Learn OpenOffice.org Spreadsheet Macro Programming: OOoBasic and Calc Automation

A fast and friendly tutorial to writing macros and spreadsheet applications

Dr. Mark Alexander Bain

From Technologies to Solutions
Learn
OpenOffice.org Spreadsheet Macro Programming
OOoBasic and Calc Automation
A fast and friendly tutorial to writing macros and spreadsheet applications
Dr. Mark Alexander Bain
PACKT PUBLISHING

Chapter 6 "Working with Databases"

In this package, you will find:

A Biography of the authors of the book

A preview chapter from the book, Chapter 6 “Working with Databases”

A synopsis of the book’s content

Information on where to buy this book

About the Author

Dr. Mark Alexander Bain

Dr. Mark Alexander Bain hasn't always been the leading authority on open-source software that you know him as now. Back in the late seventies he started work as a woodsman at Bowood Estates in Wiltshire. After that he spent a number of years working at Lowther Wildlife Park in Cumbria—it's not clear if his character made him suitable for looking after packs of wolves, or whether the experience made him the way he is now.In the mid eighties there was a general downturn in the popularity of animal parks in the UK, and Mark found himself out of work with two young sons (Simon and Michael) but with a growing interest in programming. His wife had recently bought him the state-of-the-art Sinclair ZX 81, and it was she who suggested that he went to college to study computing.Mark left college in 1989 and joined Vodafone—then a very small company—where he started writing programs using VAX/VMS. It was shortly after that, that he became addicted to something that was to drastically affect the rest of his life—Unix. His demise was further compounded when he was introduced to Oracle. After that there was no saving him. Over the next few years, Vodafone became the multinational company that it is now, and Mark progressed from Technician to Engineer, and from Engineer to Senior Engineer, and finally to Principal Engineer.At the turn of the century, general ill health made Mark reconsider his career; and his wife again came to his rescue when she saw a job advert for a lecturer at the University of Central Lancashire. It was also she who suggested that he should think about writing.Today Mark writes regularly for Linux Format, Newsforge.com, and Linux Journal. He's still teaching. And (apparently) he writes books as well.

In memory of my Father—he would've got a real real kick out of this.Thanks to my Mother and my family for their continual (and continuing) support and encouragement.Thanks also to Noel Power and Michael Meeks for help with the chapter on VBA, to Paul Hudson for introducing me to Noel and Michael, to Andrew Pitonyak—I'm pleased that I could teach something to even you, and to David Barnes for that email back in May 2006.

About the Reviewers

Andrew Pitonyak

Andrew Pitonyak is a Principal Research Scientist / Software Engineer for Battelle Memorial Institute. He has been using OpenOffice.org since it was StarOffice 5, and he is the author of "OpenOffice.org Macros Explained", "Andrew's Macro Document", and other OOo related documents (see http://www.pitonyak.org). He has a Master of Science degree in computer science and another in mathematics.In his spare time Andrew is very involved in his church, is a trained Stephen Minister and a professional puppeteer, works on his house, and spends time with his wife and daughter. He is an NRA certified firearms instructor, holds a General class amateur radio license, and spends a lot of time working with his digital camera. You can reach Andrew at andrew@pitonyak.org.

Learn OpenOffice.org Spreadsheet Macro Programming

What would you say if I asked you to name the thing that had the greatest impact on Western Society in the second half of the 20th Century? Chances are you'd say the PC—the ubiquitous Personal Computer. But that's only half the story; it wasn't the PCs themselves that caused the revolution. After all, I got my first PC, a Sinclair ZX 81 back in 1981, and although it made an interesting hobby, it certainly wasn't life changing.

By the end of the 80's I was using something that anyone today would recognize as looking like a PC, but it was still very primitive. Apart from running a word processor called Lex-WP, it was really just an interface to VAX and Unix servers.

So, what was it that turned the PC from just a useful tool into the essential, number one requirement for any business? One answer is Excel—we can even put a date to the start of this revolution—November 1987.

After starting life as Multiplan, Excel became available to everyone who was running Microsoft Windows (and who had the money). Overnight, virtually every major business became addicted to the software; and Microsoft became the giant that we know and love today.

It's not really a surprise that Excel was so successful. It was an application with which you could organize your information to analyze and manipulate your data. You could even extend the basic functionality by using macros.

And that's pretty well how things remained for the rest of the century.

However, things were about to change.

In January 1998, a new term was introduced in a meeting at Palo Alto in California—open source. Then in 2000, Sun Microsystems informed the world that they were going to join the open-source community; so on 13th October, 2000, OpenOffice.org was born.

Today, the realm of the professional spreadsheet is not just limited to those that can afford it. Today even the smallest business or individual user can use Calc, and (as we'll see in this book) we can take the basic application and bend it to our own will. Now that's a revolution.

What This Book Covers

Chapter 1 introduces you to the tools that you'll need in order to write your own macros. By the end of the chapter you'll have become acclimatized to Calc's development environment, and you'll know which buttons to press to make your life a little bit easier.

Chapter 2 starts to make use of the basic building blocks that you'll need for your macros: Libraries, Modules, Subroutines, and Functions. By the end of the chapter you'll have your first macro up and running.

Chapter 3 gives an overview of the objects that are built into Calc, and which we can make use of in order to create macros that perform quite complex operations; we'll see just how easy they are to use. We'll also see where to get further information on these objects.

Chapter 4 is where we really get into writing macros. Here you'll learn how to manipulate the contents of one (or more) spreadsheets—and after all, that's what we're here for, isn't it?

Chapter 5 looks at how we can format the data contained in our spreadsheet—it doesn't matter how accurate our data is, if all of the columns overlap each other making the contents impossible to read.

Chapter 6 is an introduction to databases—how to access them, how to display the results of queries in a spreadsheet, and how to change the contents of the databases themselves.

Chapter 7 explains how to make use of other documents (such as charts) within Calc, and how they can be sources of information; for instance, the contents of websites.

Chapter 8 moves away from purely writing code, and shows how you can build a user interface—by building your own dialogs.

Chapter 9 brings everything together. By the end of the chapter you'll be able to create and distribute a complete application.

Chapter 10 takes a look into the future of Calc, and what to do if you're moving from Excel to Calc but don't want to have to rewrite all of your code.

6
Working with Databases

Sphen's grin rapidly evaporated.

"What do you mean? What database?"

"You don't think that we'd be stupid enough to store all of the information here, do you?"

Sphen looked Pygoscelis up and down. The grin returned as he pointed the pistol at Korora and he pulled back the hammer.

"Well, I don't need both of you to get the data back."

"Actually, you do. I only have a password for some of the data. Korora has the password for the rest. You need both of us alive."

Sphen carefully released the hammer and put the gun into his pocket.

"OK. You've won this round. Now, about these databases..."

And so, as Sphen said, about these databases...

In Chapter 6 (as you may have guessed), we're going to be looking at how to use Calc macros with information stored in a database. By the end of the chapter, you should be able to:

- Use a macro to connect to connect to a database
- Use **SQL** (**Structured Query Language**) in a macro to extract information from a database and use it in a spreadsheet
- Use macros to add data to a database
- Use macros to update data in a database

What we *won't* be looking at is:

- How to design and build a database
- How to set up database connectivity using ODBC (Open DataBase Connectivity)
- The full extent of SQL—we'll only be looking at the basics, just enough for us to read and write to a database

So let's get started by accessing a database.

Accessing Databases

Having decided that we want to use a database, it might be worth thinking about databases in general for a moment.

If you google "What is a database?", then you'll find a myriad of definitions, but they all boil down to the same thing—A database is a structured set of data. So, using this definition, a database could actually be a website or text files in a directory. However, there are limitations to this type of database:

- Searching can be difficult and often slow.
- It's easy for a number of people to be reading the data at the same time, but writing to the data at the same time can cause problems.
- The ways in which data can be extracted and analyzed are usually very limited.

And that, of course, is why most people use one of the readily available commercial databases or preferably one of the many free and open-source ones.

Which Databases can We Use?

I can't advise you on which database to choose. You may be in an organization that limits you to the database that the company has already bought, for example Oracle, MS Access, and so on.

If you have a completely open choice, then you might choose from Kexi, MySQL, and PostgreSQL.

If you do use any of these databases, then you'll have to set up the database connectivity, and this depends on your system and your preferences. A common one is **Open DataBase Connectivity** (**ODBC**). You'll need to do two things:

1. Install the database-specific drivers for your system.

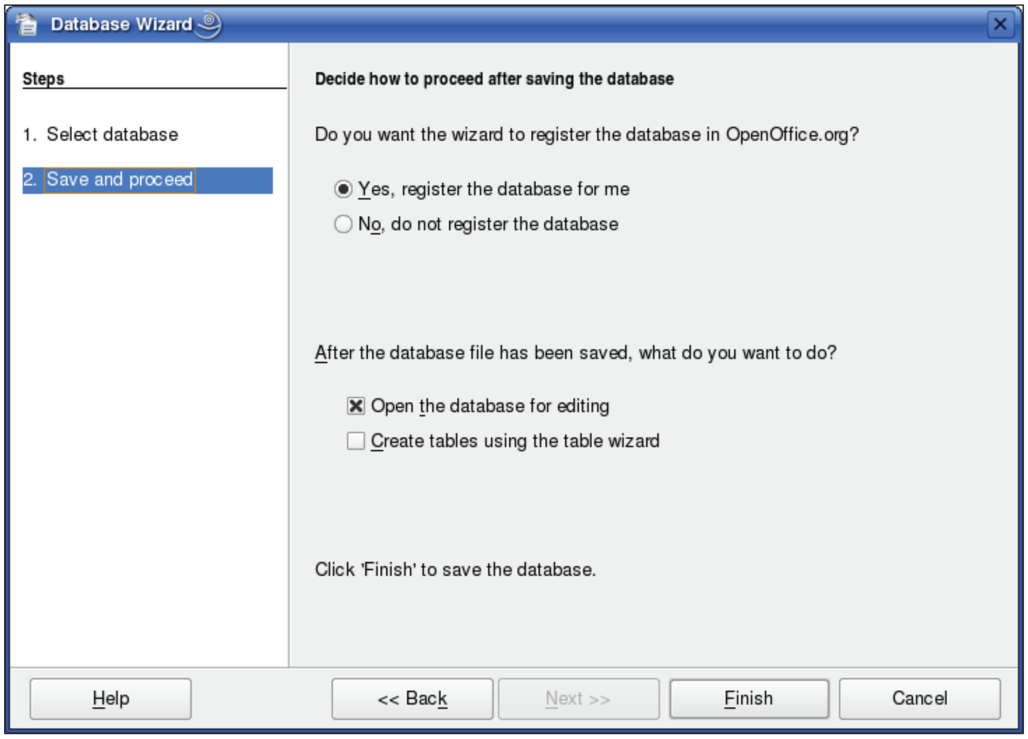
2. Use the database-connectivity software for your system to set up the connection. The software could be (for example) unixODBC for Linux or MS ODBC on Windows.

On the other hand, since you're already using OpenOffice.org, you could just stick with OpenOffice.org's Base database.

Registering the Database as an OOo Data Source

Once you've set up the database connectivity, you can register it as a data source with OpenOffice.org—this makes accessing the data just that little bit simpler. It's easily done by going to the **OOo** menu and selecting **File | New | Database**. You'll be given the choice of selecting an existing database (either an existing `OpenOffice.org Base` file or a database connection) or you can create a completely new database.

Once you've decided what you want to do, then OOo will register the database for you:



Decide how to proceed after saving the database

Do you want the wizard to register the database in OpenOffice.org?

● Yes, register the database for me
○ No, do not register the database

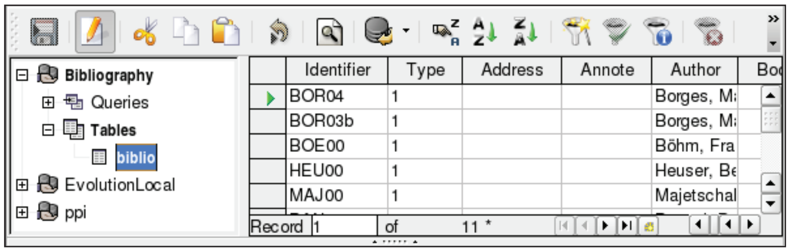
After the database file has been saved, what do you want to do?

[✓ Open the database for editing]
☐ Create tables using the table wizard

Click 'Finish' to save the database.

Viewing Registered Data Sources

Undoubtedly, at some point you'll want to see a list of registered data sources. One way is to click on **View | Data Sources** (or press *F4*).

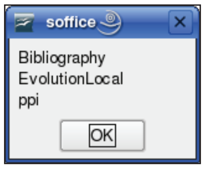


Identifier	Type	Address	Annote	Author	Bo
BOR04	1			Borges, M	
BOR03b	1			Borges, M	
BOE00	1			Böhm, Fra	
HEU00	1			Heuser, Be	
MAJ00	1			Majetschal	

However, you may prefer to use a macro to get the list:

```
Sub Main
    list_available_databases
End Sub

Sub list_available_databases
    Dim dbContext as Object
    Dim dbNames
    Dim d as Integer
    Dim dbText as String

    dbContext = createUnoService("com.sun.star.sdb.DatabaseContext")
    dbNames = dbContext.getElementNames()
    For d = 0 To UBound(dbNames())
        dbText = dbText +  dbNames(d) + chr(10)
    Next d
    msgbox dbText
End Sub
```

The macro creates a `DatabaseContext` service and accesses its `getElementNames` method to create an array containing the list of data sources. The macro then loops through the array to create a string that can be displayed in a text box:



```
Bibliography
EvolutionLocal
ppi
```

Having identified the data sources, you'll want to do something useful with them—this means connecting to the databases and then getting information from the tables contained in them.

Connecting to a Database

We've already seen that we can use the `DatabaseContext` service to enable us to see the list of registered data sources. Next, we can use the service to connect to the database:

```
Sub Main
    Dim db as Object
    db = connect_to_database("ppi")

    'At the end of the session the connection should end
    'automatically
    'but it is best just to make sure of the job:
    disconnect_from_database (db)
End Sub

Sub disconnect_from_database (db as Object)
    db.close
    db.dispose()
End Sub

Function connect_to_database (dbName as String) as Object
    Dim dbContext As Object
    Dim oDataSource As Object

    dbContext = _
        createUnoService("com.sun.star.sdb.DatabaseContext")
    oDataSource = dbContext.getByName(dbName)
    connect_to_database = oDataSource.GetConnection("","")
End Function
```

You'll notice that the final line of the code contains two empty strings:

```
connect_to_database = oDataSource.GetConnection("","")
```

These are the spaces for a user name and a password in case you need these for connecting to the database. If your database does need a user name and a password (and obviously it's sensible to use them), then you would use something like:

```
db = oDataSource.GetConnection("bainm","nottelling")
```

So, that's easy enough. Next we can look at the actual tables in the database.

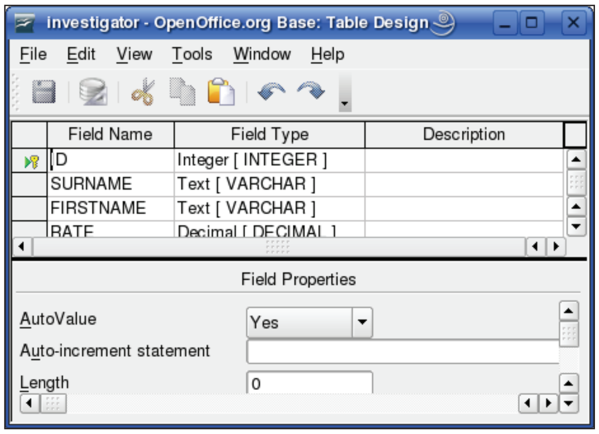
Accessing Database Tables

As I mentioned at the start of the chapter, we're not going to delve into the intricacies of designing and creating the database tables. That's because the techniques and tools that you'll have at your fingertips will vary according to the actual database that you're using. For example, if you've got MySQL or PostgreSQL, then you could create the tables from the command line or use third-party tools to create the table visually.

On the other hand, if you've got MS Access or you're using OpenOffice.org's Base application, then you will use their built-in table creation forms and wizards.

However, irrespective of the database you're using, it's worth spending the time constructing the data structure *before* you start creating macros. It's easier to build macros around well-structured data rather than trying to fit your data around the macros.

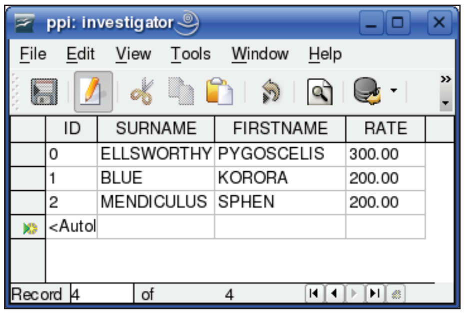
If you are using Base, then you can build tables easily using Base's **Table Design** dialog.


investigator - OpenOffice.org Base: Table Design
File Edit View Tools Window Help
Field Properties

Field Name	Field Type	Description
ID	Integer [INTEGER]	
SURNAME	Text [VARCHAR]	
FIRSTNAME	Text [VARCHAR]	
RATE	Decimal [DECIMAL]	

AutoValue: Yes
Auto-increment statement:
Length: 0

You can then use Base (with the table in **Edit** mode) to populate the tables manually (since we haven't written a macro to do it yet) as shown in the figure below:



ID	SURNAME	FIRSTNAME	RATE
0	ELLSWORTHY	PYGOSCELIS	300.00
1	BLUE	KORORA	200.00
2	MENDICULUS	SPHEN	200.00
<AutoI			

Record 4 of 4

Of course, for the time being, you may prefer to use the database that comes with Base—Biblio.

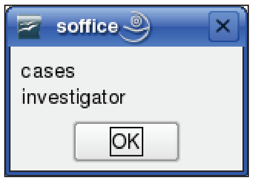
Earlier we saw that using the `DatabaseContext` service allowed us to list the available data sources and connect to any of the databases registered with OpenOffice.org.

Next we can use the service to obtain a list of the tables in the database:

```
Sub Main
    Dim db As Object
    db = connect_to_database ("ppi")
    list_tables (db)
    disconnect_from_database (db)
End Sub

Sub list_tables (db as Object)
    Dim dbTables as Object
    Dim dbTableNames
    Dim opText as String

    dbTables = db.getTables
    dbTableNames = dbTables.getElementNames
    opText = join ( dbTableNames , chr(10))
    msgbox opText
End Sub
```

You'll notice that the `list_tables` subroutine does not create the connection to the database; instead the `connect_to_database` function returns this as the variable `db`. This then represents the connection, and can be passed to any of the macros that you write.

When you run the `Main` macro, you'll see the list of tables displayed.



```
cases
investigator
```

Of course, if you're using your own database or OOo's Biblio, then you'll get a different list—but you get the idea.

Don't forget you'll always have to run the `connect_to_database`; without this you have no connection to the database. If you do want to write any stand-alone macros that you can run independently and which use the database, then you'll have to include the statement

```
        db = connect_to_database ("ppi")
```

In our example `Main` does all of the work for us—calling the subroutine to connect to the database, and then calling a macro to list the tables. So having obtained a list of all of the tables in the database we can now start to think about extracting data from them.

Running Queries on the Tables

We're going to use the following simple procedure:

- Connect to the database
- Send a SQL statement to the database
- Get a result from the database
- Make use of the results

Since we've already learned how to connect to the database, let's carry on and do the rest:

```
Sub Main
    Dim db As Object
    db = connect_to_database ("ppi")
    simple_query (db)
    disconnect_from_database (db)
End Sub

Function capitalize (iName as String) as String
    Dim wordStart as String
    Dim wordEnd as String

    wordStart = UCase (Mid (iName,1,1))
    wordEnd = LCase (Mid (iName,2))
    capitalize  = wordStart & wordEnd
End Function
Sub simple_query (db as Object)
    Dim oSql as String
    Dim i as Integer
    Dim oRowSet as Object
    Dim oResult as String

    oSql = _
    "SELECT SURNAME, FIRSTNAME, RATE" _
    & " FROM ""investigator"""

    oRowSet = createUnoService("com.sun.star.sdb.RowSet")


    oRowSet.activeConnection = db
    oRowSet.Command = oSql
    oRowSet.execute
```

```
        while oRowSet.Next
        oResult = oResult _
        & capitalize (oRowSet.getString(2)) & " " _
           & capitalize (oRowSet.getString(1)) & " " _
            & "£" & oRowSet.getFloat(3) _
            & " " & chr(13)
            wend
        msgbox oResult, ,"PPI Hourly Rate "
    End Sub
```

You'll see from the `simple_query` subroutine that we create a `RowSet` service, which then uses the database connection to send a SQL statement to the database and retrieve the results. Next, we just have to loop through the `RowSet` extracting information from it line by line.

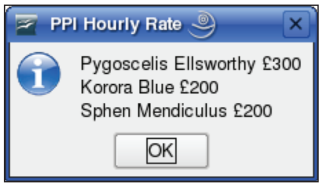
There are a couple of things to consider before we move on. The first one being the following three lines of the code:

```
        oRowSet.activeConnection = db
        oRowSet.Command = oSql
        oRowSet.execute
```

Personally, I'm perfectly happy with this, but you may prefer the `With` format. If so, then you could change the code to:

```
        With oRowSet
           .activeConnection = db
           .Command = oSql
           .execute
        End With
```

The other thing is the `capitalize` function. This is just used to format any names nicely (since they may have been saved as all lowercase, or all uppercase, or any mixture of the two). The end result is as follows:



```
Pygoscelis Ellsworthy £300
Korora Blue £200
Sphen Mendiculus £200
```

Putting it All into a Spreadsheet

We've seen just how easy it is to get data out of a database. We can now look at actually doing something useful with the information. The most obvious thing to do is to load the data into a spreadsheet:

```
Sub Main
    Dim db As Object
    db = connect_to_database ("ppi")
    load_investigator (db)
    disconnect_from_database (db)
End Sub

Sub load_investigator (db as Object)
    Dim oDoc as Object
    Dim oURL as String
    Dim oSheet as Object
    Dim oCell as Object
    Dim oRowSet as Object
    Dim i as Integer

    oURL = "private:factory/scalc"
        oDoc = starDeskTop.loadComponentFromURL _
                     (oURL, "_blank", 0, Array() )
    oSheet = oDoc.Sheets(0)
    oSheet.Name = "PPI Investigator"

    oRowSet = get_rowset (db, sql_select _
               ("investigator", Array("SURNAME","FIRSTNAME","RATE")))

    While oRowSet.Next
        i = i + 1
       oCell = oSheet.getCellByPosition(0,i)
       'Remember to use capitalize from page 8
       oCell.String = capitalize (oRowSet.getString(2))
        oCell = oSheet.getCellByPosition(1,i)
        oCell.String = capitalize (oRowSet.getString(1))
        oCell = oSheet.getCellByPosition(2,i)
        oCell.Value = capitalize (oRowSet.getString(3))
    Wend
End Sub
```

You'll realize, of course, that there's nothing new here. We are just using some of the techniques that we've learned in Chapter 4 (where we manipulated data in a spreadsheet), Chapter 5 (where we formatted the contents of spreadsheets), and this chapter (where we've extracted information from a database, and introduce functions such as `capitalize`).

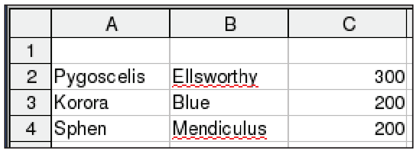
However, there are a couple of functions that we call from `load_investigator` that you may be wondering about. The first is the function `sql_select`. If you do not prefer to create the SQL yourself, you can use this to do the job for you. All you have to do is feed it with the table name (as a string) and the list of fields (as an array):

```
Function sql_select (iTable as String, iFields())
    sql_select = _
    "SELECT " & join (iFields,",") _
    & " FROM """ & iTable + """"
End Function
```

The other function is `get_rowset`. If you don't want to remember how to create a `RowSet`, then you can use this to do the work for you. Just pass a SQL statement to it and it will return the `RowSet` as an object to you:

```
Function get_rowset (db as Object, iSql as String) as Object
    Dim oRowSet as Object

    oRowSet = createUnoService("com.sun.star.sdb.RowSet")
    oRowSet.activeConnection = db
    oRowSet.Command = iSql
    oRowSet.execute

    get_rowset = oRowSet
End Function
```

After all this, the result is a spreadsheet containing the result of the query that you've sent to the database:



	A	B	C
1			
2	Pygoscelis	Ellsworthy	300
3	Korora	Blue	200
4	Sphen	Mendiculus	200

I'm sure that you'll find these techniques very useful. However, it's rather limiting, isn't it? You'll need a completely new macro for every set of data that you want to load. Instead of that let's investigate a more generic approach.

Loading Data into Custom Worksheets

I don't know about you, but I enjoy using code to come up with fresh solutions to problems. I don't particularly enjoy just typing the same old lines of code again, and again, and again. So let's look at changing `load_investigator` into a macro that will work for any data that we want to insert into the spreadsheet:

```
Sub Main
    Dim doc as Object
    Dim db As Object

    db = connect_to_database ("ppi")
    doc = open_spreadsheet
    ppi_sheets (db, doc)
    disconnect_from_database (db)
End Sub
```

We're still using the `connect_to_database` function (for obvious reasons); however, we've also added `open_spreadsheet` and `ppi_sheets`.

```
Function open_spreadsheet
    Dim oURL as String

    oURL = "private:factory/scalc"
    open_spreadsheet = starDeskTop.loadComponentFromURL (oURL, _
        "_blank", 0, Array() )
End Function
```

You'll realize immediately that all `open_spreadsheet` does is encapsulate code that we've already used often enough and open a blank spreadsheet. It's in `ppi_sheets` that all of the database work is done:

```
Sub ppi_sheets (db as Object, iDoc as Object)
    Dim sheetName as String
    Dim fields
    Dim oRowSet as Object
    Dim r as Integer

    sheetName = "PPI Investigator"
    oRowSet = get_rowset (db, sql_select _
        ("investigator", _
            Array("SURNAME","FIRSTNAME","RATE", "'END_OF_RECORD'")))
    r = oRowSet.RowCount + 1
    load_sheet (iDoc, sheetName, oRowSet)
    capitalize_column (iDoc, sheetName,0,r)
    capitalize_column (iDoc, sheetName,1,r)
    format_column (iDoc, sheetName,2,r,"£#,##0.00")
    deleteSheets (iDoc)
End Sub
```

OK, nothing contentious here. You can work out from the function name what's going on. However, you may be wondering about the `'END_OF_RECORD'` used in the SQL statement. It's not actually a field name; we're using it to mark the final record in each row. You can see it in use in the `load_sheet` subroutine:

```
Sub load_sheet (iDoc as Object, iName as String, iRowSet as Object)
    Dim oSheet as Object
    Dim oCell as Object
    Dim r as Integer
    Dim c as Integer
    Dim endMarker as String

    oSheet = iDoc.createInstance ( "com.sun.star.sheet.Spreadsheet" )
    iDoc.Sheets.insertByName ( iName, oSheet )

    If Not isNull (iRowSet) Then
        While iRowSet.Next
            r = r + 1
            c = 1
            endMarker = ""
            While endMarker <> "END_OF_RECORD"
               oCell = oSheet.getCellByPosition(c - 1,r)
             if isNumeric (iRowSet.getString(c)) Then
               oCell.Value = iRowSet.getString(c)
             Else
                 oCell.String = iRowSet.getString(c)
             End If
             c = c + 1
               endMarker = iRowSet.getString(c)
           Wend
       Wend
    End If
End Sub
```

The `load_sheet` subroutine is useful because you can use it to add a completely new worksheet, give it an appropriate name, and load it with the contents of a `RowSet`.

Having loaded the data that we require, the `ppi_sheets` subroutine then calls a few custom-build subroutines to carry out some formatting. The macro `capitalize_column` takes the `capitalize` function and applies it to a portion of a column:

```
Sub capitalize_column (iDoc as Object, _
                       iSheetName as String, _
                       iColumn as Integer, _
                       iRow as Integer)
    Dim oCell as Object
    Dim oSheet as Object
    Dim r as Integer

    oSheet = iDoc.Sheets.getByName (iSheetName)

    For r = 0 to iRow
       oCell = oSheet.getCellByPosition(iColumn,r)
```

```
            oCell.String = capitalize(oCell.String)
        Next r
    End Sub
```

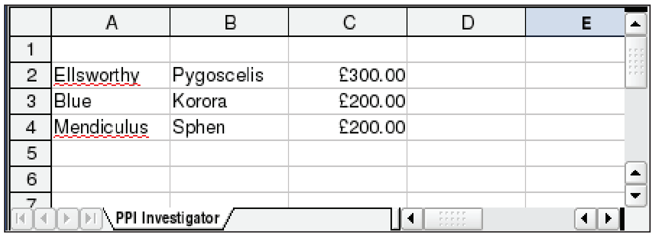
We can also apply number formats to any column that we want:

```
    Sub format_column ( iDoc as Object, _
                        iSheetName as String, _
                        iColumn as Integer, _
                        iRow as Integer, _
                        iFormat as String)
        Dim oCell as Object
        Dim oSheet as Object
        Dim r as Integer

        oSheet = iDoc.Sheets.getByName (iSheetName)

        For r = 0 to iRow
            oCell = oSheet.getCellByPosition(iColumn,r)
            'The getNumberFormat function was created in chapter 5
            oCell.NumberFormat = getNumberFormat (iDoc, iFormat)
        Next r
    End sub
```

Finally, we remove any unused worksheets:

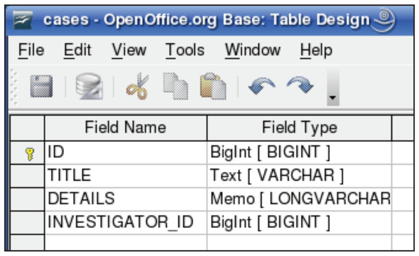
```
    Sub deleteSheets (iDoc as Object)
        iDoc.Sheets.removeByName("Sheet1")
        iDoc.Sheets.removeByName("Sheet2")
        iDoc.Sheets.removeByName("Sheet3")
    End Sub
```

At the end of the process we've got a formatted spreadsheet with only the necessary worksheets:



	A	B	C	D	E
1					
2	Ellsworthy	Pygoscelis	£300.00		
3	Blue	Korora	£200.00		
4	Mendiculus	Sphen	£200.00		
5					
6					

Of course, we may have other tables in the database, for example our friend Pygoscelis would have a table containing all the PPI cases:

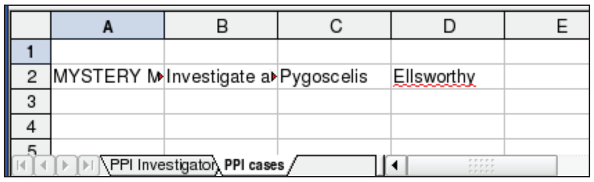

cases - OpenOffice.org Base: Table Design

	Field Name	Field Type
🔑	ID	BigInt [BIGINT]
	TITLE	Text [VARCHAR]
	DETAILS	Memo [LONGVARCHAR
	INVESTIGATOR_ID	BigInt [BIGINT]

And to use another table we only have to add a small amount of extra code:

```
sheetName = "PPI cases"
oSql =_
      "select TITLE,DETAILS,FIRSTNAME,SURNAME,'END_OF_RECORD' " _
      & " from ""investigator"" i, ""cases"" c "_
      & " where i.ID = c.INVESTIGATOR_ID" _
      & " order by c.ID "
oRowSet = get_rowset (db, oSql)
load_sheet (iDoc, sheetName, oRowSet)
r = oRowSet.RowCount + 1
capitalize_column (iDoc, sheetName,2,r)
capitalize_column (iDoc, sheetName,3,r)
```

The SQL statement is a little more complicated this time—rather than using the contents of a single table we're combining the contents of two tables (often referred to as a `Join` query). The end result is a worksheet containing details from both tables:

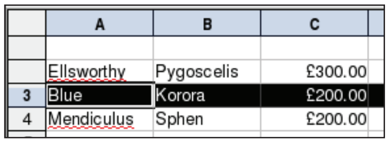


	A	B	C	D	E
1					
2	MYSTERY M	Investigate a	Pygoscelis	Ellsworthy	
3					
4					
5					

You'll notice that the column widths are not optimized, but I'm not going to do everything for you. All you have to do is create a new subroutine similar to `capitalize_column` and `format_column`, but this time using the column's `OptimalWidth` property (if you remember, we used that in Chapter 5).

Adding New Records to the Database

So far we've extracted static data from the database, but we're not limited to that. It's easy to collect information from the spreadsheet or from manual input, and then use it with an `insert` query in order to create a new set of records in a table. In this example, we first need to select a row in the PPI Investigator worksheet.



	A	B	C
	Ellsworthy	Pygoscelis	£300.00
3	Blue	Korora	£200.00
4	Mendiculus	Sphen	£200.00

We then call the `add_case` macro (by clicking on **Tools | Macros | Run Macro**...). The macro will obtain the investigator name from the selection and then ask the user (e.g. Pygoscelis) for the new case details.

```
Sub add_case
    Dim oRange as Object
    Dim oSheet as Object
    Dim oCell as Object
    Dim surname as string
    Dim firstname as string
    Dim title as string
    Dim details as string
    Dim r as Integer

    oRange =  thisComponent.getCurrentSelection.getRangeAddress
    r = oRange.startRow
    oSheet = thisComponent.CurrentSelection.getSpreadsheet
    surname = ucase(oSheet.getCellByPosition(0,r).String)
    firstname = ucase(oSheet.getCellByPosition(1,r).String)
    If surname = "" Or firstname = "" Then
        msgbox "Please select a row containing a name "
        stop
    End If
    While title = ""
         title = inputbox ("Enter case title (enter 0 to stop)")
    Wend
    If title = 0 Then
        stop
    End If
    While details = ""
        details = inputbox ("Enter case details (enter 0 to stop)")
    Wend
    If details = 0 Then
         stop
    End If
    create_case (surname, firstname, title, details)
End Sub
```

You'll notice that the subroutine collects some of the information from the spreadsheet itself.

```
    surname = ucase(oSheet.getCellByPosition(0,r).String)
    firstname = ucase(oSheet.getCellByPosition(1,r).String)
```

While the rest of the data is obtained from manual input as:

```
        title = inputbox ("Enter case title (enter 0 to stop)")
        details = inputbox ("Enter case details (enter 0 to stop)")
```

The collected information is then passed on to the `create_case` subroutine:

```
Sub create_case (iSurname as String, _
                 iFirstName as String, _
                 iTitle as String, _
                 iDetails as String)
    Dim id as Integer
    Dim oSql as String
    Dim oResult as Object
    Dim oStatement as Object
    Dim db as object

    db = connect_to_database ("ppi")

    id = investigator_id (db, iSurname, iFirstName)

    oSql = "insert into ""cases"" " _
    + "(""TITLE"",""DETAILS"",""INVESTIGATOR_ID"") values " _
    + "('" + iTitle + "','" + iDetails + "'," + id + ")"

    oStatement = db.createStatement
    oResult = oStatement.executeQuery (oSql)
    disconnect_from_database (db)
End Sub
```

You may wonder why the subroutine contains the line:

```
    db - connect_to_database ("ppi")
```

If you remember, previously the connection to the database was created when you first ran `Main`. It was, of course, released when `Main` finished. However, this time `Main` is not being called, and so in order for `create_case` to run correctly it needs its own connection (and, of course, its own disconnection).

The subroutine finally sends an `insert` SQL statement to the database:

```
    oSql = "insert into ""cases"" " _
    + "(""TITLE"",""DETAILS"",""INVESTIGATOR_ID"") values " _
    + "('" + iTitle + "','" + iDetails + "'," + id + ")"
```

However, before it carries out the `insert`, `create_case` obtains the field `investigator_id`—a number stored in the `investigator` table, and we use the `investigator_id` function:

```
Function investigator_id _
   (db as Object, iSurname as String, iFirstName as String)
    Dim oRowSet as Object
    Dim oSql
    oSql = "select ID from investigator " _
    + "where SURNAME ='" + iSurname + "'" _
    + " and FIRSTNAME = '" + iFirstName + "'"
    oRowSet = get_rowset (db, oSql)
    oRowSet.Next
    investigator_id = oRowSet.getInt(1)
End Function
```

You'll also notice that the function gets the information from a single table, but it is filtered to obtain a single ID number (by adding a *Where* clause to the SQL).

The final point to be aware of is that we don't use the `RowSet` when inserting into the database; we just create a statement and then execute the SQL:

```
    oStatement = db.createStatement
    oResult = oStatement.executeQuery (oSql)
```

Updating the Database

So far we've learned how to:

- Use SQL to query the database and use the information obtained to fill a worksheet
- Use an `insert` SQL statement in a macro to create new records in the database

The next thing that we can do is to update data already in the database; again by deriving information from the spreadsheet. Let's say that Pygoscelis has changed some of the details in his "PPI Cases" worksheet (perhaps changing one of the case titles), then all he has to do is to click on **Tools | Macros | Run Macro...** to run the macro, which will scroll through the worksheet looking for any changes:

```
Sub update_case
    Dim oRange as Object
    Dim oSheet as object
    Dim oRowSet as Object
    Dim oCell as Object
    Dim oSql as String
```

```
        Dim r as Integer
        Dim c as Integer
        Dim oResult as Object
        Dim oStatement as Object
        Dim db as Object

        db = connect_to_database ("ppi")

        oRange =  thisComponent.getCurrentSelection.getRangeAddress
        oSheet = thisComponent.CurrentSelection.getSpreadsheet
        oRowSet = get_rowset (db, sql_select _
                    ("case", Array("TITLE", "DETAILS")))
        r = 1
        oCell = oSheet.getCellByPosition(0,r)
        While oCell.String <> "" And Not oRowSet.isLast
            oRowSet.absolute(r)

            oCell = oSheet.getCellByPosition(0,r)
            If oCell.String <> oRowSet.getString(1) Then
                oSql = """TITLE"" = '" & oCell.String & "'"
            End If

            oCell = oSheet.getCellByPosition(1,r)
            If oCell.String <> oRowSet.getString(2) Then
               If oSql <> "" Then
                     oSql = oSql & ","
               End If
               oSql = oSql & """DETAILS"" = '" & oCell.String & "'"
            End If
            If oSql <> "" Then
             oSql =  "Update ""cases"" set " & oSql _
             & " where ""TITLE"" = '" & oRowSet.getString(1) & "'" _
             & " and ""DETAILS"" = '" & oRowSet.getString(2) & "'"
            End If
            oStatement = db.createStatement
            oResult = oStatement.executeQuery (oSql)
            r = r + 1
        Wend
        disconnect_from_database (db)
    End Sub
```

The process of updating the database is similar to inserting new data:

- Information is obtained from the spreadsheet by making use of the range address.
- The appropriate SQL statement is built from the information.
- The SQL statement is passed to the database, but again no `RowSet` is created.

The main difference is that rather than taking an individual line of data from the spreadsheet, every line is checked. The macro then compares each line with the contents of the equivalent line in the `RowSet`. If there is a difference, then an update is carried out.

Summary

In this chapter we've learned how to make use of the data stored in a database. And so we're now able to use a macro to obtain and use data in a database, create new records in a database, and update existing records in a database.

If you're using any database other that OpenOffice.org Base, then you'll have to install drivers for the database that you're going to be using, configure your ODBC software so that the database is available on your system, and register the database with OpenOffice.org as a Data Source. You can then connect to it using the `DatabaseContext` service.

With a connection in place you can extract information from the database, add new records to tables in the database, and update existing records in the database. You have also learned how to extract information from the database and also how to insert a new record or update an existing one.

Anyway...

Pygoscelis ignored Sphen and slumped back in his chair. All he was really aware of was the increasing pain in his head and Korora's heart beating through his back. He tried to concentrate on the contents of the report now smoldering in the bin.

At last he gave in and slipped back into oblivion.

In Chapter 7 we'll be looking at how Pygoscelis created that report when we look at '*Working with Other Documents*'.