



# A Quick Guide to SQL Functions

2017-02-15

# Contents

<b>1</b>	<b>CREATING AND USING SQL FUNCTIONS .....</b>	<b>3</b>
1.1	New SQL Connection	3
1.2	New SQL Query	3
1.3	Output	5
1.4	Input Parameters	5
1.5	Using in XSLT	6

# 1 Creating and Using SQL Functions

Along with Visual, XSLT and C# functions, C1 CMS allows you to create and use SQL functions.

An SQL function is a standard SQL query against a database. A query can return data and an SQL function returns the queried data as XML.

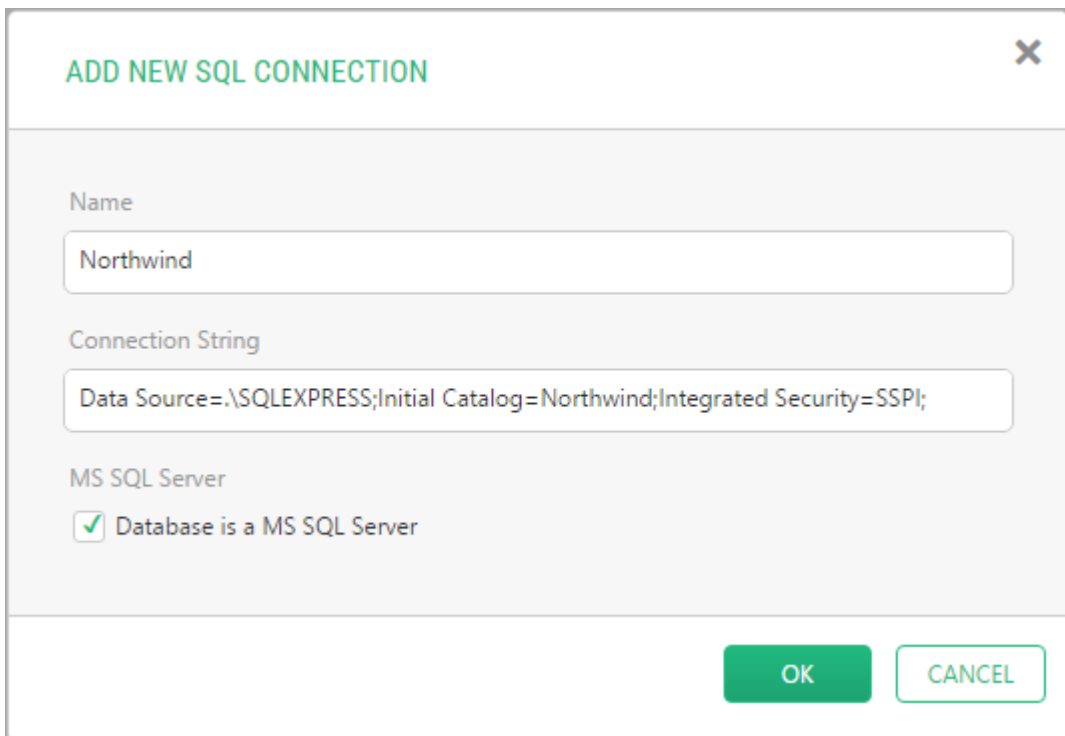
An SQL function behaves as a standard CMS function. You can use parameters on SQL functions and use them as standard SQL parameters in a query. You can call SQL functions from other CMS functions.

Let's create and use an SQL function. For this example, let's use Microsoft's demo database "Northwind" and query its "Products" table for data.

## 1.1 New SQL Connection

Before you create an SQL function, you should create a connection to a database.

1. In the **Functions** perspective, select **SQL Functions** and click **Add SQL Connection** on the toolbar.
2. Specify the **Name** of your connection: e.g. "Northwind".
3. Specify the connection string: e.g. "Data Source=.\SQLEXPRESS;Database=Northwind;Integrated Security=True;".
4. Check the option "**Database is a MS SQL Server**" if needed (which is our case in this example).
5. Click **OK**.



**ADD NEW SQL CONNECTION**

Name  
Northwind

Connection String  
Data Source=.\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=SSPI;

MS SQL Server  
 Database is a MS SQL Server

OK CANCEL

## 1.2 New SQL Query

Once the connection has been created, let's create the function itself:

1. In the **Functions** perspective under **SQL Functions**, select the connection you have just created (e.g. "Northwind") and click **Add New SQL Query** on the toolbar.

2. Specify the **Name** and the **Namespace** for your SQL function: e.g. “GetProducts” and “Demo.Northwind”.
3. In the **SQL command text** field, write the SQL query: e.g. “SELECT \* FROM Products”. (You will be able to edit the query if needed once the function has been created.)
4. Keep other options with default values.
5. Click **OK**.

### ADD NEW SQL QUERY ✕

Name

Namespace

**SQL COMMAND**

SQL command text

**SQL COMMAND BEHAVIOUR**

Is a Stored Procedure

Yes, the command is a procedure

Returns result as XML

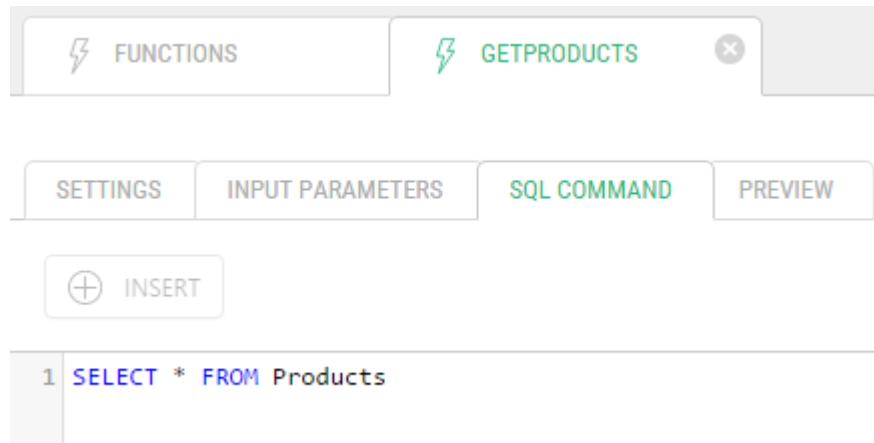
Yes, the command returns XML

Is a query

Yes, the command returns data

The function opens in the function editor.

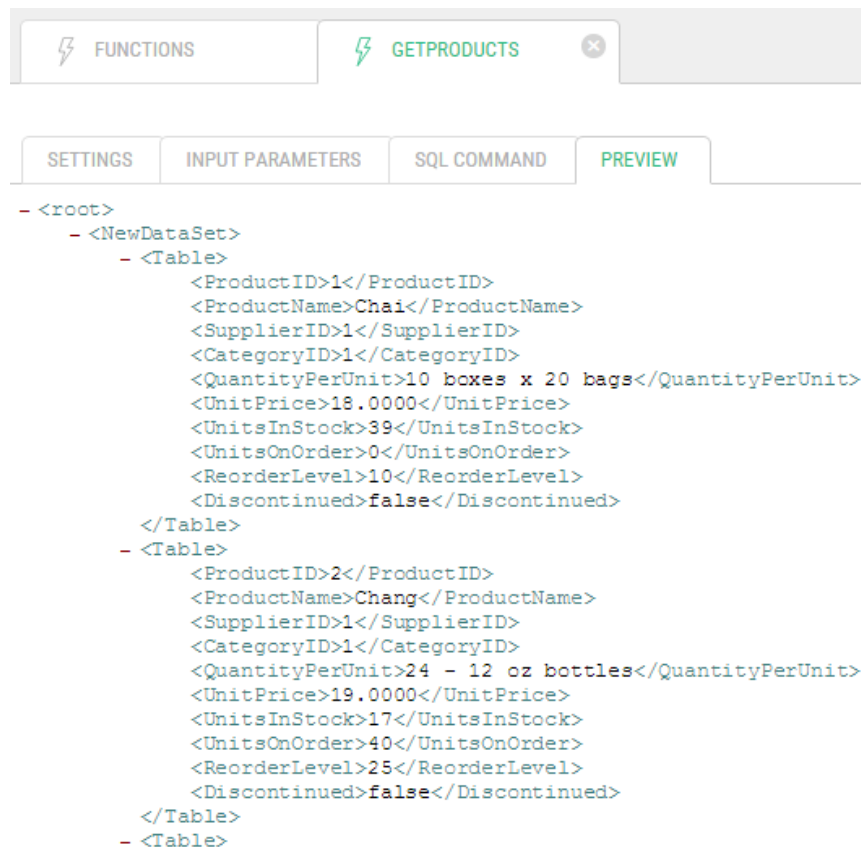
Here, you can edit its settings and its command text, add parameters and preview the output on the corresponding tabs.



### 1.3 Output

To preview what data the function returns, click the Preview tab.

As you can see in the output, the structure of the output XML is 'root/NewDataSet/Table' where each row from the table is specified as the <Table> element with child elements for each column:



### 1.4 Input Parameters

You can add one or more input parameters to the SQL function and use it in the query in a standard way.

For our example, let's add a parameter called "MinUnitsInStock" (the minimum number of products in stock) with default value of "1". We will use this parameter in the query to filter out products that are not in stock.

1. On the **Input Parameters** tab, add a parameter called "MinUnitsInStock".
2. Select "Int32" for its **Parameter type**.
3. Sets its **Default value** to 1.

Now let's use it in our query. On the SQL Command tab, correct the query to read:

```
SELECT * FROM Products WHERE UnitsInStock >= @MinUnitsInStock
```

If you preview the output again, you'll see that products whose UnitsInStock value is greater than 0 are only returned.

## 1.5 Using in XSLT

To use the output of the SQL function, let's create an XSLT function that will output the data in a table:

1. Create an XSLT function called "Demo.Northwind.ProductsInStock".
2. On the **Function Calls** tab, add a call to our function "Demo.Northwind.GetProducts".
3. On the **Template** tab, transform the data from our SQL function to present products in stock in a table. For example:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:in="http://www.composite.net/ns/transformation/input/1.0"
xmlns:lang="http://www.composite.net/ns/localization/1.0"
xmlns:f="http://www.composite.net/ns/function/1.0"
xmlns="http://www.w3.org/1999/xhtml"
exclude-result-prefixes="xsl in lang f">
<xsl:template match="/">
  <html>
    <head></head>
    <body>
      <table>
        <tr><th>Product Name</th><th>Unit Price</th><th>Units in
Stock</th></tr>
        <xsl:apply-templates
select="/in:inputs/in:result[@name='GetProducts']/root/NewDataSet/Table"/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template
match="/in:inputs/in:result[@name='GetProducts']/root/NewDataSet/Table">
    <tr>
      <td><xsl:value-of select="ProductName"/></td>
      <td><xsl:value-of select="UnitPrice"/></td>
      <td><xsl:value-of select="UnitsInStock"/></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Now you can insert the ProductsInStock function on a page and view the results.

Product Name	Unit Price	Units in Stock
Chai	18.0000	39
Chang	19.0000	17
Aniseed Syrup	10.0000	13
Chef Anton's Cajun Seasoning	22.0000	53
Chef Anton's Gumbo Mix	21.3500	0
Grandma's Boysenberry Spread	25.0000	120
Uncle Bob's Organic Dried Pears	30.0000	15
Northwoods Cranberry Sauce	40.0000	6