

# Inhaltsverzeichnis

**How to filter your row data** ..... 1



# How to filter your row data

This document briefly explains how to use the data filter syntax.

## Column names

If a column name contains any of these special characters `~ ( ) # \ / = > < + - * % & | ^ ' „ [ ]`, you must enclose the column name within square brackets `[ ]`. If a column name contains right bracket `]` or backslash `\`, escape it with backslash (`\]` or `\\`).

```
1. "id = 10";           // no special character in column name "id"
2. "$id = 10";          // no special character in column name "$id"
3. "[#id] = 10";        // special character "#" in column name "#id"
4. "[[id\\]] = 10";     // special characters in column name "[id]"
```

## Literals

**String values** are enclosed within single quotes `' '`. If the string contains single quote `'`, the quote must be doubled.

```
1. dataView.RowFilter = "Name = 'John'"           // string value
2. dataView.RowFilter = "Name = 'John 'A'''"       // string with single
quotes "John 'A'"
3. dataView.RowFilter = String.Format("Name = '{0}'", "John
'A'").Replace("'", "'");
```

**Number values** are not enclosed within any characters. The values should be the same as is the result of `int.ToString()` or `float.ToString()` method for invariant or English culture.

```
1. dataView.RowFilter = "Year = 2008"             // integer value
2. dataView.RowFilter = "Price = 1199.9"           // float value
3. dataView.RowFilter =
    String.Format(CultureInfo.InvariantCulture.NumberFormat,
4.                "Price = {0}", 1199.9f);
```

**Date values** are enclosed within sharp characters `# #`. The date format is the same as is the result of `DateTime.ToString()` method for invariant or English culture.

```
1. dataView.RowFilter = "Date = #12/31/2008#"       // date value (time
is 00:00:00)
2. dataView.RowFilter = "Date = #2008-12-31#"       // also this format
is supported
```

```
3. dataView.RowFilter = "Date = #12/31/2008 16:44:58#" // date and time value
4. dataView.RowFilter =
    String.Format(CultureInfo.InvariantCulture.DateTimeFormat,
5.         "Date = #{0}#", new DateTime(2008, 12, 31, 16, 44,
        58));
```

**Alternatively** you can enclose all **values** within single quotes ' '. It means you can use string values for numbers or date time values. In this case the current culture is used to convert the string to the specific value.

```
1. dataView.RowFilter = "Date = '12/31/2008 16:44:58'" // if current culture is English
2. dataView.RowFilter = "Date = '31.12.2008 16:44:58'" // if current culture is German
3. dataView.RowFilter = "Price = '1199.90'" // if current culture is English
4. dataView.RowFilter = "Price = '1199,90'" // if current culture is German
```

## Comparison operators

**Equal, not equal, less, greater** operators are used to include only values that suit to a comparison expression. You can use these operators = <> < <= > >= .

Note: **String comparison is culture-sensitive**, it uses CultureInfo from DataTable.Locale property of related table (dataView.Table.Locale). If the property is not explicitly set, its default value is DataSet.Locale (and its default value is current system culture Thread.CurrentThread.CurrentCulture).

```
1. dataView.RowFilter = "Num = 10" // number is equal to 10
2. dataView.RowFilter = "Date < #1/1/2008#" // date is less than 1/1/2008
3. dataView.RowFilter = "Name <> 'John'" // string is not equal to 'John'
4. dataView.RowFilter = "Name >= 'Jo'" // string comparison
```

**Operator IN** is used to include only values from the list. You can use the operator for all data types, such as numbers or strings.

```
1. dataView.RowFilter = "Id IN (1, 2, 3)" // integer values
2. dataView.RowFilter = "Price IN (1.0, 9.9, 11.5)" // float values
3. dataView.RowFilter = "Name IN ('John', 'Jim', 'Tom')" // string values
```

```
4. dataView.RowFilter = "Date IN (#12/31/2008#, #1/1/2009#)" // date time
   values
5. dataView.RowFilter = "Id NOT IN (1, 2, 3)" // values not from the list
```

**Operator LIKE** is used to include only values that match a pattern with wildcards. **Wildcard** character is \* or %, it can be at the beginning of a pattern '\*value', at the end 'value\*', or at both '\*value\*'. Wildcard in the middle of a pattern 'va\*lue' is not allowed.

```
1. dataView.RowFilter = "Name LIKE 'j*'" // values that start with
   'j'
2. dataView.RowFilter = "Name LIKE '%jo%'" // values that contain 'jo'
3. dataView.RowFilter = "Name NOT LIKE 'j*'" // values that don't start
   with 'j'
```

If a pattern in a LIKE clause contains any of these special characters \* % [ ], those characters must be escaped in brackets [ ] like this [\*], [%], [ ] or [ ].

```
1. dataView.RowFilter = "Name LIKE '[*]*'" // values that starts with
   '*'
2. dataView.RowFilter = "Name LIKE '[[]*'" // values that starts with
   '['
```

The following method escapes a text value for usage in a LIKE clause.

```
1. public static string EscapeLikeValue(string valueWithoutWildcards)
2. {
3.     StringBuilder sb = new StringBuilder();
4.     for (int i = 0; i < valueWithoutWildcards.Length; i++)
5.     {
6.         char c = valueWithoutWildcards[i];
7.         if (c == '*' || c == '%' || c == '[' || c == ']')
8.             sb.Append("[").Append(c).Append("]");
9.         else if (c == '\\')
10.            sb.Append("\\");
11.        else
12.            sb.Append(c);
13.    }
14.    return sb.ToString();
15. }
```

```
1. // select all that starts with the value string (in this case with "*")
2. string value = "*";
3. // the dataView.RowFilter will be: "Name LIKE '[*]*'"
4. dataView.RowFilter = String.Format("Name LIKE '{0}*',
   EscapeLikeValue(value));
```

## Boolean operators

Boolean operators AND, OR and NOT are used to concatenate expressions. Operator NOT has precedence over AND operator and it has precedence over OR operator.

```
1. // operator AND has precedence over OR operator, parenthesis are needed
2. dataView.RowFilter = "City = 'Tokyo' AND (Age < 20 OR Age > 60)";
3. // following examples do the same
4. dataView.RowFilter = "City <> 'Tokyo' AND City <> 'Paris'";
5. dataView.RowFilter = "NOT City = 'Tokyo' AND NOT City = 'Paris'";
6. dataView.RowFilter = "NOT (City = 'Tokyo' OR City = 'Paris')";
7. dataView.RowFilter = "City NOT IN ('Tokyo', 'Paris')";
```

## Arithmetic and string operators

**Arithmetic operators** are addition +, subtraction -, multiplication \*, division / and modulus %.

```
1. dataView.RowFilter = "MotherAge - Age < 20"; // people with young
    mother
2. dataView.RowFilter = "Age % 10 = 0"; // people with decennial
    birthday
```

There is also one **string** operator concatenation +.

## Parent-Child Relation Referencing

A **parent table** can be referenced in an expression using parent column name with *Parent.* prefix. A column in a **child table** can be referenced using child column name with *Child.* prefix.

The reference to the child column must be in an **aggregate function** because child relationships may return multiple rows. For example expression *SUM(Child.Price)* returns sum of all prices in child table related to the row in parent table.

If a table has more than one child relation, the prefix must contain relation name. For example expression *Child(OrdersToItemsRelation).Price* references to column Price in child table using relation named OrdersToItemsRelation.

## Aggregate Functions

There are supported following aggregate functions SUM, COUNT, MIN, MAX, AVG (average), STDEV (statistical standard deviation) and VAR (statistical variance). This example shows aggregate function performed on a single table.

```
1. // select people with above-average salary
2. dataView.RowFilter = "Salary > AVG(Salary)";
```

Following example shows aggregate functions performed on two tables which have parent-child relation. Suppose there are tables Orders and Items with the parent-child relation.

```
1. // select orders which have more than 5 items
2. dataView.RowFilter = "COUNT(Child.IdOrder) > 5";
3. // select orders which total price (sum of items prices) is greater or
   equal $500
4. dataView.RowFilter = "SUM(Child.Price) >= 500";
```

## Functions

There are also supported following functions. Detailed description can be found here [DataColumn.Expression](#).

- **CONVERT** - converts particular expression to a specified .NET Framework type
- **LEN** - gets the length of a string
- **ISNULL** - checks an expression and either returns the checked expression or a replacement value
- **IIF** - gets one of two values depending on the result of a logical expression
- **TRIM** - removes all leading and trailing blank characters like \r, \n, \t, , ' , "
- **SUBSTRING** - gets a sub-string of a specified length, starting at a specified point in the string

1.

From:  
<https://jmz-elektronik.ch/dokuwiki/> - Bücher & Dokumente

Permanent link:  
<https://jmz-elektronik.ch/dokuwiki/doku.php?id=start:visualstudio2019:programmieren:dotnet:component:dataview&rev=1670235921>

Last update: 2022/12/05 11:25

