Calc Guide

Chapter 13
Calc as a Simple Database

A guide for users and macro programmers

Copyright

Contributors

This book is adapted and updated from the *LibreOffice 6.2 Calc Guide*.

To this edition

Steve Fanning
Leo Moons

To previous editions

Andrew Pitonyak
Barbara Duprey
Jean Hollis Weber
Simon Brydon
Kees Kriek
Zachary Parliman

Feedback

Please direct any comments or suggestions about this document to the Documentation Team's mailing list: documentation@global.libreoffice.org.

Warning

Everything you send to a mailing list, including your email address and any other personal information that is written in the message, is publicly archived and cannot be deleted.

Publication date and software version

Published June 2020. Based on LibreOffice 6.4.

Using LibreOffice on macOS

Some keystrokes and menu items are different on macOS from those used in Windows and Linux. The table below gives some common substitutions for the instructions in this book. For a more detailed list, see the application Help and Appendix A (Keyboard Shortcuts) to this guide.

Windows or Linux	*macOS equivalent*	*Effect*
Tools > Options menu selection	**LibreOffice > Preferences**	Access setup options
Right-click	Control+click and/or right-click depending on computer setup	Open a context menu
Ctrl (Control)	⌘ *(Command)*	Used with other keys
Ctrl+Q	⌘*+Q*	Exit / quit LibreOffice
F11	⌘*+T*	Open the Sidebar's Styles deck

Contents

Introduction

Though it is a spreadsheet program, Calc has sufficient functionality to act as a simple yet capable database-like platform. This chapter presents an overview of these capabilities and explains them using LibreOffice Basic macros and GUI (Graphical User Interface) examples.

Note

Though it was created for macro programmers, this guide is meant to be accessible for all users. If you do not want to use macros, simply skip the sections that deal with them. However, if you are interested in learning more about them, see Chapter 12, Macros, in this book, and Andrew Pitonyak's book, *OpenOffice.org Macros Explained* (OOME).

All the macro information in this chapter is drawn or adapted from the OOME and LibreOffice's API reference at https://api.libreoffice.org/docs/idl/ref/index.html.

A database primer

In a typical database, related data is organized into tables, which are arranged in a grid-like series of rows and columns similar to a spreadsheet. Each row of a table represents a data record, while each column represents a field within each record. Each cell in a field contains an individual data item or attribute, such as a name, while each record consists of related attributes that correspond to a single entity, like a person. A database table tends to have a fixed number of fields, but can have an indefinite number of records.

While a table may have hundreds or thousands of rows, individual records can be easily found, retrieved, and updated using information requests, called queries, that search for records that meet a specified set of criteria. It is this ease of access that makes a database table more useful than simply filing away information in an unordered spreadsheet.

To illustrate this concept of a database table, consider the example of a class grading sheet (Figure 1). In this sheet, each row represents individual students taking the class, while each column contains their names and grades. With this table, you can quickly look up individual students' grades simply by searching for their names, and you can determine which students are passing the class by filtering out records with failing average scores.

	A	B	C	D	E	F	G	H
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
2	**Andrew**	90	100	82	90	88	92	90.33
3	**Bethany**	95	100	82	80	88	93	89.67
4	**Charles**	80	93	73	80	75	84	80.83
5	**David**	75	86	91	40	88	79	76.50
6	**Emily**	100	100	81	100	75	94	91.67
7	**Ferdinand**	85	93	73	60	50	72	72.17
8	**Georgia**	70	80	55	39	75	67	64.33
9	**Haley**	85	93	82	70	75	76	80.17
10	**Ian**	100	100	91	90	100	96	96.17
11	**Jennifer**	85	93	73	80	100	90	86.83

Figure 1: Grading sheet example

Note

This simple tabular design is based on the *relational database model,* which is one of the most common and well-known design models used in modern databases.

Calc as a database-like program

As mentioned, a database table is similar to a spreadsheet, and can even be contained within one. Additionally, as a spreadsheet program, Calc offers several features, particularly sorting and filtering, that allow users to search tables similar to how one would in a database program such as LibreOffice Base or Microsoft Access. While this does not make Calc a replacement for either of those programs, it is nevertheless still useful for managing data in a small-scale personal or professional context without having to learn how to use a dedicated database system.

Associating a range with a name

In order to set up a database table in a Calc sheet, you first need to set up an area for it to occupy. This is necessary since some of Calc's database-like features depend on accessing or modifying a table's location. Such an area is represented by a *range*, which is a contiguous group of one or more cells. To make the range for a table easy to access, you can assign a meaningful name to it. Doing this has four particular benefits:

- **Giving a range a name makes it easier to identify,** especially if you are working with multiple ranges in a document**.**
- **A named range can be referenced by its name rather than just by its address.** For example, if you have a range named *Scores*, you can simply reference it in a cell with an equation like `=SUM(Scores)`.
- **References by name to a named range are automatically updated every time the range's address is changed.** This prevents the need to change individual references every time a range's location is modified.
- **All named ranges can be quickly viewed and accessed through the Navigator**, which is opened by selecting **View > Navigator**, pressing the *F5* key, or clicking on the  icon in the Sidebar panel.

Two types of named range exist in Calc: *database ranges*, which store settings for database-like operations, and standard *named ranges*, which do not.

Named ranges

Standard named ranges are created using the Define Names dialog (Figure 2), which is opened by selecting **Sheet > Named Ranges and Expressions > Define** from the Menu bar.

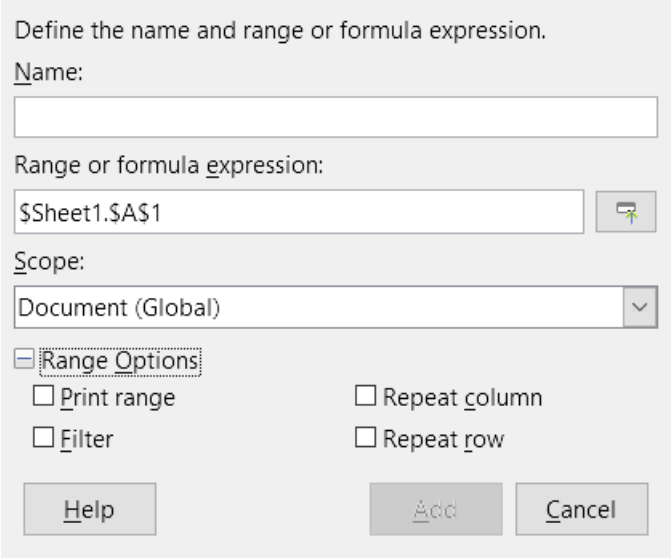

Figure 2: Define Names dialog

To create a named range, select a range of cells from a sheet, then open the dialog. Next, give the range a meaningful name, and click on **Add** to add it to the current document's list of named ranges. You can then access and modify these ranges using the Manage Names dialog, which is opened by selecting **Sheet > Named Ranges and Expressions > Manage** from the Menu bar, or pressing *Ctrl+F3* (Figure 3). For more detail about how to create and manage ranges, see Chapter 6, Printing, Exporting, E-mailing, and Signing, and Chapter 7, Using Formulas and Functions.

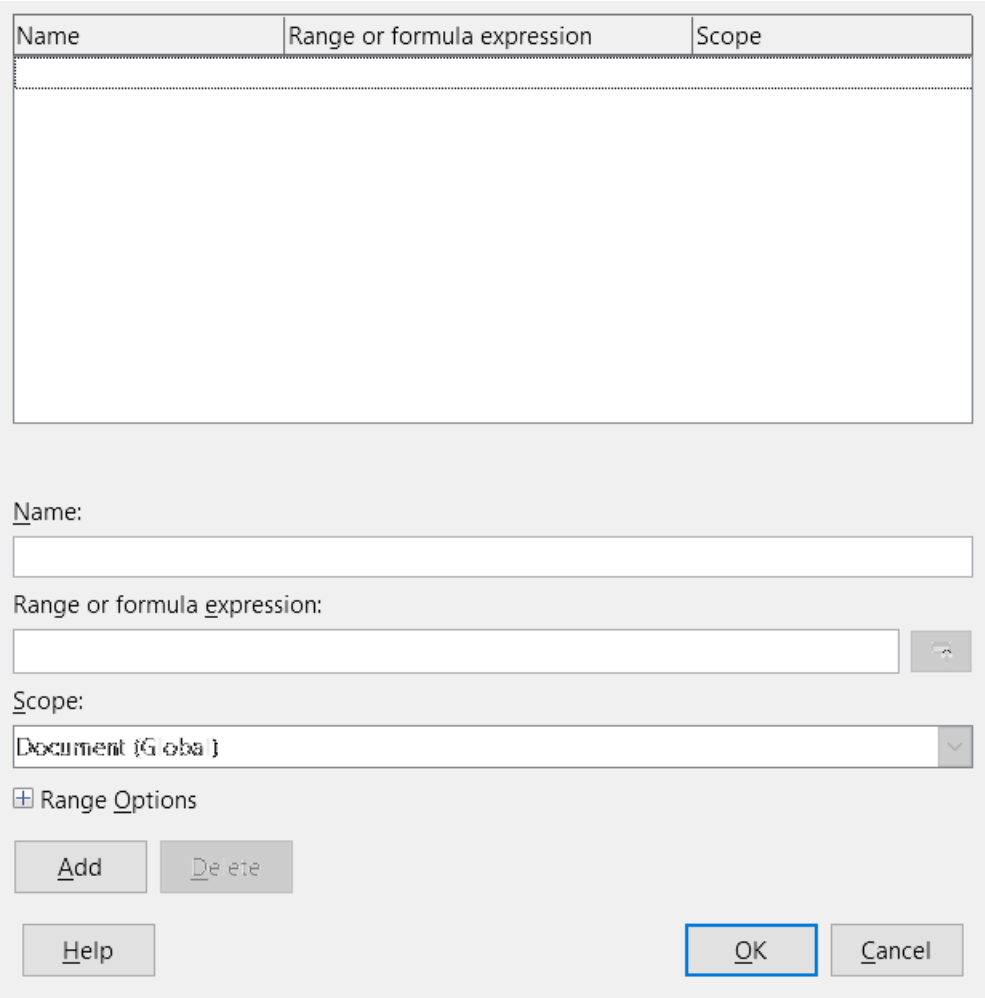

Figure 3: Manage Names dialog

Creating named ranges with macros

In a macro, a named range is accessed, created, and deleted using the *NamedRanges* service of a Calc document. This service has a number of methods associated with it, the following of which are particularly useful for creating named ranges:

getByName(Name)
Returns the range or expression with the specified name.

getElementNames()
Returns an array of all named ranges in the current document.

hasByName(Name)
Returns a boolean: TRUE if a range with this name is in the current document, and FALSE otherwise.

addNewByName(Name, Content, Position, Type)
Adds a new named range to the current document. This method has four arguments:

- *Name* – A string that contains the name of the new range

- *Content* – A string that contains the range address or formula expression being named
- *Position* – The base address for relative cell references
- *Type* – A combination of flags that specify the type of named range being defined. These flags are listed in Table 1. This parameter defaults to zero for any common named range.

Table 1: com.sun.star.sheet.NamedRangeFlag constant group reference

Value	***Name***	***Description***
1	FILTER_CRITERIA	The range contains filter criteria.
2	PRINT_AREA	The range can be used as a print range.
4	COLUMN_HEADER	The range can be used as column headers for printing.
8	ROW_HEADER	The range can be used as row headers for printing.

As an example, the macro in Listing 1 uses the above methods to check if a range with a given name exists. If it does not exist, then the macro creates a range with the name and sets it to access the cell range `B3:D6`.

Listing 1: AddNamedRange creates a new named range that references $Sheet1.$B$3:$D$6

```
Sub AddNamedRange()
  Dim oRange     ' The created range.
  Dim oRanges    ' All named ranges.
  Dim sName$     ' Name of the named range to be created.
  Dim oCell      ' Cell object.
  Dim s$

  sName$ = "MyNRange"
  oRanges = ThisComponent.NamedRanges
  If NOT oRanges.hasByName(sName$) Then
    REM Setting the base address for relative cell references
    Dim oCellAddress As new com.sun.star.table.CellAddress
    oCellAddress.Sheet = 0     'The first sheet.
    oCellAddress.Column = 1    'Column B.
    oCellAddress.Row = 2       'Row 3.

    REM The first argument is the range name.
    REM The second argument is a string that defines the formula
    REM or expression to be used.
    REM The third argument specifies the base address for
    REM relative cell references.
    REM The fourth argument is a set of flags that define
    REM how the range is used, but most ranges use 0.
    REM The fourth argument uses values from the
    REM NamedRangeFlag constants.
    s$ = "$Sheet1.$B$3:$D$6"
    oRanges.addNewByName(sName$, s$, oCellAddress, 0)
  End If
  REM Get the created named range.
  oRange = ThisComponent.NamedRanges.getByName(sName$)

  REM Print the string contained in cell $Sheet1.$B$3
  oCell = oRange.getReferredCells().getCellByPosition(0,0)
  Print oCell.getString()
End Sub
```

Using relative references with named expressions

If a named range uses any cell addresses that are not absolute, then these addresses will be referenced relative to the range's base address, which is defined by the third argument of the *addNewByName* method, *Position*. This behavior is illustrated in Listing 2, where the macro *AddNamedFunction* creates the named expression *AddLeft*. This expression references the equation A3+B3 with C3 as its base address. Because relative references are being used, *AddLeft* sums the values of the two cells directly to the left of any cell containing the equation =AddLeft. For example, if *AddLeft* is referenced in cell E5, then it will sum the values in C5 and D5 (Figure 4).

Note

For more information about absolute and relative references, see Chapter 7, Using Formulas and Functions.

Listing 2: AddNamedFunction creates the AddLeft named formula expression

```
Sub AddNamedFunction()
  Dim oSheet           'Sheet that contains the range oRange.
  Dim oCellAddress     'Address for relative references.
  Dim oRanges          'The NamedRanges property.
  Dim oRange           'Single cell range.
  Dim sName As String 'Name of the equation to create.

  sName = "AddLeft"
  oRanges = ThisComponent.NamedRanges
  If NOT oRanges.hasByName(sName) Then
    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("C3")
    oCellAddress = oRange.getCellAddress()
    oRanges.addNewByName(sName, "A3+B3", oCellAddress, 0)
  End If
End Sub
```

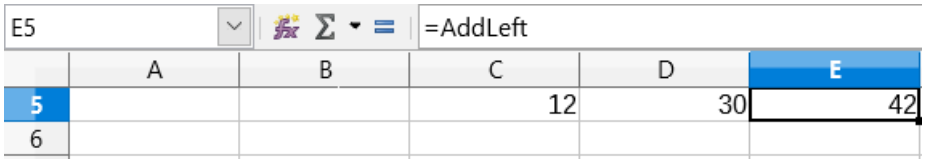

Figure 4: AddLeft sums the values from C5 and D5 in E5

Tip

Listing 2 illustrates another little-known attribute of Calc: named ranges are a subset of named expressions, which can include formulas as well.

Creating named ranges using row or column headers

With the *Create Names* tool, which is accessed by selecting **Sheet > Named Ranges and Expressions > Create** from the Menu bar (Figure 5), you can create multiple named ranges simultaneously from the headers of a table. These headers can be drawn from the table's borders – top and bottom rows and left and right columns – and each row or column that corresponds to each header are used to create the named ranges themselves. For example, if you choose to create ranges from headers contained in the top row of a table, each range will be generated from the individual columns that correspond to each header label.

Note

Header cells are not included in the named ranges generated using the *Create Names* tool. This is because the labels in each of these cell are used to name the ranges.

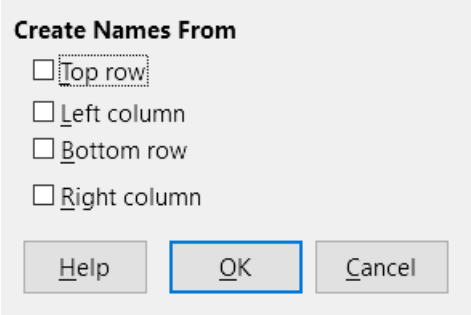

Figure 5: Create Names dialog

To use the *Create Names* tool:

1) In a sheet, select the table from which to create the named ranges. Be sure to include the header rows or columns as part of your selection.
2) Open the Create Names From dialog by selecting **Sheet > Named Ranges and Expressions > Create** from the Menu bar.
3) Calc automatically identifies which rows or columns contain headers, and will mark the checkboxes – **Top row**, **Left column**, **Bottom row**, **Right column** – that apply. However, if you wish to change this selection, you can manually select any of the boxes at this point.
4) Click on **OK** to close the dialog and create the new named ranges.

Creating named ranges from labels using macros

In addition to the Create Names From dialog, named ranges can be generated from labels using the macro method *addNewFromTitles*:

addNewFromTitles(Source, Border)

Creates named ranges from column or row headers. This method has two arguments:

- *Source* – The cell range address of the named range to be created
- *Border* – Is an enumeration value that specifies the location of the header labels. This enumeration has one of four possible values (Table 2):

Table 2: com.sun.star.sheet.Border enumeration values

Enumerator	*Description*
TOP	Selects the top border row.
BOTTOM	Selects the bottom border row.
RIGHT	Selects the right border column.
LEFT	Selects the left border column.

Tip

To generate names from multiple borders, you must call *addNewFromTitles* for each header row or column that you wish to use.

The macro in Listing 3 creates three named ranges using headers from the top row of the range `A1:C20` (Figure 6). Figure 7 shows the resulting ranges listed in the Manage Names dialog, which is accessed by selecting **Sheet > Named Ranges and Expressions > Manage**.

	A	B	C
1	**X**	**X^2**	**X^3**
2	1	1	1
3	2	4	8
4	3	9	27
5	4	16	64
6	5	25	125
7	6	36	216
8	7	49	343
9	8	64	512
10	9	81	729
11	10	100	1000
12	11	121	1331
13	12	144	1728
14	13	169	2197
15	14	196	2744
16	15	225	3375
17	16	256	4096
18	17	289	4913
19	18	324	5832
20	19	361	6859

Figure 6: Example range A1:C20

Listing 3: AddManyNamedRanges creates named ranges using labeled columns

```
Sub AddManyNamedRanges()
  Dim oSheet    'Sheet that contains the named range.
  Dim oAddress  'Range address.
  Dim oRanges   'The NamedRanges property.
  Dim oRange    'Single cell range.

  oRanges = ThisComponent.NamedRanges
  oSheet = ThisComponent.getSheets().getByIndex(0)

  oRange = oSheet.getCellRangeByName("A1:C20")
  oAddress = oRange.getRangeAddress()
  oRanges.addNewFromTitles(oAddress, com.sun.star.sheet.Border.TOP)
End Sub
```

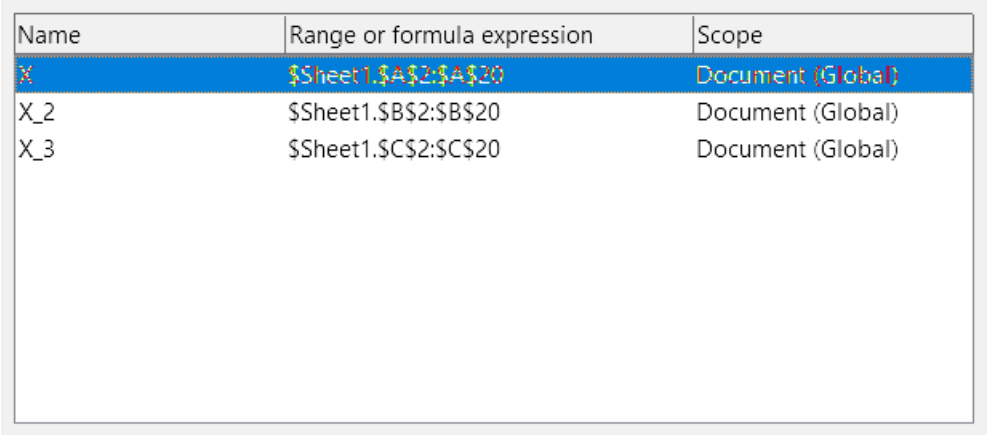

Figure 7: Manage Names dialog with generated named ranges

Caution

Avoid giving multiple rows or columns the same label, as the ranges generated from them will likewise share the same name, and can end up being overwritten by Calc.

Database ranges

Although it can be used like a regular named range, a database range is, unsurprisingly, meant to be used like a database table, with each row representing a record and each cell as fields within each record. Specifically, a database range differs from a named range in the following ways:

- A database range cannot be a formula expression, only a cell range. This range can be formatted as a table, with the first row reserved for headings and the last row for subtotals. Cell formatting can also be preserved for each field in the table.
- Database ranges cannot be referenced relative to a base address within a sheet.
- Database ranges store sorting, filtering, subtotaling, and data import settings in data structures called *descriptors*, which can be retrieved and accessed using macros.
- A database range can be linked to an external database source, and can be refreshed by selecting **Data > Refresh Range** from the Menu bar. Registering and linking to external database sources are explained in more detail in Chapter 10, Linking Data.
- Database ranges can be created, modified, and deleted using the Define Database Range dialog, which is opened by selecting **Data > Define Range** from the Menu bar (Figure 8).

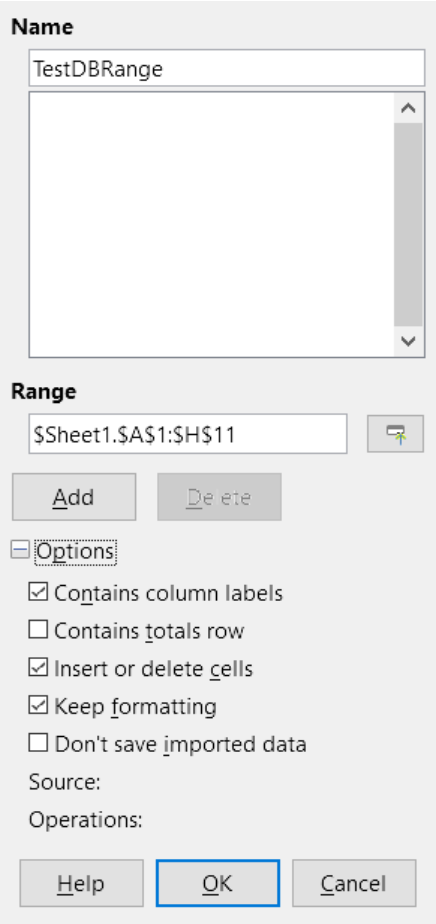

Figure 8: Define Database Range dialog

To create a database range:

1) Select a range of cells from a sheet.
2) Open the Define Database Range dialog by using **Data > Define Range**.
3) Type a name for the range in the *Name* field. Only use letters, numbers, and underscores; spaces, hyphens, and other characters are not allowed.
4) Click on the plus (+) sign next to the *Options* label to expand this section and view and select the following options:
   - *Contains column labels* – Denotes whether the top row is reserved for field headings.
   - *Contains totals row* – Denotes whether the bottom row is reserved for totals.
   - *Insert or delete cells* – If active, this option will insert new rows and columns into the database range when new records are added to its source. Only works if an external database source is linked to the range. To manually update the database range, use **Data > Refresh Range**.
   - *Keep formatting* – Applies the existing cell formats of the first data row to the whole database range.
   - *Don't save imported data* – If selected, this option only saves a reference to the source database; the contents of the range's cells are not preserved.
   - *Source* – Displays information about the current database source, if one exists.
   - *Operations* – Denotes what operations, such as sorting or filtering, have been applied to the database range.
5) Click **Add** to add a range to the database range list under the *Name* field.

To modify an existing database range:

1) Select a range from the range list under the *Name* field or type its name into the *Name* field. The **Add** button will change to **Modify** at this point.
2) Make any modifications in the *Range* field and the *Options* section.
3) Click **Modify** to update the range.

To delete an existing database range, select it from the range list, then click **Delete**.

To select an existing database range from the current document, open the Select Database Range dialog by choosing **Data > Select Range** from the Menu bar (Figure 9). Next, select a range from the *Ranges* list and click **OK**. Calc will automatically highlight the range's position in the sheet in which it is located.

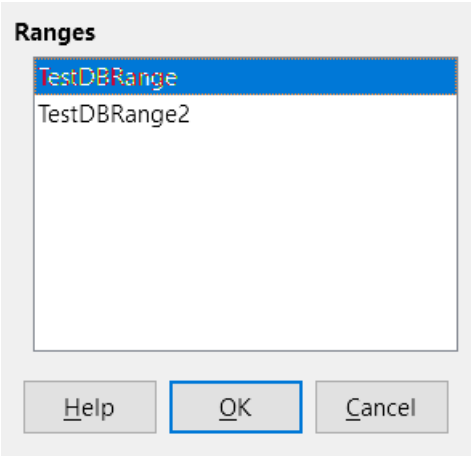

Figure 9: Select Database Range dialog

Creating database ranges with macros

In a macro, a database range is created, accessed, and deleted using the *DatabaseRanges* service. This service has many of the same methods as the *NamedRanges* service, but lacks the *addNewFromTitles* method. *DatabaseRanges* also uses a reduced version of the *addNewByName* method that lacks arguments for a relative base address and range type:

addNewByName(Name, Range)

Adds a new database range to the current document.

As an example of creating a range using this method, the macro in Listing 4 creates a database range named *MyName* and automatically applies auto filters to each of the range's columns:

Listing 4: AddNewDatabaseRange creates a database range and applies an auto filter

```
Sub AddNewDatabaseRange()
  Dim oRange 'DatabaseRange object.
  Dim oAddr  'Cell address range for the database range.
  Dim oSheet 'First sheet, which will contain the range.
  Dim oDoc   'Reference ThisComponent with a shorter name.

  oDoc = ThisComponent
  If NOT oDoc.DatabaseRanges.hasByName("MyName") Then
    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("A1:F10")
    oAddr = oRange.getRangeAddress()
    oDoc.DatabaseRanges.addNewByName("MyName", oAddr)
  End If
  oRange = oDoc.DatabaseRanges.getByName("MyName")
  oRange.AutoFilter = True
End Sub
```

Sorting

Sorting is the process of rearranging data in a range or a sheet according to a specified sort order. In Calc, sorting is commonly done using the Sort dialog, which is accessed by selecting **Data > Sort** from the Menu bar. How to use this dialog and its options is described in further detail in Chapter 2, Entering, Editing, and Formatting Data. Here, we present how to sort data in the context of macros.

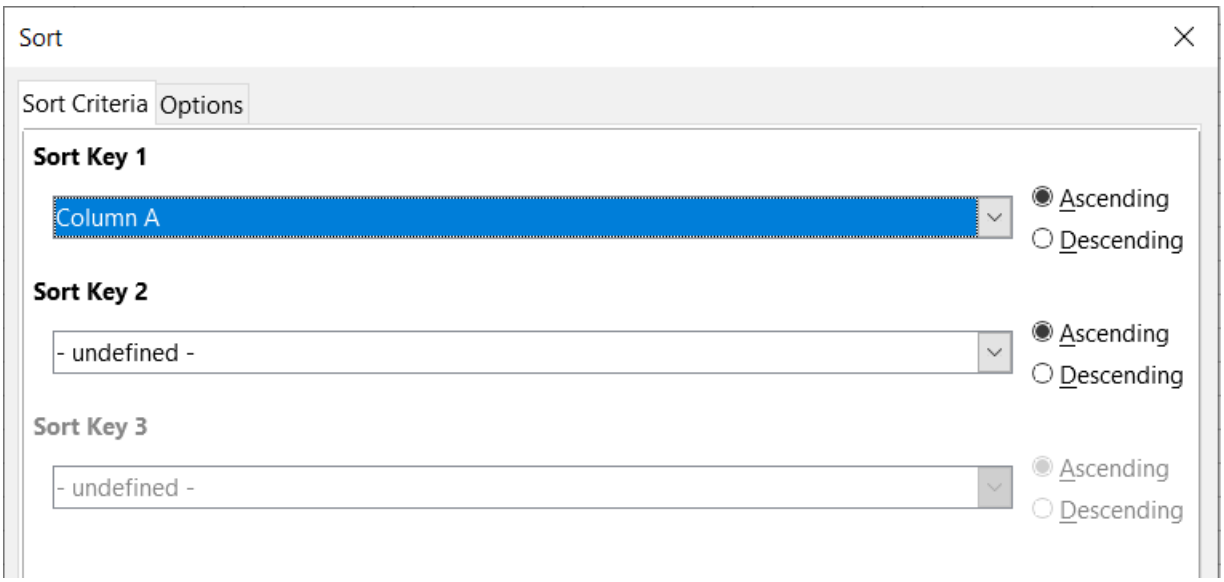

Figure 10: Sort dialog

Sorting a table using one column with a macro

In a Calc macro, data within a range is sorted by calling the *sort()* method on the range object. When a sort operation is called on a range, an array of properties known as a *sort descriptor,* is passed to the *sort* routine. Contained within a descriptor's properties are *sort fields*, which are structures that inform Calc how to sort a range according to the data contained within one of its rows or columns.

As an example, the macro in Listing 5 sorts the grade sheet from Figure 1 according to average grade in descending order. The results are shown in Figure 11.

Listing 5: SortAverageGrade sorts the grade sheet data range (A1:H11) using a single column

```
Sub SortAverageGrade
  Dim oSheet
  Dim oRange
  Dim oSortFields(0) as new com.sun.star.util.SortField
  Dim oSortDesc(0) as new com.sun.star.beans.PropertyValue

  oSheet = ThisComponent.Sheets(0)
  REM Set the range on which to sort
  oRange = oSheet.getCellRangeByName("A1:H11")

  REM Sort by the Average grade field in the range in descending order
  oSortFields(0).Field = 7
  oSortFields(0).SortAscending = FALSE

  REM Set the sort fields to use
  oSortDesc(0).Name = "SortFields"
  oSortDesc(0).Value = oSortFields()

  REM Now sort the range!
  oRange.Sort(oSortDesc())
End Sub
```

	A	B	C	D	E	F	G	H
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
2	**Ian**	100	100	91	90	100	96	96.17
3	**Emily**	100	100	81	100	75	94	91.67
4	**Andrew**	90	100	82	90	88	92	90.33
5	**Bethany**	95	100	82	80	88	93	89.67
6	**Jennifer**	85	93	73	80	100	90	86.83
7	**Charles**	80	93	73	80	75	84	80.83
8	**Haley**	85	93	82	70	75	76	80.17
9	**David**	75	86	91	40	88	79	76.50
10	**Ferdinand**	85	93	73	60	50	72	72.17
11	**Georgia**	70	80	55	39	75	67	64.33
12								

Figure 11: Grading sheet after sorting by average grade in descending order

Sorting a table using multiple columns

As with the Sort dialog, a range can be sorted using up to three columns or rows in a macro. Sorting with extra columns or rows is as easy as adding extra sort fields to a sort descriptor. The macro in Listing 6 again uses the grade sheet example from Figure 1 to illustrate how to sort by two columns. Figure 12 shows the results of this operation – note that records are sorted first by Quiz #1 scores, then Quiz #2 scores.

Listing 6: SortByQuizScores sorts the grade sheet data range (A1:H11) using two columns

```
Sub SortByQuizScores
  Dim oSheet
  Dim oRange
  Dim oSortFields(1) as new com.sun.star.util.SortField
  Dim oSortDesc(0) as new com.sun.star.beans.PropertyValue

  oSheet = ThisComponent.Sheets(0)

  REM Set the range on which to sort
  oRange = oSheet.getCellRangeByName("A1:H11")

  REM Sort by the Quiz #1 field in the range
  oSortFields(0).Field = 4
  oSortFields(0).SortAscending = True
  oSortFields(0).FieldType = com.sun.star.util.SortFieldType.NUMERIC

  REM Sort by the Quiz #2 field in the range
  oSortFields(1).Field = 5
  oSortFields(1).SortAscending = True
  oSortFields(1).FieldType = com.sun.star.util.SortFieldType.ALPHANUMERIC

  REM Set the sort fields to use
  oSortDesc(0).Name = "SortFields"
  oSortDesc(0).Value = oSortFields()

  REM Now sort the range!
  oRange.Sort(oSortDesc())
End Sub
```

	A	B	C	D	E	F	G	H
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
2	**Georgia**	70	80	55	39	75	67	64.33
3	**David**	75	86	91	40	88	79	76.50
4	**Ferdinand**	85	93	73	60	50	72	72.17
5	**Haley**	85	93	82	70	75	76	80.17
6	**Charles**	80	93	73	80	75	84	80.83
7	**Bethany**	95	100	82	80	88	93	89.67
8	**Jennifer**	85	93	73	80	100	90	86.83
9	**Andrew**	90	100	82	90	88	92	90.33
10	**Ian**	100	100	91	90	100	96	96.17
11	**Emily**	100	100	81	100	75	94	91.67
12								

Figure 12: Grading sheet sorted by quiz scores in ascending order

Retrieving sorting information from a range

You can use the method *createSortDescriptor()* to retrieve the sorting information for a given cell range. If this method is called on a database range, it will create a sort descriptor using the sorting information stored with that range. On the other hand, if *createSortDescriptor* is called on a standard named range or an unnamed range, it will generate a sort descriptor with default properties. In either case, the newly-generated sort descriptor can be modified and passed as an argument to a *sort* routine called on a given range.

The macro in Listing 7 demonstrates how to generate and display the sorting information associated with a range. The output of this macro is displayed in Figure 13.

Listing 7: DisplaySortDescriptor displays sort descriptor properties in a dialog

```
Sub DisplaySortDescriptor
  On Error Resume Next
  Dim oSheet
  Dim oRange       ' A range is needed to create the sort descriptor.
  Dim oSortDescript
  Dim i%
  Dim s$
  Dim oDoc          'Reference newly created calc document.

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", "_default",
0, Array())
  oSheet = oDoc.Sheets(0)
  oRange = oSheet.getCellRangeByName("B28:D33")
  oSortDescript = oRange.createSortDescriptor()
  For i = LBound(oSortDescript) To UBound(oSortDescript)
    s = s & oSortDescript(i).Name & " = "
    s = s & oSortDescript(i).Value
    s = s & CHR$(10)
  Next
  MsgBox s, 0, "Sort Descriptor"
End Sub
```

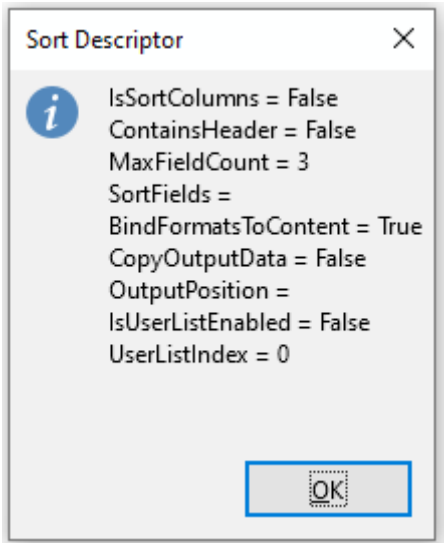

Figure 13: Sort descriptor info

Filtering

A *filter* is a tool that hides or displays records within a sheet based on a set of filtering criteria. Similar to sorting, filters are useful for narrowing down long lists of data in order to find particular data items. In Calc, three types of filter exist:

- AutoFilters
- Standard filters
- Advanced filters.

Filters are also described in Chapter 2, Entering, Editing, and Formatting Data.

AutoFilter

AutoFilters are the most straightforward of the three filter types, and work by inserting a combo box into one or more data columns (Figure 14). To add an AutoFilter to one or more columns, simply

select the columns, then select **Data > AutoFilter** from the Menu bar. To access the AutoFilter combo box for a column, click on the down-arrow button in the first cell of that column.

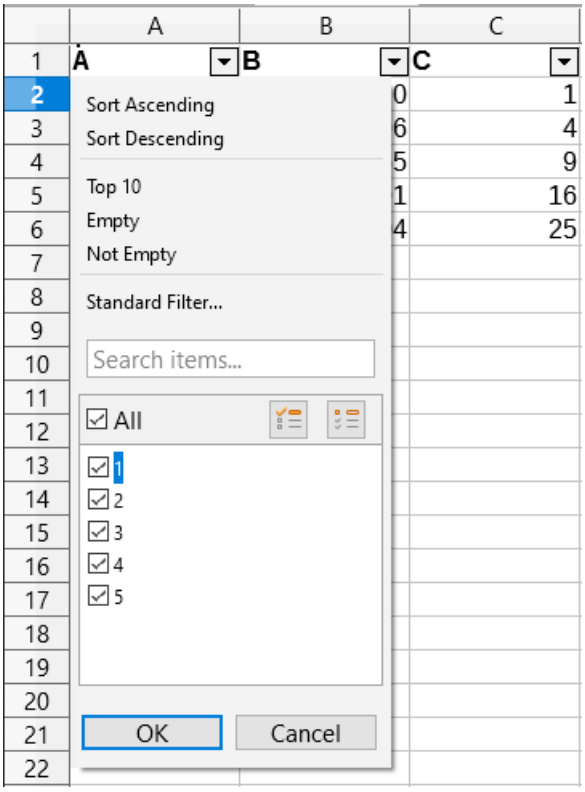

Figure 14: Auto filter combo box

To remove an AutoFilter, select the columns again and click on **Data > AutoFilter**. Each combo box and down-arrow button will disappear. In this way, the menu option acts like a toggle for AutoFilters.

Each AutoFilter combo box has the following set up of criteria:

- A basic sort can be applied using the **Sort Ascending** or **Sort Descending** options.
- The **Standard Filter** option opens the Standard Filter dialog (Figure 15) and automatically sets the current field as the field for the first condition in the dialog.
- Selecting **Empty** hides all non-empty rows that contain a value in the current column. Likewise, selecting **Not Empty** hides all non-empty rows that lack a value in the current column. Entirely empty rows are ignored.
- Selecting the **Top 10** filter causes the ten rows with the largest value to be displayed. More than ten rows may be displayed if there are more than ten instances of the largest value in a column. For example, if there are eleven students with a perfect score of 100, then the filter will display all eleven instances.
- Check the **All** box to display or hide all values in the current column.
- The auto filter creates entries for each unique value in the current column. These values can be filtered simply by checking off any of the check boxes next to each entry.

Toggling AutoFilters with a macro

Database ranges in Calc contain an `AutoFilter` boolean flag that allows you to toggle auto filters on or off. The macro from Listing 4 demonstrates how to do this.

Standard filters

Standard filters are more complex than AutoFilters, and allow for up to eight filter conditions. Also, unlike AutoFilters, standard filters use a dialog (Figure 15), which is accessed by selecting **Data > More Filters > Standard Filter** from the Menu bar.

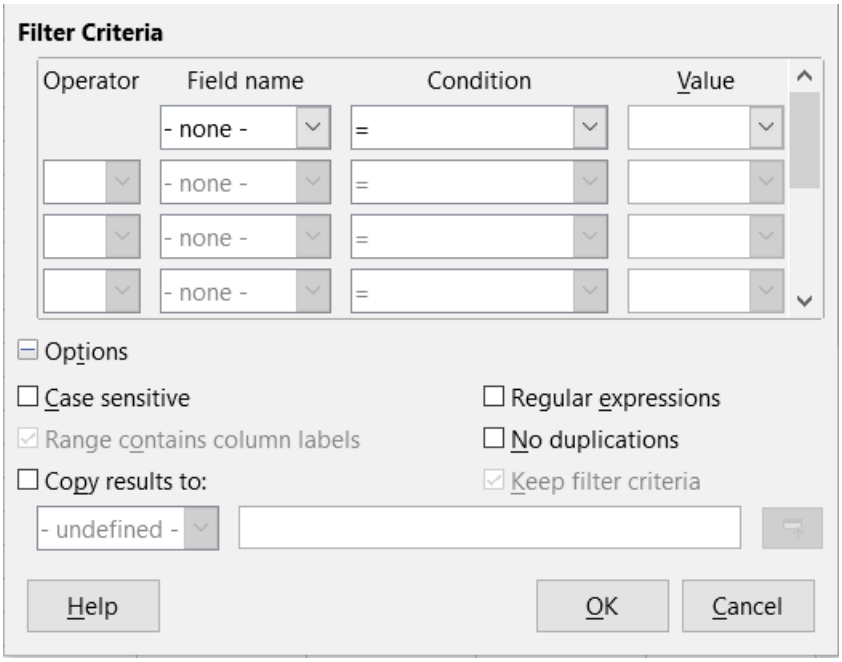

Figure 15: Standard Filter dialog

For more information on how to use this dialog and its options, see Chapter 2, Entering, Editing, and Formatting Data.

Creating standard filters with macros

In a macro, filtering a data range is performed by calling the *filter()* routine either on the range or the sheet in which the range is contained. Like a sort descriptor, a *filter descriptor* contains the filter settings for the current sheet, such as whether the first row or column in the sheet contains headers which should not be filtered. A filter descriptor can be generated by calling the *createFilterDescriptor()* method on a sheet or on a cell range:

createFilterDescriptor(Empty)

Creates a filter descriptor. If the boolean flag *Empty* is set to TRUE, then an empty filter descriptor is created. If *Empty* is FALSE, then the descriptor is filled with the previous settings of the current object (such as a database range).

As with sort descriptors, filter descriptors can be created using this method, then modified and passed as an argument to the *filter* method. The macro in Listing 8 demonstrates this process by creating a simple standard filter for the first sheet in a document. Figure 16 display the results of filtering the grading sheet example in Figure 1:

Listing 8: SimpleSheetFilter creates a simple standard filter

```
Sub SimpleSheetFilter()
  Dim oSheet       ' Sheet that will contain the filter.
  Dim oFilterDesc  ' Filter descriptor.
  Dim oFields(0) As New com.sun.star.sheet.TableFilterField

  oSheet = ThisComponent.getSheets().getByIndex(0)

  REM If argument is True, creates an empty filter
  REM descriptor. If argument is False, create a
```

```
  REM descriptor with the previous settings.
  oFilterDesc = oSheet.createFilterDescriptor(True)

  With oFields(0)
    REM You could use the Connection property to indicate
    REM how to connect to the previous field. This is
    REM the first field, so this is not required.
    '.Connection = com.sun.star.sheet.FilterConnection.AND
    '.Connection = com.sun.star.sheet.FilterConnection.OR

    REM The Field property is the zero based column
    REM number to filter. If you have the cell, you
    REM can use .Field = oCell.CellAddress.Column.
    .Field = 5          ' The Quiz #2 grades field

    REM Compare using a numeric or a string?
    .IsNumeric = True

    REM The NumericValue property is used
    REM because .IsNumeric = True from above.
    .NumericValue = 80

    REM If IsNumeric was False, then the
    REM StringValue property would be used.
    REM .StringValue = "what ever"

    REM Valid operators include EMPTY, NOT_EMPTY, EQUAL,
    REM NOT_EQUAL, GREATER, GREATER_EQUAL, LESS,
    REM LESS_EQUAL, TOP_VALUES, TOP_PERCENT,
    REM BOTTOM_VALUES, and BOTTOM_PERCENT
    .Operator = com.sun.star.sheet.FilterOperator.GREATER_EQUAL
  End With

  REM The filter descriptor supports the following
  REM properties: IsCaseSensitive, SkipDuplicates,
  REM UseRegularExpressions,
  REM SaveOutputPosition, Orientation, ContainsHeader,
  REM CopyOutputData, OutputPosition, and MaxFieldCount.
  oFilterDesc.setFilterFields(oFields())
  oFilterDesc.ContainsHeader = True
  oSheet.filter(oFilterDesc)
End Sub
```

	A	B	C	D	E	F	G	H
1	Student	HW #1	HW #2	HW #3	Quiz #1	Quiz #2	Test #1	Average
2	Andrew	90	100	82	90	88	92	90.33
3	Bethany	95	100	82	80	88	93	89.67
5	David	75	86	91	40	88	79	76.50
10	Ian	100	100	91	90	100	96	96.17
11	Jennifer	85	93	73	80	100	90	86.83

Figure 16: Grading sheet filtered by Quiz #2 values greater than 80

Note

When the *filter* method is called on a sheet, every empty row in the sheet is hidden. When *filter* is called on a range, only empty rows in the range itself are hidden.

Clearing all filters for a sheet

When a filter is applied to a sheet, it replaces any existing filter for that sheet. Therefore, to remove a filter in a sheet, simply create and set an empty filter for that sheet (Listing 9).

Listing 9: RemoveSheetFilter removes the current sheet filter by setting an empty filtered

```
Sub RemoveSheetFilter()
  Dim oSheet          ' Sheet to filter.
  Dim oFilterDesc     ' Filter descriptor.

  oSheet = ThisComponent.getSheets().getByIndex(0)
  oFilterDesc = oSheet.createFilterDescriptor(True)
  oSheet.filter(oFilterDesc)
End Sub
```

Filtering multiple columns and filtering with regular expressions

The macro in Listing 10 demonstrates a filter that filters two columns and uses regular expressions. Note that the *filter* method is called on a range rather than its sheet in this example. Figure 17 displays the results of this macro on the grading sheet example in Figure 1.

Listing 10: SimpleRangeFilter uses two columns

```
Sub SimpleRangeFilter()
  Dim oSheet          ' Sheet to filter.
  Dim oRange          ' Range to be filtered.
  Dim oFilterDesc     ' Filter descriptor.
  Dim oFields(1) As New com.sun.star.sheet.TableFilterField

  oSheet = ThisComponent.getSheets().getByIndex(0)
  oRange = oSheet.getCellRangeByName("A1:H11")

  REM If argument is True, creates an
  REM empty filter descriptor.
  oFilterDesc = oRange.createFilterDescriptor(True)

  REM Setup a field to view cells with content that
  REM start with the letter B.
  With oFields(0)
    .Field = 0              ' Filter column A (Student names).
    .IsNumeric = False      ' Use a string, not a number.
    .StringValue = "b.*"    ' Every name starting with a B.
    .Operator = com.sun.star.sheet.FilterOperator.EQUAL
  End With
  REM Set up a field that requires at least one of the conditions.
  REM This new condition requires a value less than or
  REM equal to 90.
  With oFields(1)
    .Connection = com.sun.star.sheet.FilterConnection.OR
    .Field = 6              ' Filter column G (Test #1 grades).
    .IsNumeric = True       ' Use a number
    .NumericValue = 90      ' Scores less than 90
    .Operator = com.sun.star.sheet.FilterOperator.LESS_EQUAL
  End With

  oFilterDesc.setFilterFields(oFields())
  oFilterDesc.ContainsHeader = True
  oFilterDesc.UseRegularExpressions = True
  oRange.filter(oFilterDesc)
End Sub
```

	A	B	C	D	E	F	G	H
1	Student	HW #1	HW #2	HW #3	Quiz #1	Quiz #2	Test #1	Average
2	Georgia	70	80	55	39	75	67	64.33
3	David	75	86	91	40	88	79	76.50
4	Ferdinand	85	93	73	60	50	72	72.17
5	Haley	85	93	82	70	75	76	80.17
6	Charles	80	93	73	80	75	84	80.83
7	Bethany	95	100	82	80	88	93	89.67
8	Jennifer	85	93	73	80	100	90	86.83
12								

Figure 17: Grading sheet filtered by test scores under 90% and student names that begin with "B"

Advanced filters

In Calc, the criteria for an advanced filter are stored in a sheet rather than entered into a dialog. As a result, you must first set up a cell range that contains the criteria before you use the Advanced Filter dialog (Figure 18).

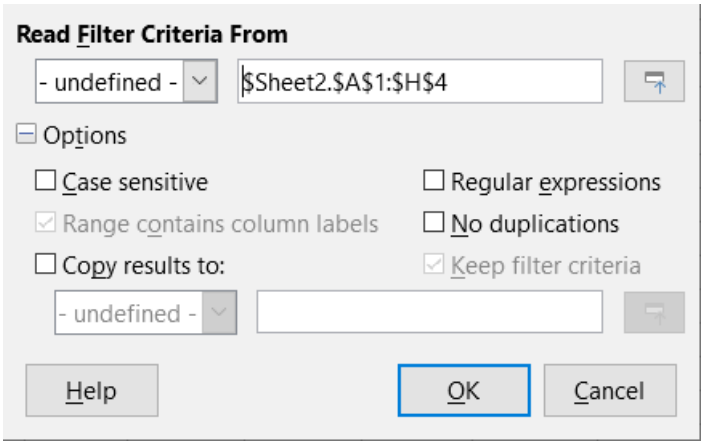

Figure 18: Advanced Filter dialog

To set up a criteria range:

1) Copy the column headings of the range to be filtered to an empty space in a sheet. It does not need to be the same sheet as the one with the source range.
2) Enter filter criteria underneath the column headings in the criteria range. Each individual criterion in the same row is connected with AND, while the criteria groups from each row are connected with OR. Empty cells are ignored. Up to eight criteria rows may be defined for a filter.

After creating a criteria range, set up an advanced filter as follows:

1) Select the cell range that you wish to filter.
2) Go to **Data > More Filters > Advanced Filter** in the Menu bar to open the Advanced Filter dialog (Figure 18).
3) In the *Read Filter Criteria From* field, enter the address for a named range, either by selecting a named range from the drop-down box, typing in a reference, or selecting cells from a sheet. Remember to use the **Shrink / Expand** button if you need to temporarily minimize the dialog while selecting cells.
4) Click **OK** to apply the filter and close the dialog.

Advanced filter options are the same as standard filter options, and are described in further detail in Chapter 2, Entering, Editing, and Formatting Data.

Advanced filter example

Figure 19 demonstrates an example filter range for the grading sheet example in Figure 1:

	A	B	C	D	E	F	G	H
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
2		>75	>75	>75				
3	Ferdinand							
4								

Figure 19: Advanced filter criteria range (in Sheet 2)

In this range, there are two criteria groups: the first displays the records of students who scored above a 75% in every homework, and the second displays records of any student named Ferdinand. Figure 20 displays the result of this filter operation using these criteria:

	A	B	C	D	E	F	G	H
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
2	**Andrew**	90	100	82	90	88	92	90.33
3	**Bethany**	95	100	82	80	88	93	89.67
6	**Emily**	100	100	81	100	75	94	91.67
7	**Ferdinand**	85	93	73	60	50	72	72.17
9	**Haley**	85	93	82	70	75	76	80.17
10	**Ian**	100	100	91	90	100	96	96.17
12								

Figure 20: Grading sheet example filtered using an advanced filter

Using an advanced filter with macros

Applying an advanced filter with a macro works similarly to setting up a standard filter. The key difference is that the cell range containing the filter criteria is used to create the filter descriptor. The macro in Listing 11 demonstrates how this is done using the *createFilterDescriptorByObject* method. It uses the grading sheet example in Figure 1 as the data range and the range in Figure 19 as the criteria range. The results should be the same as those found in Figure 20:

Listing 11: AdvancedRangeFilter creates a filter descriptor from the filter criteria range

```
Sub AdvancedRangeFilter()
  Dim oSheet     'A sheet from the Calc document.
  Dim oRanges    'The NamedRanges property.
  Dim oCritRange 'Range that contains the filter criteria.
  Dim oDataRange 'Range that contains the data to filter.
  Dim oFiltDesc  'Filter descriptor.

  REM Range that contains the filter criteria
  oSheet = ThisComponent.getSheets().getByIndex(1)
  oCritRange = oSheet.getCellRangeByName("A1:H3")

  REM You can also obtain the range containing the
  REM filter criteria from a named range.
  REM oRanges = ThisComponent.NamedRanges
  REM oRange = oRanges.getByName("AverageLess80")
  REM oCritRange = oRange.getReferredCells()

  REM The data that you want to filter
  oSheet = ThisComponent.getSheets().getByIndex(0)
  oDataRange = oSheet.getCellRangeByName("A1:H11")

  oFiltDesc = oCritRange.createFilterDescriptorByObject(oDataRange)
  oDataRange.filter(oFiltDesc)
End Sub
```

Table 3 contains a list of properties that correspond to advanced (and standard) filter settings:

Table 3: Advanced and standard filter properties

Function	Description
ContainsHeader	Boolean (TRUE/FALSE) that specifies if the first row or column contains headers which should not be filtered.
CopyOutputData	Boolean that specifies if the filtered data should be copied to another position in the document.
IsCaseSensitive	Boolean that specifies if the case of letters is important when comparing entries.
Orientation	An enumeration that specifies if a range is filtered by column or row: Column – com.sun.star.table.TableOrientation.COLUMNS Row – com.sun.star.table.TableOrientation.ROWS
OutputPosition	If *CopyOutputData* is TRUE, this property specifies the position where filtered data are to be copied.
SaveOutputPosition	Boolean that specifies if the *OutputPosition* position is saved for future calls.
SkipDuplicates	Boolean that specifies if duplicate entries are left out of the result.
UseRegularExpressions	Boolean that specifies if the filter strings are interpreted as regular expressions.

Copy advanced filter results to a different location

The results of an advanced filter can be extracted to a different position using the *OutputPosition* property. Copying results in this way eliminates the need for Calc to hide rows that do not match search criteria, which it would normally do if you filter in-place.

The macro code snippet in Listing 12 demonstrates how to copy filter results to a different location, and Figure 21 shows the results when this snippet is applied to the macro in Listing 11 just before the *filter* method is called. Note that the filter descriptor must first be modified before these filter settings are applied.

Listing 12: A code snippet that copies filtered results to a different locations

```
REM Copy the output data rather than filter in place.
oFiltDesc.CopyOutputData = True

REM Create a CellAddress and set it for Sheet1
REM Column B, Row 13 (remember, start counting with 0)
Dim outputCell As New com.sun.star.table.CellAddress
outputCell.Sheet = 0
outputCell.Column = 1
outputCell.Row = 12
oFiltDesc.OutputPosition = outputCell
```

Note

The *OutputPosition* property returns a copy of a struct. As a result, it is not possible to set individual values, such as the row or column, directly through this property. For example, `oFilterDesc.OutputPosition.Row = 2` will not work, since it is the Row property on the copy, not the original, that changes.

	A	B	C	D	E	F	G	H	I
1	**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**	
2	**Andrew**	90	100	82	90	88	92	90.33	
3	**Bethany**	95	100	82	80	88	93	89.67	
4	**Charles**	80	93	73	80	75	84	80.83	
5	**David**	75	86	91	40	88	79	76.50	
6	**Emily**	100	100	81	100	75	94	91.67	
7	**Ferdinand**	85	93	73	60	50	72	72.17	
8	**Georgia**	70	80	55	39	75	67	64.33	
9	**Haley**	85	93	82	70	75	76	80.17	
10	**Ian**	100	100	91	90	100	96	96.17	
11	**Jennifer**	85	93	73	80	100	90	86.83	
12									
13		**Student**	**HW #1**	**HW #2**	**HW #3**	**Quiz #1**	**Quiz #2**	**Test #1**	**Average**
14		**Andrew**	90	100	82	90	88	92	90.33
15		**Bethany**	95	100	82	80	88	93	89.67
16		**Emily**	100	100	81	100	75	94	91.67
17		**Ferdinand**	85	93	73	60	50	72	72.17
18		**Haley**	85	93	82	70	75	76	80.17
19		**Ian**	100	100	91	90	100	96	96.17
20									

Figure 21: Advanced filter results copied to cell B13

Useful database-like functions

Calc has many functions that are often used in the context of databases. Some of these functions are straightforward to use (like SUM) or are familiar in the context in which they are typically used (like STDEV for statistics). A few, like the LOOKUP functions, are somewhat more infrequently used, but are nevertheless useful to know if you plan to use Calc for database tables. This section provides a condensed list of some of these functions. Further reference material for all Calc's functions can be found in the Help system.

Note

Functions with the suffix *-A* treat text values as a number with the value of zero. Blank cells are still ignored by these functions.

Function	***Category***	***Description***
AVERAGE	Statistical	Returns the average of its arguments. Ignores empty cells and cells that contain text.
AVERAGEA	Statistical	Returns the average of its arguments, but only ignores empty cells. The value of text is 0.
AVERAGEIF	Statistical	Returns the arithmetic mean of all cells in a range that satisfy a given condition.
AVERAGEIFS	Statistical	Returns the arithmetic mean of all cells in a range that satisfy given multiple criteria
COUNT	Statistical	Counts the number of numeric values in a list of arguments. Ignores empty cells and cells that contain text.
COUNTA	Statistical	Counts the number of values in a list of arguments, but counts both numeric and text arguments. Empty cells are still ignored.

Function	*Category*	*Description*
COUNTBLANK	Statistical	Returns the number of empty cells within a range.
COUNTIF	Statistical	Returns the number of cells in a range that meet the specified search criteria.
COUNTIFS	Statistical	Returns the number of cells that meet criteria in multiple ranges.
HLOOKUP	Spreadsheet	Searches for a “look-up” value in the first row of an array and returns a value from a different row in the same column.
INDEX	Spreadsheet	Returns the contents of a cell at a specified index (denoted by row and column numbers) within a range.
INDIRECT	Spreadsheet	Returns the reference specified by a text string.
LOOKUP	Spreadsheet	Returns the contents of a cell contained within a single row or column of a range or from an array.
MATCH	Spreadsheet	Searches an array for an item and returns its relative position in the array.
MAX	Statistical	Returns the largest value in a list of arguments.
MAXA	Statistical	Returns the largest value in a list of arguments. The value of text is 0.
MAXIFS	Statistical	Returns the largest value in the cells of a range that meet multiple criteria in multiple ranges.
MEDIAN	Statistical	Returns the median of a list of numbers.
MIN	Statistical	Returns the smallest value in a list of arguments.
MINA	Statistical	Returns the smallest value in a list of arguments. The value of text is 0.
MINIFS	Statistical	Returns the smallest value in the cells of a range that meet multiple criteria in multiple ranges.
MODE	Statistical	Returns the most common value in a list of values.
OFFSET	Spreadsheet	Return the value of a cell offset by certain number of rows and columns from a given reference point.
PRODUCT	Mathematical	Multiples all the numbers in a list of arguments and returns the product.
STDEV STDEV.S	Statistical	Calculates the standard deviation of a population sample.
STDEVA	Statistical	Calculates the standard deviation of a population sample. The value of text is 0.
STDEVP STDEV.P	Statistical	Calculates the standard deviation of an entire population.
STDEVPA	Statistical	Calculates the standard deviation of an entire population. The value of text is 0.
SUBTOTAL	Mathematical	Calculates the total of a subset of an array.
SUM	Mathematical	Returns the sum of a list of values.
SUMIF	Mathematical	Calculates the sum of values from cells that meet the specified search criteria.
SUMIFS	Mathematical	Returns the sum of values from cells in a range that meet multiple criteria in multiple ranges.

Function	Category	Description
VAR VAR.S	Statistical	Calculates the variance of a population sample.
VARA	Statistical	Calculates the variance of a population sample. The value of text is 0.
VARP VAR.P	Statistical	Calculates the variance of an entire population.
VARPA	Statistical	Calculates the variance of an entire population. The value of text is 0.
VLOOKUP	Spreadsheet	Searches for a “look-up” value in the first column of an array and returns a value from a different column in the same row.

Database-specific functions

Some Calc functions are specifically designed for use with a database table. With one exception (DGET), these functions are specialized forms of commonly-used functions such as COUNT, and all are denoted with the *D-* prefix (such as DAVERAGE). A brief list of these functions is given in Table 4, while more detailed descriptions of all Calc’s functions are presented in the Help system.

Note

Table 4 uses the following terms interchangeably: row and record; column and field.

Table 4: Database functions in a Calc document

Function	Description
DAVERAGE	Returns the average of all fields that match the search criteria.
DCOUNT	Counts the number of records containing numeric data that match the search criteria.
DCOUNTA	Counts the number of records containing numeric or alphanumeric data that match the search criteria.
DGET	Returns the contents of a field that matches the specified search criteria.
DMAX	Returns the maximum value in a field for every record that matches the specified search criteria.
DMIN	Returns the minimum value in a field for every record that matches the specified search criteria.
DPRODUCT	Returns the product of all values in a field that match the search criteria.
DSTDEV	Calculates the standard deviation of all values in a field that match the search criteria. The values are treated as a sample.
DSTDEVP	Calculates the standard deviation of all values in a field that match the search criteria. The values are treated as an entire population.
DSUM	Sums all values in a field that match the search criteria.
DVAR	Calculates the variance of all values in a field that match the search criteria. The values are treated as a sample.
DVARP	Calculates the variance of all values in a field that match the search criteria. The values are treated as an entire population.