me perguntou fiquei curioso também. Por que será que a Microsoft forçou o nome da coluna com este identificador incompreensível ? Depois de um pouco de investigação concluí o mistério: colunas que tenham este nome de identificador são reconhecidas como colunas que contém dados em XML pelo provider OLE DB que já sabe como tratar dados neste formato.

6. ORDER BY 3

A primeira vez que vi este tipo de utilização do ORDER BY confesso que estranhei. Mas na verdade é muito útil identificar por um número qual é a coluna que se deseja ordenar. O SQL Server verifica a lista de coluna na cláusula SELECT que foi fornecida e numera internamente: a primeira coluna é 1, a segunda coluna é 2 e assim por diante. Com isso quando dizemos ORDER BY 3 em uma instrução SELECT estamos ordenando o resultado pela terceira coluna a partir da instrução SELECT.

7. Funções criada pelo usuário (UDF) sem acesso a tabelas de fora

Pode parecer estranho mais é isso mesmo. Quando criamos qualquer tipo de UDF , seja Scalar Function , Inline Table Valued Function ou Multi-Statement Table-valued , o SQL Server não permite que utilizemos valores que não sejam os passados como parâmetros ou façamos chamadas a funções não-determinísticas dentro do corpo da função.

Isso tem a ver com o fato de que somente podemos criar funções não-determinísticas no SQL Server. Definindo:

Funções determinísticas: São as funções que com o mesmo conjunto de parâmetros retornam valores diferentes a cada nova chamada. Exemplo: a função GETDATE().

Funções não-determinísticas: São as funções que com o mesmo conjunto de parâmetros sempre retornam o mesmo valor a cada nova chamada. Exemplo: a função DATALENGTH().

Porém pode ocorrer o caso em que uma função é determinística e não determinística ao mesmo tempo , dependendo do que foi passado como parâmetro para ela. Para mais informações sobre este tópico recomendo uma consulta ao Books Online.

A alternativa para este caso é construir Stored Procedures que podem acessar qualquer tabela do banco de dados.

8. Tipo de dado SQL_VARIANT

A partir do SQL Server 2000 o tipo de dados sql_variant foi introduzido para permitir que uma mesma coluna guarde vários tipos de dados diferentes. Isso pode gerar situações onde fazemos uma instrução SQL supondo que todas as linhas da tabela vão possuir o mesmo comportamento (em uma soma , por exemplo) e como cada tipo de dados reage de maneira diferente podemos estar com um problema em nossas mãos. Para isso o SQL Server possui a função SQL_VARIANT_PROPERTY() que pode ajudar nestes casos. Um exemplo:

```
-- Criando a tabela com a coluna do tipo
CREATE TABLE T_VAR
(
COLUNA_A SQL_VARIANT NOT NULL
)

-- Inserindo diversos valores de tipos de dados diferentes na tabela
INSERT INTO T_VAR VALUES(1)
INSERT INTO T_VAR VALUES(CONVERT(VARCHAR,'A'))
INSERT INTO T_VAR VALUES(CONVERT(CHAR,'A'))
INSERT INTO T_VAR VALUES(1.0)
INSERT INTO T_VAR VALUES(CONVERT(BIT,0))
INSERT INTO T_VAR VALUES(NEWID())
INSERT INTO T_VAR VALUES(GETDATE())

-- Verificando qual linha possui qual tipo
SELECT SQL_VARIANT_PROPERTY(COLUNA_A,'BaseType') , COLUNA_A
FROM T_VAR
```

9. Promoção de Tipos

O SQL Server faz promoção do tipo de dados de acordo com os tipos de dados que estão na expressão. Por exemplo:

```
SELECT (5/2.0) + 3 , (5/2) + 3
```

As duas colunas acima retornam valores diferentes: na primeira coluna o resultado foi calculado com mais precisão de casas decimais e na segunda coluna não houve precisão nos cálculos , gerando um valor inteiro. Isso as vezes pode dar uma dor de cabeça em certas situações....

10. GO com comentário

Essa daqui chega até a ser engraçada. Geralmente quando comentamos algo em nosso código o interpretador/compilador ignora , correto ? Certo , tente fazer isso com um GO somente em uma linha , assim:

```
/*
Isso é um comentário
GO
Que gera erro se executado!
*/
```

Isso se deve ao fato de que para o SQL Server o comando GO somente é valido em algumas ferramentas , como o Query Analyser. Por isso toda a vez que o interpretar acha um GO , seja ele comentado ou não , ele é processado.

Grande abraço galera e até a próxima semana.

Mauro Pichiliani
pichiliani@originet.com.br