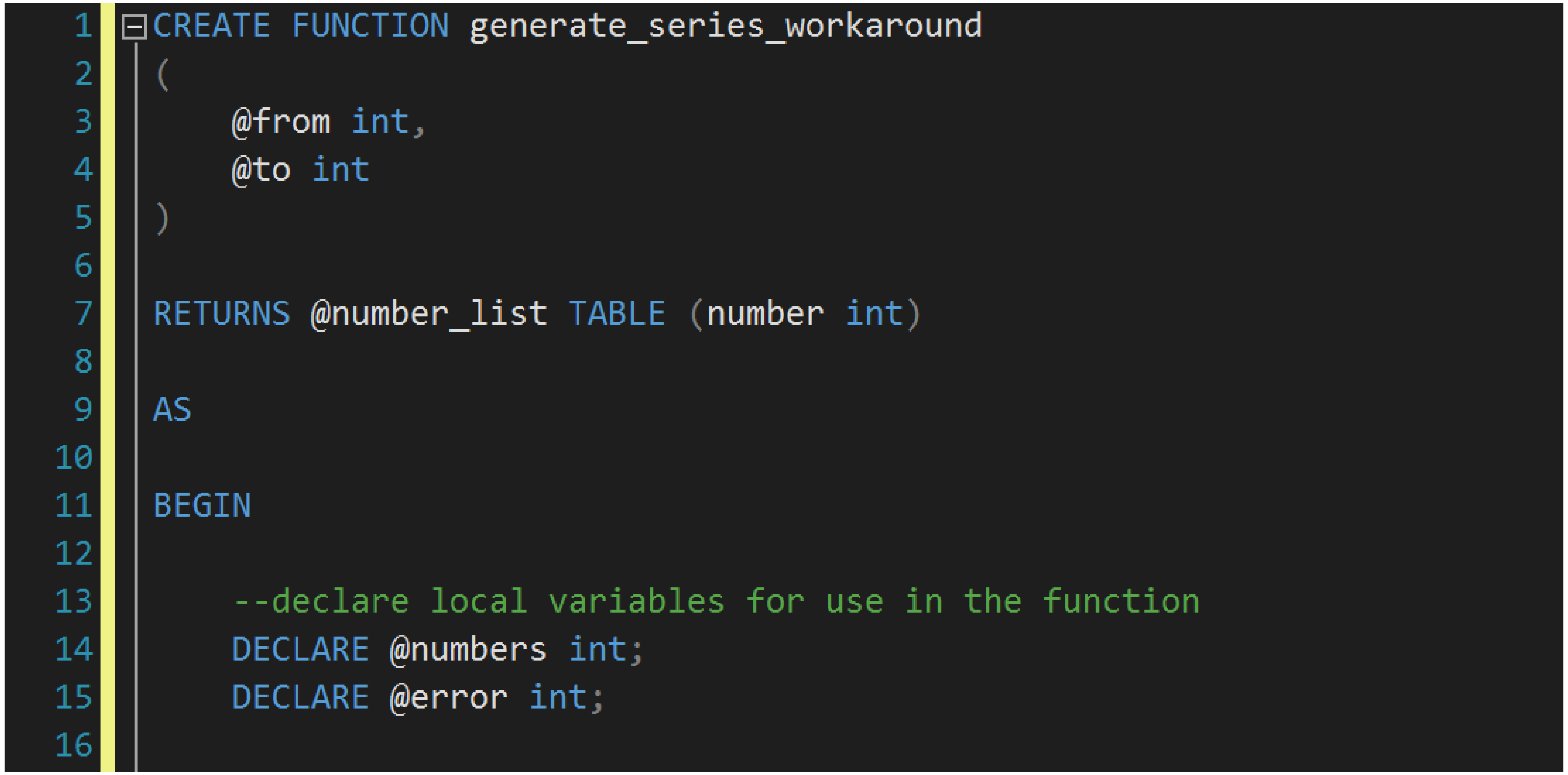
Table-valued functions (TVF) in SQL Server

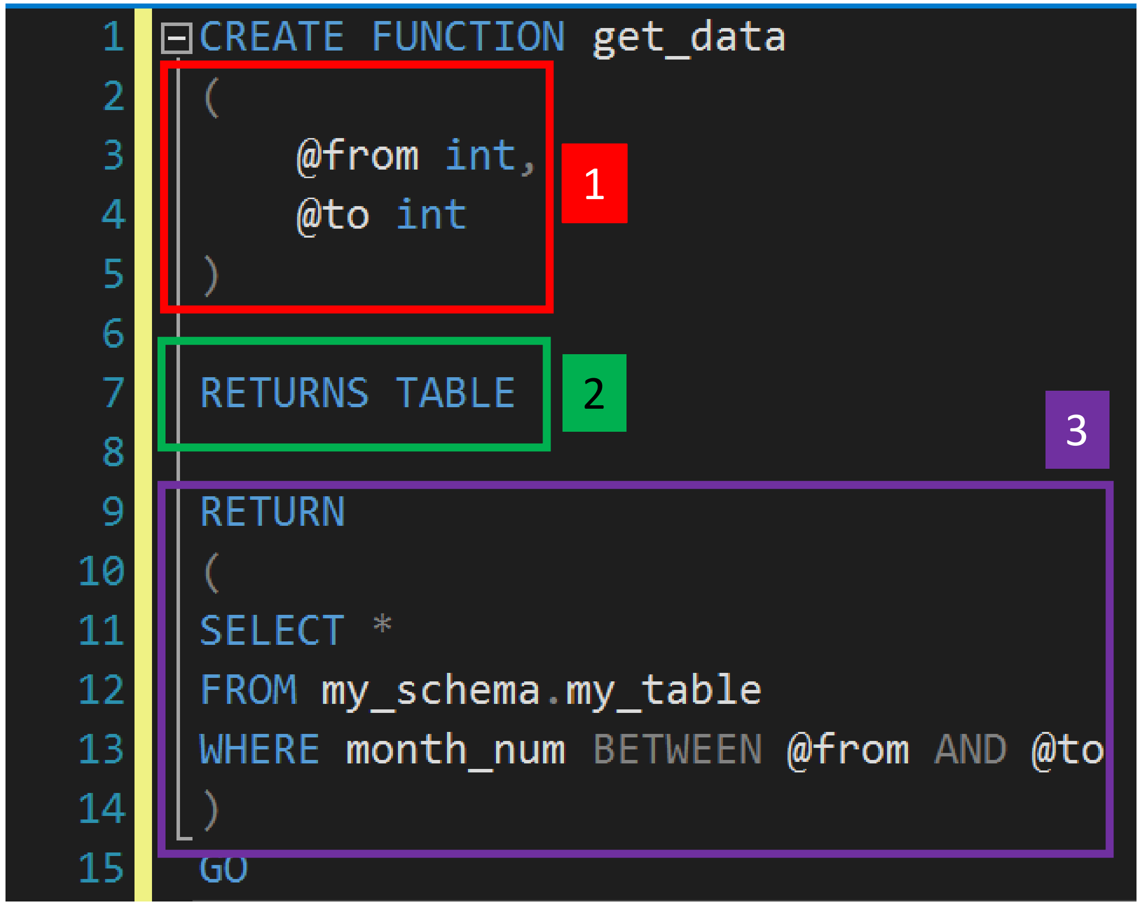


```sql
CREATE FUNCTION generate_series_workaround
(
    @from int,
    @to int
)

RETURNS @number_list TABLE (number int)

AS

BEGIN

    --declare local variables for use in the function
    DECLARE @numbers int;
    DECLARE @error int;

```

There are two types of TVF

1. Inline TVF		2. Multi-statement TVF
✓	Accepts parameters	✓
✓	Returns a table	✓
✓	Invoked in the FROM clause of a query	✓
✗	Allows multiple statements	✓
✗	Allows local variable declaration	✓
✗	Allows TRY/CATCH	✗

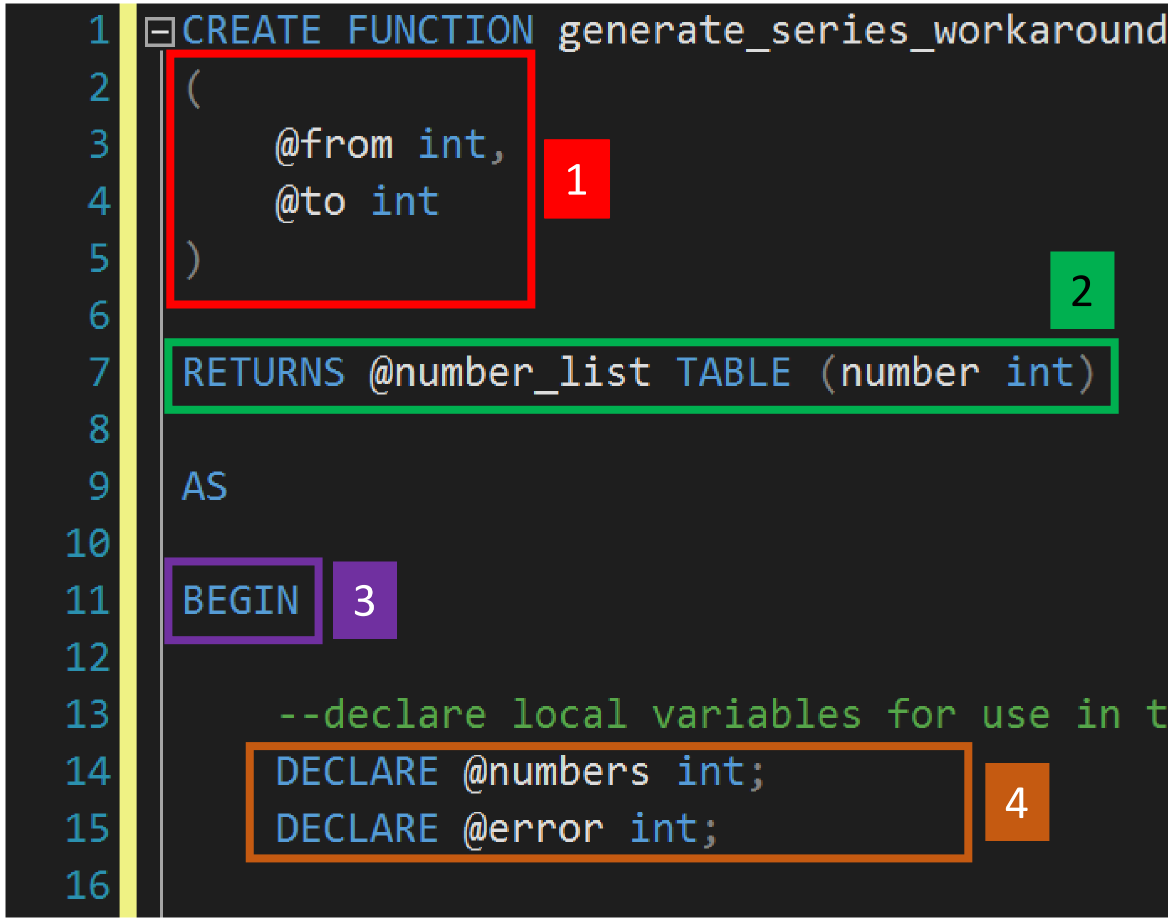
Use an inline TVF for a simple parameterized view



```
CREATE FUNCTION get_data
(
    @from int,
    @to int
)

RETURNS TABLE

RETURN
(
SELECT *
FROM my_schema.my_table
WHERE month_num BETWEEN @from AND @to
)
GO
```

1. Parameters are listed in parentheses and separated by commas
2. Using **RETURNS TABLE** without a variable name indicates this is an in-line TVF
3. We use **RETURN**, then a single SQL statement, optionally within parentheses

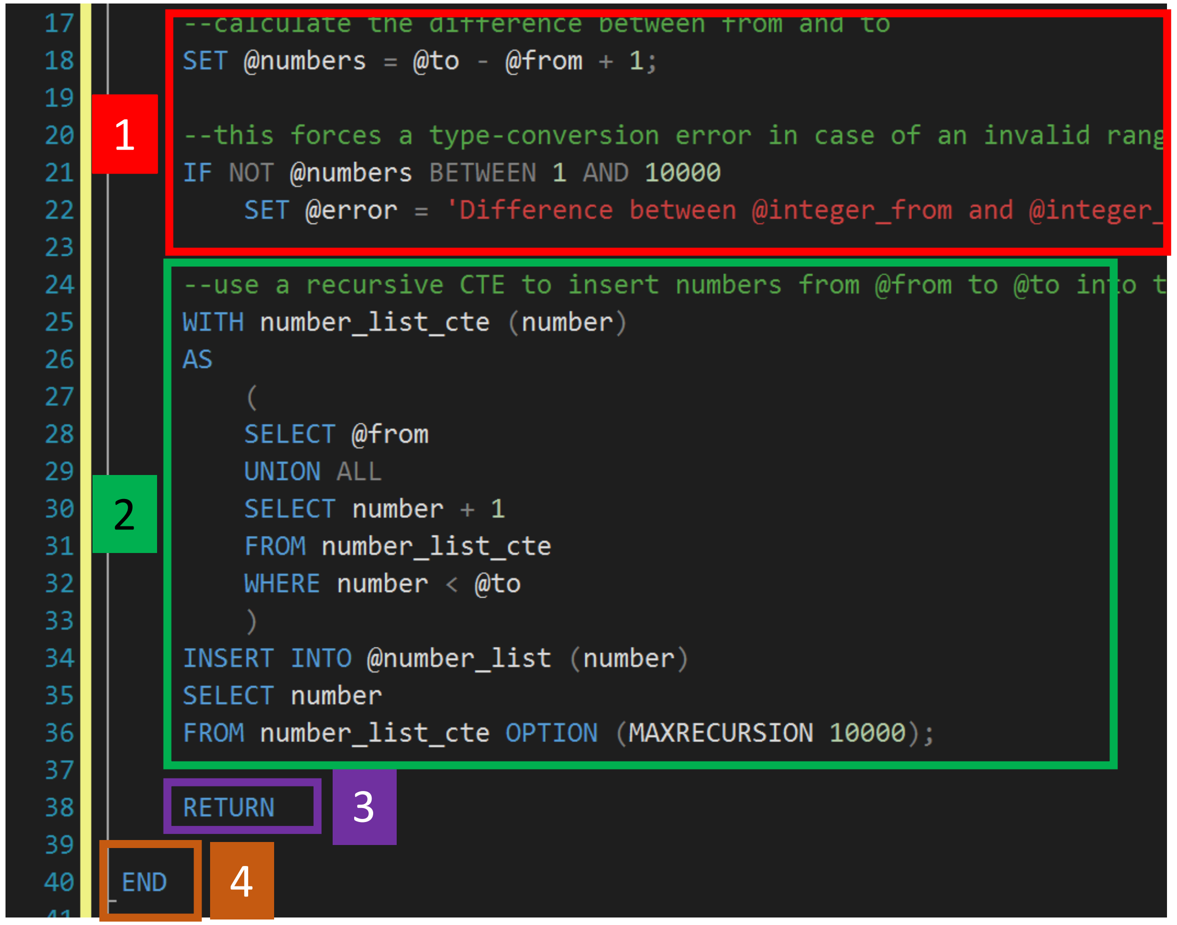
Use a multi-statement TVF for more complex logic (1/2)



```
CREATE FUNCTION generate_series_workaround
(
    @from int,
    @to int
)

RETURNS @number_list TABLE (number int)

AS

BEGIN

    --declare local variables for use in t
    DECLARE @numbers int;
    DECLARE @error int;

```

1. Parameters are listed in parentheses and separated by commas
2. **RETURNS @tablename TABLE (column list)** indicates this is a multi-statement TVF
3. The **BEGIN** keyword is used at the start of the statement block
4. We can **DECLARE** local variables to use in the function

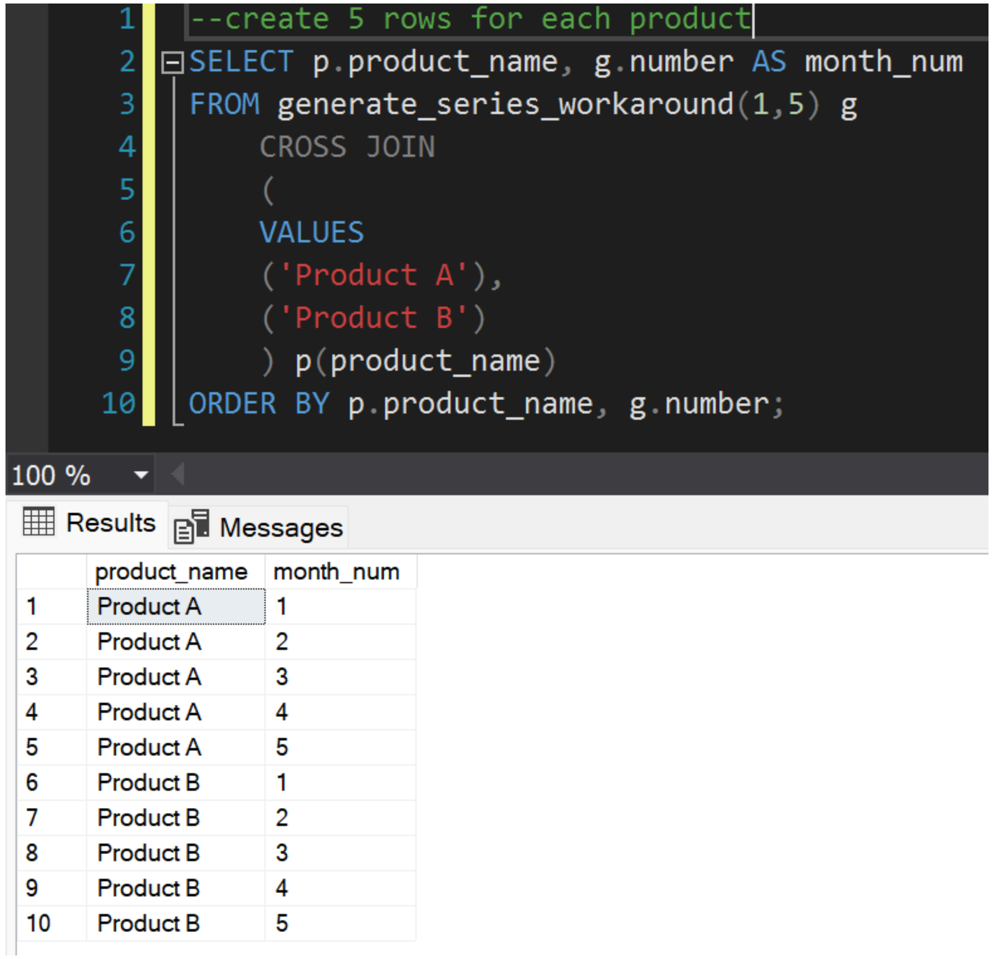
Use a multi-statement TVF for more complex logic (2/2)



```sql
    --calculate the difference between from and to
    SET @numbers = @to - @from + 1;

    --this forces a type-conversion error in case of an invalid rang
    IF NOT @numbers BETWEEN 1 AND 10000
        SET @error = 'Difference between @integer_from and @integer_

    --use a recursive CTE to insert numbers from @from to @to into t
    WITH number_list_cte (number)
    AS
        (
        SELECT @from
        UNION ALL
        SELECT number + 1
        FROM number_list_cte
        WHERE number < @to
        )
    INSERT INTO @number_list (number)
    SELECT number
    FROM number_list_cte OPTION (MAXRECURSION 10000);

    RETURN

END
```

1. Because **TRY/CATCH** is not supported, we may need to find workarounds for invalid values
2. We include some statements to prepare and **INSERT** data to the return TABLE variable
3. The last line in the **BEGIN/END** block must be **RETURN**
4. The function is closed by the **END** keyword

We use a TVF as a table in a FROM clause

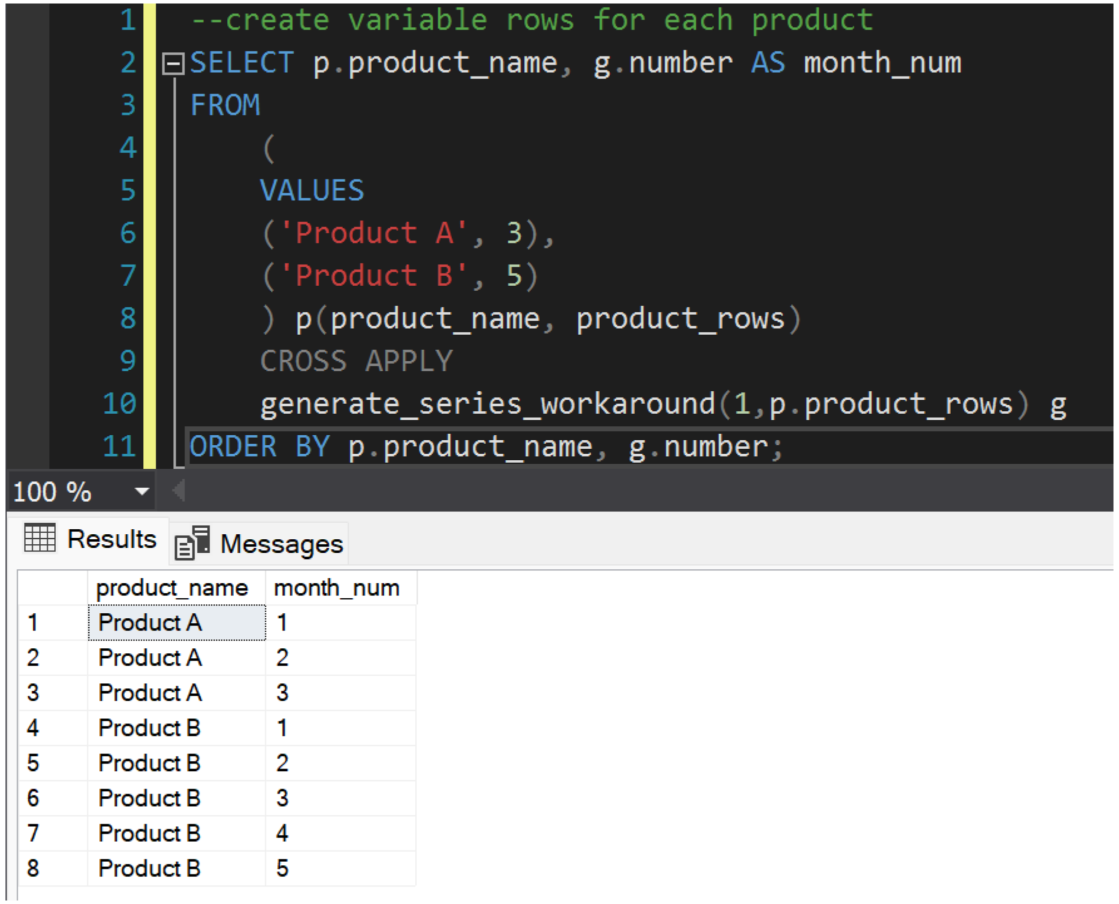


```
--create 5 rows for each product
SELECT p.product_name, g.number AS month_num
FROM generate_series_workaround(1,5) g
    CROSS JOIN
    (
    VALUES
    ('Product A'),
    ('Product B')
    ) p(product_name)
ORDER BY p.product_name, g.number;
```

	product_name	month_num
1	Product A	1
2	Product A	2
3	Product A	3
4	Product A	4
5	Product A	5
6	Product B	1
7	Product B	2
8	Product B	3
9	Product B	4
10	Product B	5

- We can use hard-coded arguments for the function's parameters
- The function call can be aliased just like any other table
- If the TVF returns multiple columns, we can **SELECT** which ones we want to retrieve

Use CROSS APPLY to feed a column into a parameter



```sql
--create variable rows for each product
SELECT p.product_name, g.number AS month_num
FROM
    (
    VALUES
    ('Product A', 3),
    ('Product B', 5)
    ) p(product_name, product_rows)
    CROSS APPLY
    generate_series_workaround(1,p.product_rows) g
ORDER BY p.product_name, g.number;
```

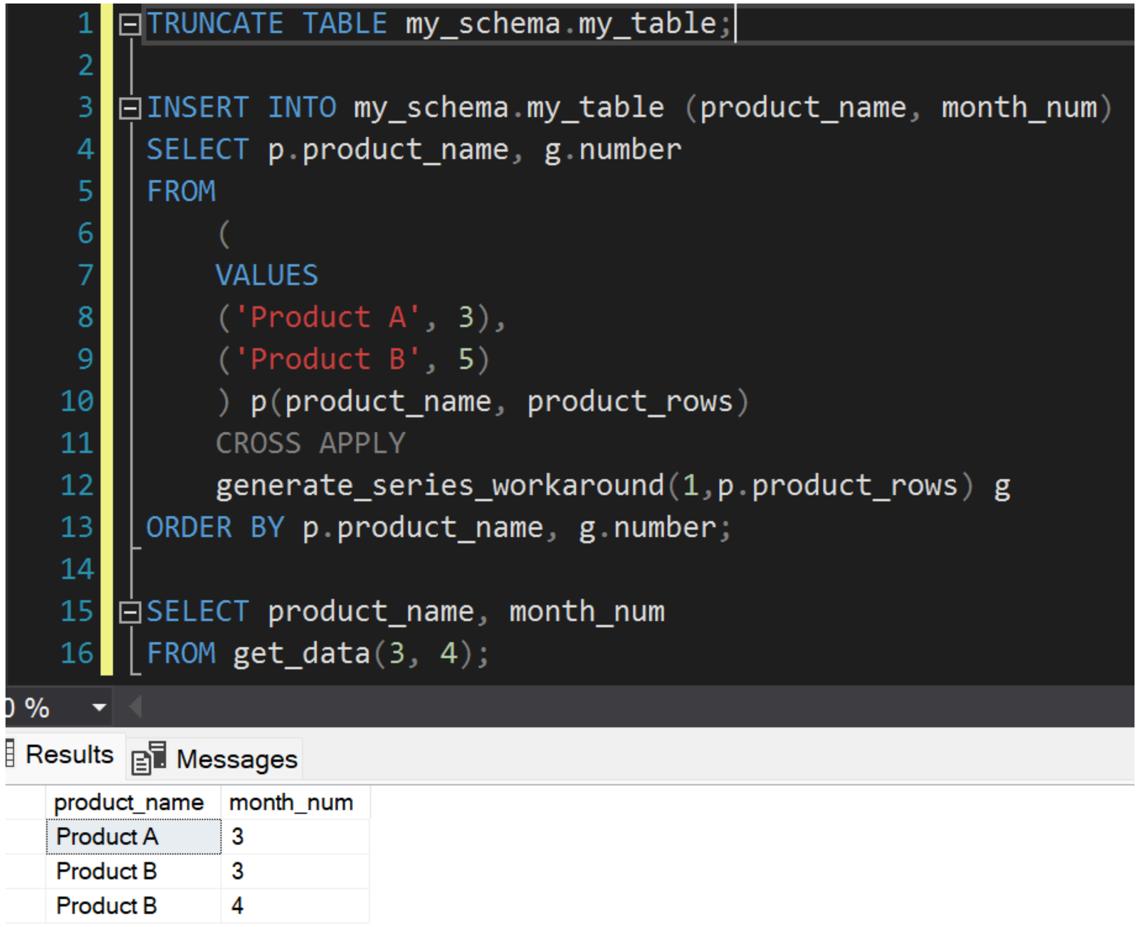
100 %

Results | Messages

	product_name	month_num
1	Product A	1
2	Product A	2
3	Product A	3
4	Product B	1
5	Product B	2
6	Product B	3
7	Product B	4
8	Product B	5

- By using **CROSS APPLY**, we can use a column from another table as an argument to the function
- Here, the **p.product_rows** column is passed to the **@to** parameter of the function

If we always use a WHERE clause to filter a certain table, that’s a candidate for an in-line TVF



```
TRUNCATE TABLE my_schema.my_table;

INSERT INTO my_schema.my_table (product_name, month_num)
SELECT p.product_name, g.number
FROM
    (
    VALUES
    ('Product A', 3),
    ('Product B', 5)
    ) p(product_name, product_rows)
    CROSS APPLY
    generate_series_workaround(1,p.product_rows) g
ORDER BY p.product_name, g.number;

SELECT product_name, month_num
FROM get_data(3, 4);
```

product_name	month_num
Product A	3
Product B	3
Product B	4

- TVFs can of course be used to **INSERT** data
- If we always want to filter **my_schema.my_table** by **month_num**, we can use the **get_data** inline TVF as a parameterized view

Owen Price – flexyourdata.com