



Diagra 2.0 Fund Charts – step-by-step tutorial

Prepared By **Andy Robinson**
Date **8 February 2010**
Draft No. **3**

Executive Summary

ReportLab's Diagra product lets you create data-aware charts and drawings. It can be used in many industries and many ways. The most comprehensive documentation is in the main manual, "**diagra-userguide.pdf**". This is a supplementary tutorial intending to provide step-by-step instructions for a first-time user.

Many users are working in the fund management industry producing highly similar charts. This extra manual provides step-by-step tutorial to create several data-aware fund related charts. It is intended to be used with a sample database, delivered in a file 'diagracharts.zip', which includes both data and runnable chart modules similar to the ones described herein.

This manual is still being expanded, but should be sufficient to let a first-time user create a working data aware chart and carry out basic procedures. In the meantime feedback on what to add to this tutorial would be most welcome.

Table of Contents

1. Installation and prerequisites.....	3
2. Installing the sample database.....	3
3. Verifying the ready-made charts work.....	4
4. Creating a data-aware pie chart.....	5
4.1. Plan your chart.....	5
4.2. Prepare your data.....	5
4.3. Create a basic chart module.....	6
4.4. Connect to the database.....	10
4.5. Making data associations.....	12
4.6. Planning output file names and formats.....	14
5. Creating a fund/index comparison chart.....	14
5.1. Planning:.....	14
5.2. Laying out the chart.....	14
5.3. Making data associations and testing.....	16
6. Creating a text table.....	17
6.1. Planning your work.....	17
6.2. Table Design.....	17
6.3. Create a chart module.....	19
6.4. Create the table	19



6.5. Connect to mysql and associate with data.....	20
7. Updating the database from data in Excel or files.....	22
7.1. Manual import of one data set.....	22
7.2. Automating imports.....	23
8. Windows and Mac OS X platform notes.....	23

1. Installation and prerequisites

This presumes you have

- a healthy installation of Diagra. The installation steps for all platforms are covered in depth on ReportLab's site at <http://www.reportlab.com/software/installation>
- a MySQL or SQL Server database server, running either on the local or remote machine. *Note: if you're competent with another SQL database, you will probably find it easy to adapt the sample data in here as we need nothing more than standard SELECT statements.*
- the Python MYSQL adapter installed. *You could get this from <http://pypi.python.org/pypi/MySQL-python/>*
- a tool for doing queries and viewing the results. This can be done with a command-line (terminal) mysql client, or a graphical tool such as the MySQL Query Browser.

2. Installing the sample database

You can download the sample data files from <http://www.reportlab.com/docs/diagracharts.zip>.

The zip file contains the sample database in various formats. To import it into MySQL, create a database schema called 'fundrep' and then execute '**fundrep.sql**' to create the sample tables and data.

There are also .mdf and .ldf files which can be imported into Microsoft SQL Server using the 'attach' function - see the README file for more details.

The remainder of this section provides step-by-step instructions to create and populate the sample database using the MySQL Query Browser.

The first thing you need to do is to run your MySQL Query Browser program which will ask you for a number of parameters before connecting to MySQL Server. (make sure that MySQL Server is running)

Enter your connection details:

Stored Connection: this is used if you want to save your connection details. leave blank for now.

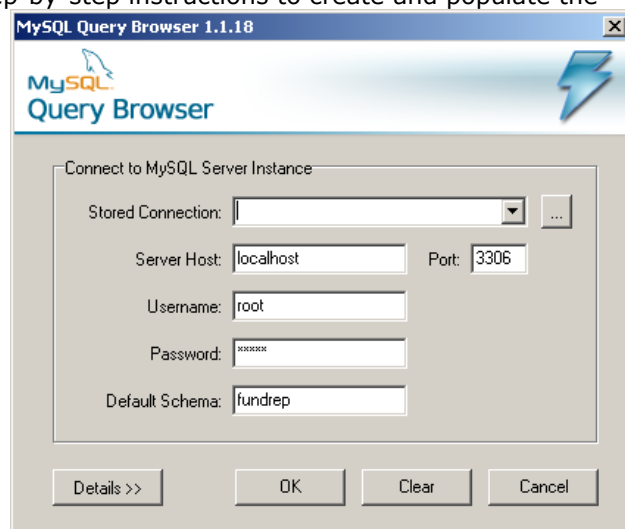
Server Host: The name of the host machine where your MySQL Server runs, or its IP address if running on another machine. (*localhost is your local machine*)

Port: default is 3306. Don't change it

Username: As it says

Password: As it says

Default Schema: This is your actual database. Insert (type) **fundrep** as your new Schema.



At this point, the program will ask you that the database doesn't exist and if want to create it.

Just click **OK**. This will connect to the Server and create an empty database for you named '**fundrep**'.

Create the database's contents:

In the sample zipped distribution, we have included a script (*fundrep.sql*) that creates a number of tables with several funds data into the MySQL database server.

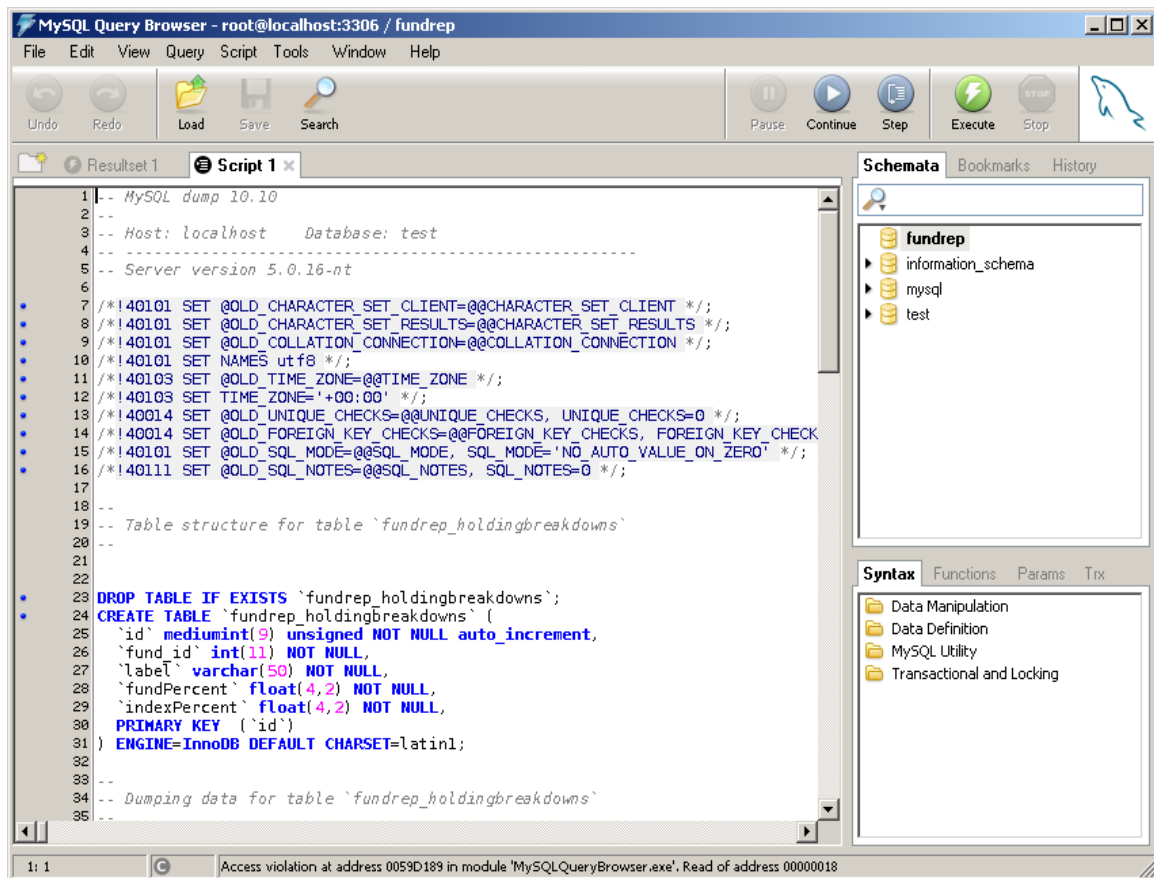
MySQL Query Browser makes easy to load the script into the server.

2.1 - To open the script in the Script Editor, choose the **Open Script ...** option from the **File** menu.

Then locate where you saved your *fundrep.sql* with the 'file open dialog' and open it.

2.2 - if your script resembles to the one below, then you should now be ready to run it.

Click on **Execute Selection** option from the **Script** menu or alternatively click on the green rounded icon on the top right had corner of your Query Browser.



3. Verifying the ready-made charts work

To be completed. This will describe executing the readymade chart modules from a DOS or Mac prompt to verify they produce output. This is intended to be a test that you are set up correctly prior to starting the tutorial.

4. Creating a data-aware pie chart

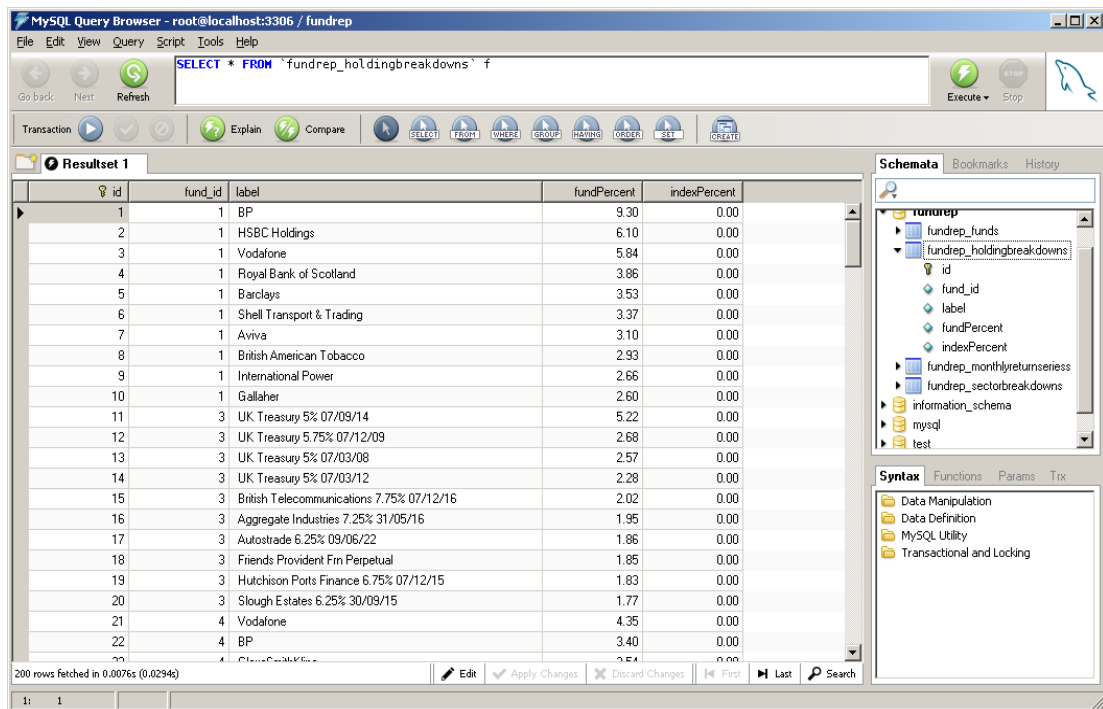
This is the bulk of the tutorial. We will start with a very simple chart, showing a pie of the top ten holdings in a given fund. We will then step through a couple more times with other chart types. Please start with this example before moving onto more complex charts!

4.1. Plan your chart

It's worth a few moments deciding what to create before starting. In this case we'll create a simple rectangular drawing containing a pie, with labels around the edge, and no title. (You can flip ahead to the later screen shots to see what we are doing). *This is a particularly dumb way to plot fund holdings with dynamic data (see why¹ below), but it's the simplest teaching example!*

4.2. Prepare your data

We will need a table of suitable data, and a query to pull it out. The sample database has a table **fundrep_holdingbreakdowns** which is suitable. The screen shot below shows us doing a query in the MySQL Query Browser to pull this out. It also shows in the right pane that you can use the Query Browser to explore and remind yourself of the table and field names.



The screenshot shows the MySQL Query Browser interface. The main window displays a query result set for the table `fundrep_holdingbreakdowns`. The query is `SELECT * FROM fundrep_holdingbreakdowns` f`. The result set contains 200 rows, with the first 22 rows visible. The columns are `id`, `fund_id`, `label`, `fundPercent`, and `indexPercent`. The data is sorted by `fundPercent` in descending order. The right pane shows the schema for the `fundrep` database, including the `fundrep_holdingbreakdowns` table and its fields: `id`, `fund_id`, `label`, `fundPercent`, and `indexPercent`.

id	fund_id	label	fundPercent	indexPercent
1	1	BP	9.30	0.00
2	1	HSBC Holdings	6.10	0.00
3	1	Vodafone	5.84	0.00
4	1	Royal Bank of Scotland	3.86	0.00
5	1	Barclays	3.53	0.00
6	1	Shell Transport & Trading	3.37	0.00
7	1	Aviva	3.10	0.00
8	1	British American Tobacco	2.93	0.00
9	1	International Power	2.66	0.00
10	1	Gallagher	2.60	0.00
11	3	UK Treasury 5% 07/09/14	5.22	0.00
12	3	UK Treasury 5.75% 07/12/09	2.68	0.00
13	3	UK Treasury 5% 07/03/08	2.57	0.00
14	3	UK Treasury 5% 07/03/12	2.28	0.00
15	3	British Telecommunications 7.75% 07/12/16	2.02	0.00
16	3	Aggregate Industries 7.25% 31/05/16	1.95	0.00
17	3	Autostrade 6.25% 09/06/22	1.86	0.00
18	3	Friends Provident Fm Perpetual	1.85	0.00
19	3	Hutchison Ports Finance 6.75% 07/12/15	1.83	0.00
20	3	Slough Estates 6.25% 30/09/15	1.77	0.00
21	4	Vodafone	4.35	0.00
22	4	BP	3.40	0.00

In this case, we don't need the unique row IDs, and it's always a good idea to be explicit about the column names you want in your chart. Otherwise, the chart will break if someone else adds a field to the database and your `'select *'` statement returns the wrong number of rows.

All we want to display for this lesson is the holdings and percentages, in descending order.

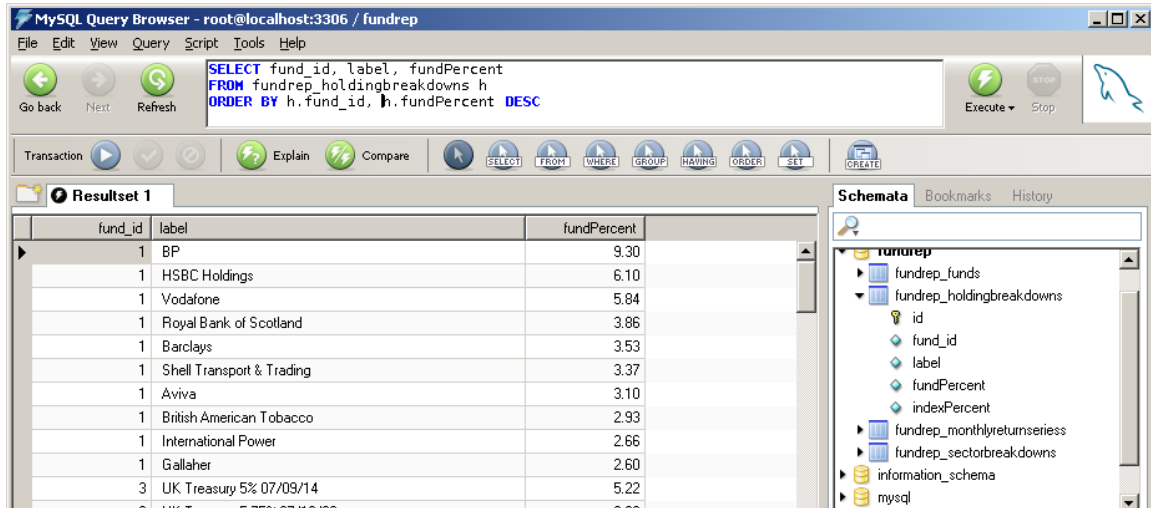
The query for this is:

```
SELECT fund_id, label, fundPercent
FROM fundrep_holdingbreakdowns h
```

¹ There are several reasons. First, a pie chart should represent the 'whole fund', and on our case the top ten holdings might only be, say, 30% of the fund's value, so we are misrepresenting data unless our database has a great big 'other = 70%' data point. Secondly, the labels around the edge will be variable length text strings. If you had several tiny holdings (say, 1-3% of the fund), these might overlap with each other and the pie itself or overrun the boundaries.

```
ORDER BY h.fund_id, h.fundPercent DESC
```

In plain English this means: show the fund id, label and percentage , ordered by fund, and then with the largest percentage first. The figure below shows the resulting query and its data. It is useful to keep the data set on screen, or a printout of it next to you.



The screenshot shows the MySQL Query Browser interface. The query editor contains the following SQL query:

```
SELECT fund_id, label, fundPercent
FROM fundrep_holdingbreakdowns h
ORDER BY h.fund_id, h.fundPercent DESC
```

The results are displayed in a table with the following data:

fund_id	label	fundPercent
1	BP	9.30
1	HSBC Holdings	6.10
1	Vodafone	5.84
1	Royal Bank of Scotland	3.86
1	Barclays	3.53
1	Shell Transport & Trading	3.37
1	Aviva	3.10
1	British American Tobacco	2.93
1	International Power	2.66
1	Gallagher	2.60
3	UK Treasury 5% 07/09/14	5.22

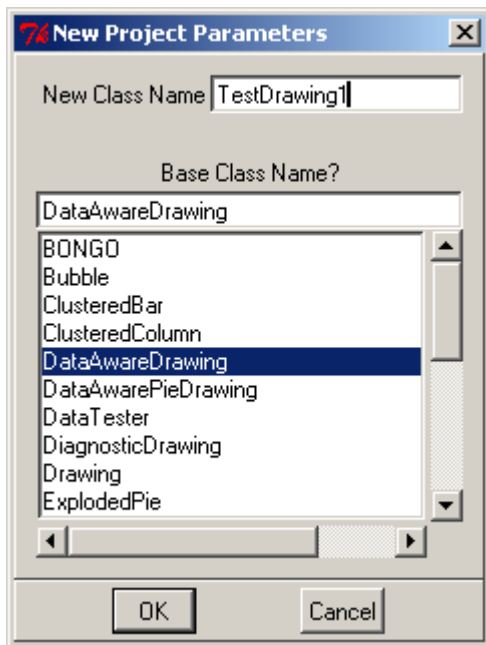
The interface also shows a schema view on the right side, listing various tables and their columns.

4.3. Create a basic chart module

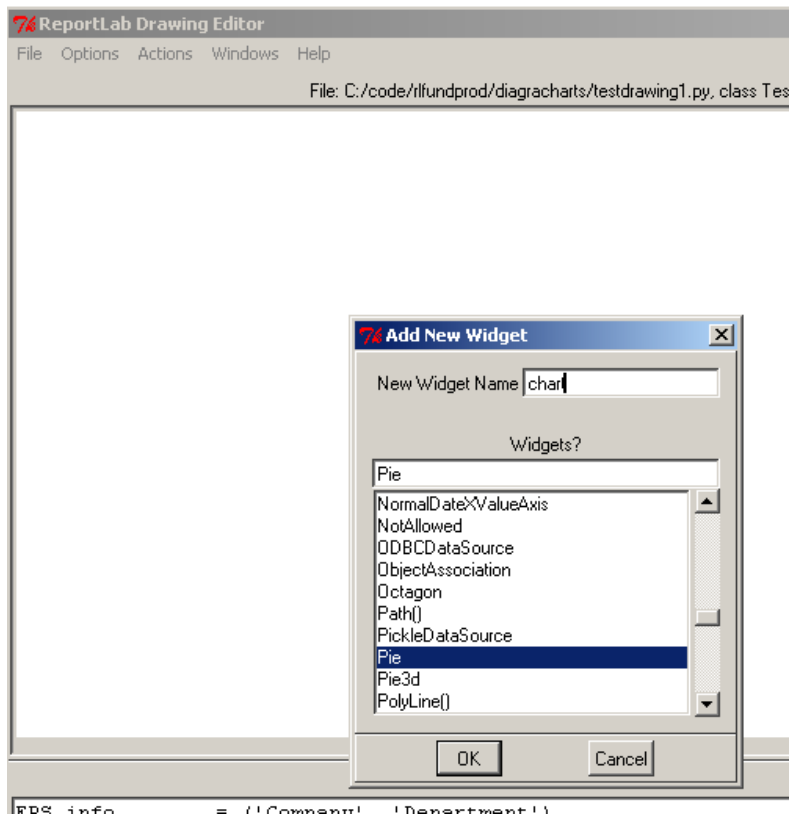
Now start to the Diagra Drawing Editor and create a new DataAwareDrawing.

1. Do **File | New** to bring up the dialog below.
2. Select 'DataAwareDrawing' in the list box – this is the basis for all charts which connect to a database and run to create lots of output charts
3. enter a class name in the New Class Name box at the top. We suggest using the name **TestDrawing1**

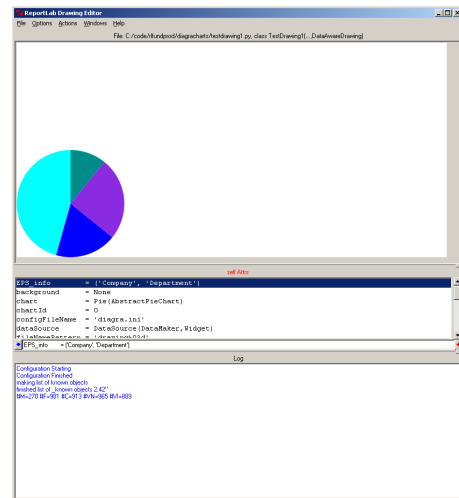
Enter a suitable name for the chart class. This should be in 'CamelCase' – capitals for each new idea in the name. If learning, TestDrawing1, TestDrawing2 etc. are normal. If doing a real chart, pick a name which will make sense in a year's time e.g. "BasicFundIndexComparison" or "HoldingsPieWithLegend". It's a good idea to make names which combine the rough chart type ('Pie') with the intended purpose ('Holdings')



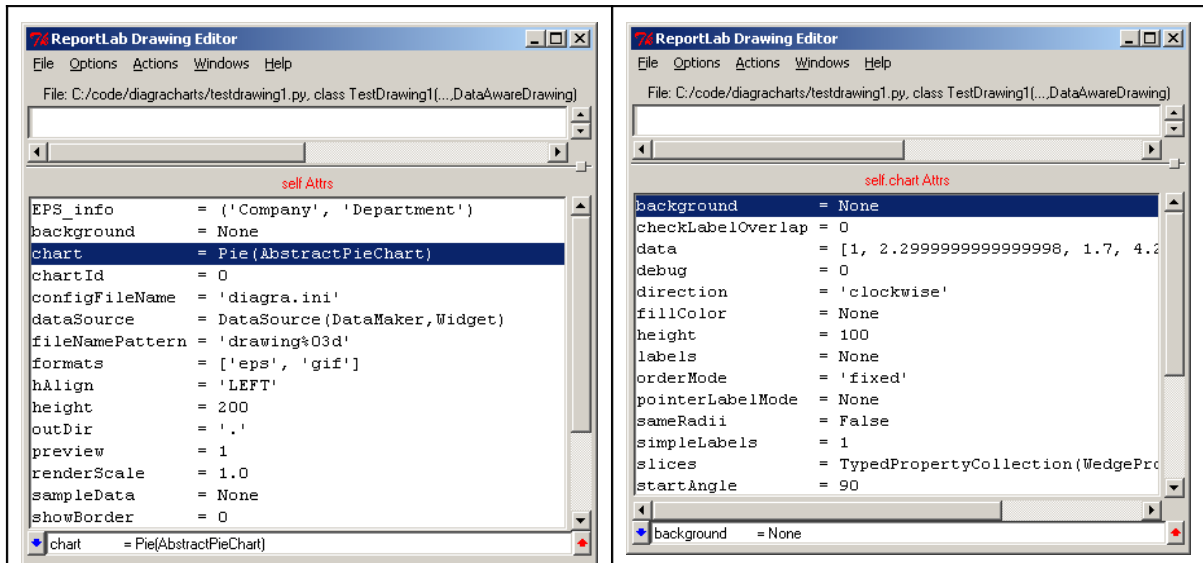
4. Click 'OK'. You now see a blank drawing.
(you might want to resize the windows and or do 'Windows | Zoom Set' to get the whole white drawing area on your screen).
5. Save the chart with **File | Save** or **File | Save As**. The module name can be any combination of letters or numbers, but a good convention is to use the class name already given without capitals or spaces. We'll use '**testdrawing1.py**' today. Put it somewhere you won't lose it on your disk.
6. Add a Pie to the drawing with "Actions | Add Widget". Give it the name "chart" and click OK. *It's very important to assign a name to each widget, which is short and easy to remember. By convention we use 'chart', 'legend', 'title', 'subtitle' and 'footnote' as widget names unless there is a good reason not to.*



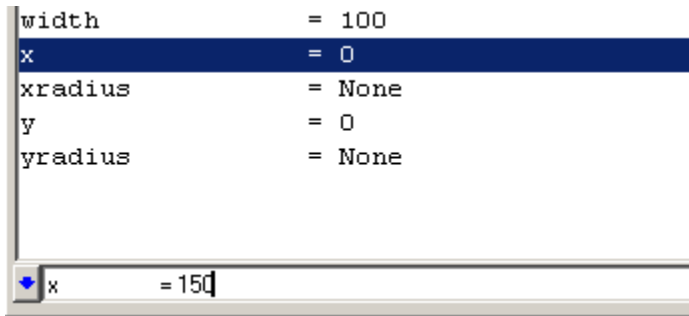
You should then see something like this:



7. We will now resize and position the chart by setting a few basic properties. Our drawing is 400 x 200 points, and we want the pie roughly centred with room for labels around the edges. To change the first property, find the 'chart' object in the properties window (3rd item down), and double-click it. This will "drill down" from the **drawing** properties into the properties of the **"chart"** object, which is a pie chart.

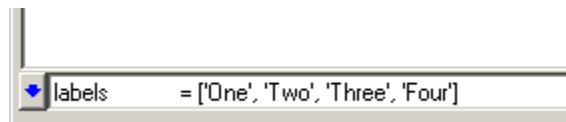


Scroll down until the 'x' property is visible, and select it with the mouse. Then edit the line below the properties, which displays the selected property, to say 'x=150', and hit 'Enter', as below:

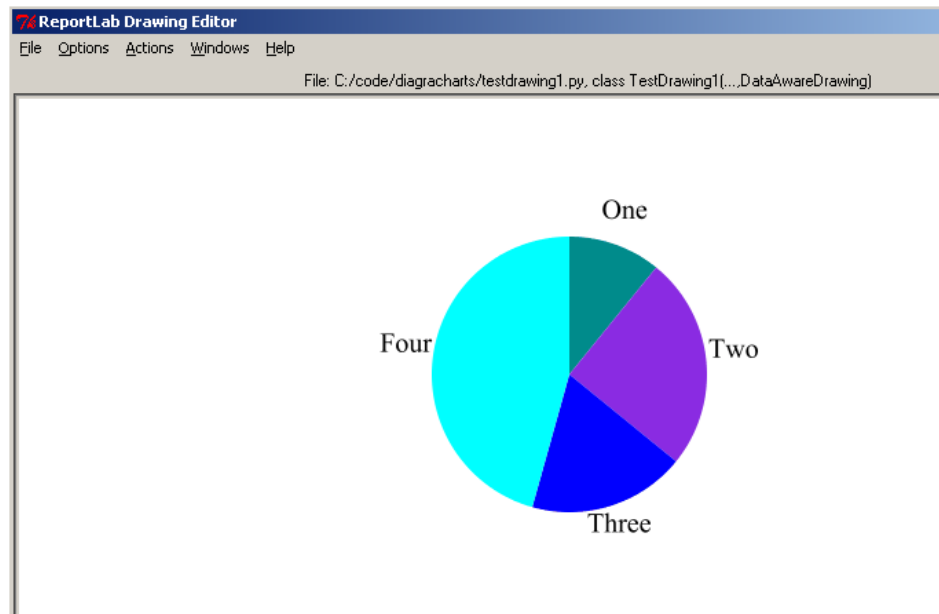


The chart should jump across towards the middle of the drawing.

8. Do as above setting $y = 50$, to move the chart up a little. It should now be roughly centered in the 400x200 drawing.
9. Give it some labels by setting the labels property of the chart as follows (*take care of the brackets and quotes; you can use single or double quotes, and straight or curved brackets, but they must match*):



If you've followed this far, your chart will look like this:

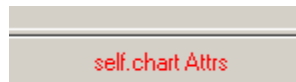


10. Now save your work with **File | Save**

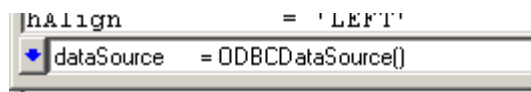
4.4. Connect to the database

Now we will connect up to our database.

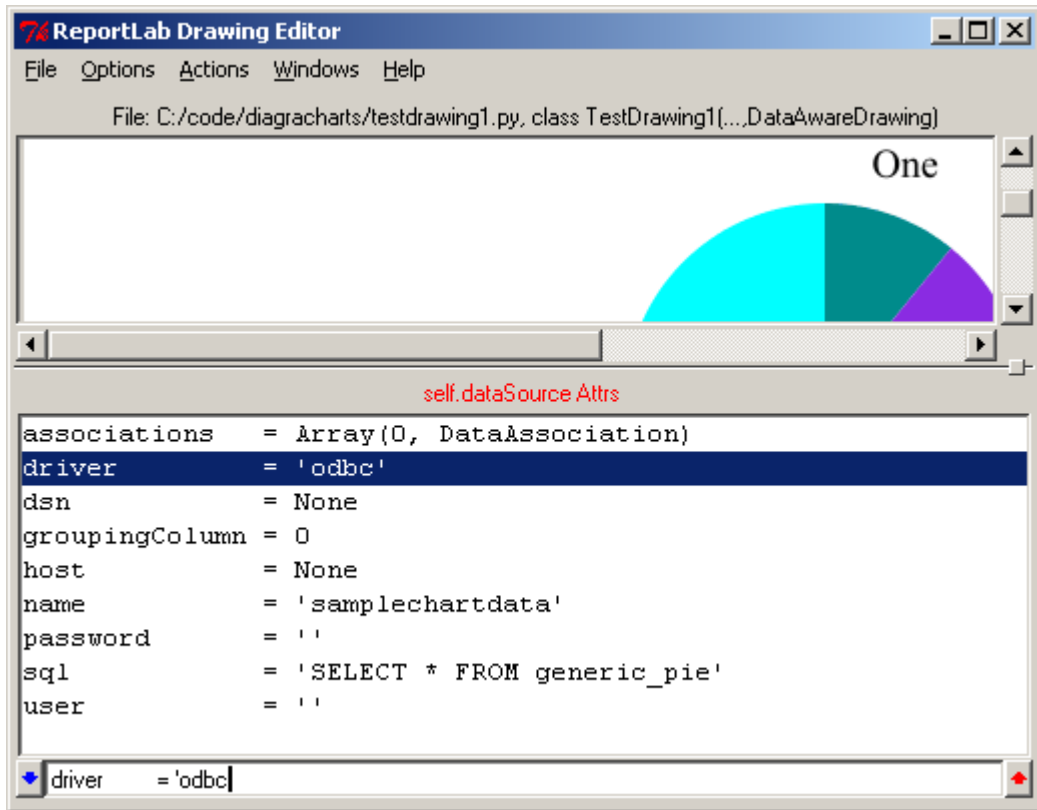
11. Move your mouse over the words below in the drawing, and click; you will move 'back up' from editing the chart within the drawing, to editing the drawing. The text will change from 'self.chart Attrs' to 'self Attrs'.



12. Edit the dataSource property of the Drawing to say 'ODBCDataSource()' exactly like this: *this tells the system that your data will come from a relational database. For historical reasons it's called an ODBCDataSource, even when using a non-ODBC mysql driver.*



13. Now double-click the ODBCDataSource in the properties window. It will 'drill down' into the data source. The properties look like this:



14. We will now set this up to connect to our sample database. Set the following properties:

Driver	Mysql
Host	'localhost' or leave as None <i>If using a remote server, enter the host name</i>
User	'root' if running a local server, or the user name
Password	Leave empty if running a server with no password, or enter the password
Name	'fundrep' if that's how the sample database was named. If you're using ODBC to connect to SQL server, you need to append your username and password; eg 'mysqlserverdsn/user/pw'
Sql	Your sql statement, copied from the query browser above.

15. Now we will test the connection. Double-click 'self.dataSource Attrs' to go back up to the Drawing level. Then set 'test' from 0 to 1. If it accepts this, your system has made a successful connection. You will also see a property appearing called 'sampleData' which was derived from the database query results, as shown below 2 lines above 'test':

```
self Attrs
height      = 200
outDir      = '.'
preview     = 1
renderScale = 1.0
sampleData  = [(1L, 'BP', 9.3000000000000007), (1L, 'HSBC Holdings', 6.0999999999999996),
test        = 1
transform   = (1, 0, 0, 1, 0, 0)
vAlign     = 'BOTTOM'
```

Let's just summarise the parameters we want in the datasource for this example with one more screen shot: *Note that my development server has password 'branch' -yours may not need a password or may differ.*

The screenshot shows the ReportLab Drawing Editor interface. The main window displays a pie chart with four segments labeled 'One', 'Two', 'Three', and 'Four'. The bottom panel shows the 'self.dataSource Attrs' configuration:

```
self.dataSource Attrs
associations = Array(0, DataAssociation)
driver       = 'mysql'
dsn          = None
groupingColumn = 0
host         = None
name         = 'fundrep'
password     = 'branch'
sql          = 'SELECT fund_id, label, fundPercent FROM fundrep_holdingbreakdowns h ORDER 1'
user        = 'root'
```

4.5. Making data associations

Now we have connected, we want to fetch something useful. We now need to create associations between the data set for the query and the chart's various elements. This is thoroughly documented in the data-aware charts chapter in the manual, so here we will just step through. We want to make three 'associations':

- One to get the fund ID into the chart, which is used to distinguish the filenames
- One to get the holding names which go around the edge
- One to get the numeric values for the plot.

Let's remind ourselves of what the data set looks like. The shot below shows the rows for the first plot in our sequence, with three columns. These will be referred to as columns 0, 1 and 2.

fund_id	label	fundPercent
1	BP	9.30
1	HSBC Holdings	6.10
1	Vodafone	5.84
1	Royal Bank of Scotland	3.86
1	Barclays	3.53
1	Shell Transport & Trading	3.37
1	Aviva	3.10
1	British American Tobacco	2.93
1	International Power	2.66
1	Gallaher	2.60

1. drill down into the 'associations' property of the dataSource
2. set 'size=3' and hit Enter. Some editable associations should appear like this:

```
self.dataSource.associations Attrs
element00 = DataAssociation(column=0, target=None, assocType='vector')
element01 = DataAssociation(column=0, target=None, assocType='vector')
element02 = DataAssociation(column=0, target=None, assocType='vector')
size      = 3
```

3. edit the first one to look like this:

```
self.dataSource.associations Attrs
element00 = DataAssociation(column=0, target='chartId', assocType='scalar')
element01 = DataAssociation(column=0, target=None, assocType='vector')
element02 = DataAssociation(column=0, target=None, assocType='vector')
size      = 3
```

This will have no visible effect just yet, but is important to get the output files all written with different names. Otherwise, you might generate 100 charts all with filename 'chart000.eps' on top of each other. After doing this each chart will take its ID from the **fund_id** in the database, and use this within the filenames.

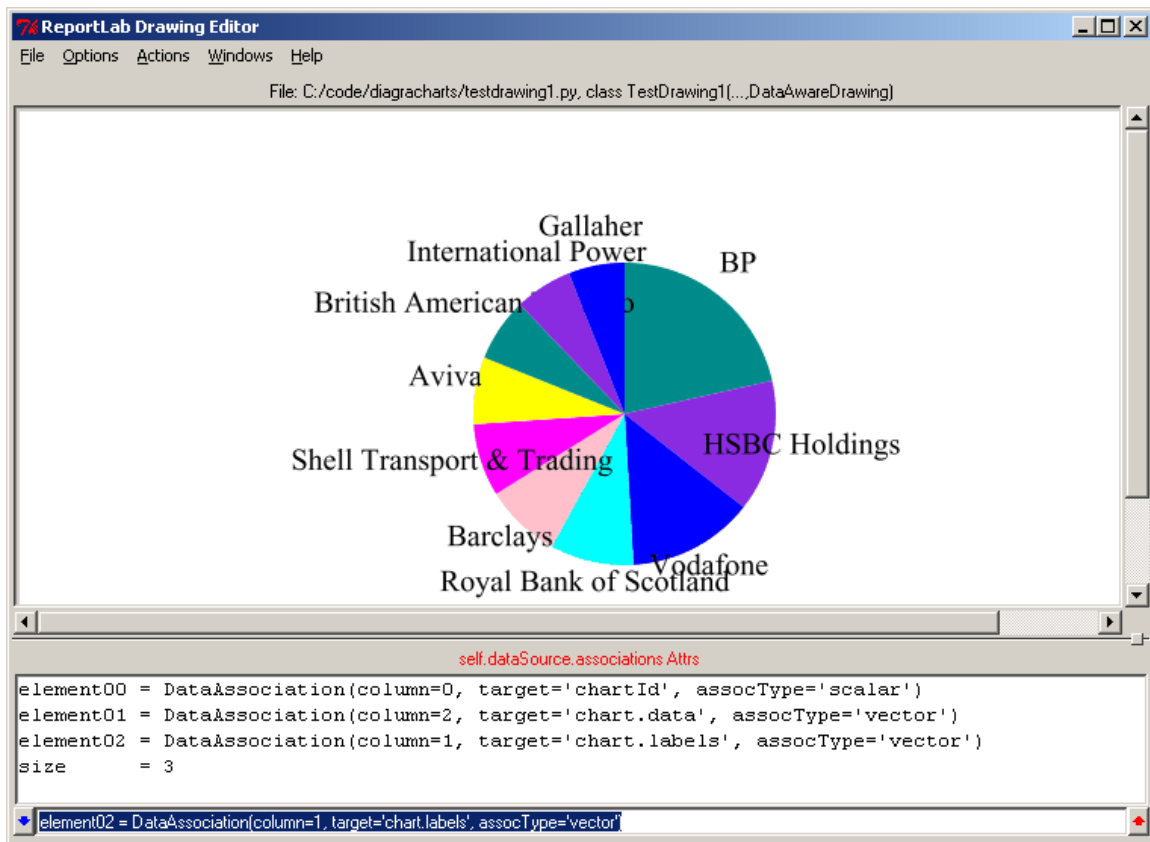
4. edit the second association to pull in the numbers, like this:

element01 = DataAssociation(column=2, target='chart.data', assocType='vector')

5. edit the third to bring in the labels, like this:

element02 = DataAssociation(column=1, target='chart.labels', assocType='vector')

6. Go back to the drawing top level, turn 'test' to zero if already on 1, then set 'test' to 1. This will now show the data for the first chart!



7. Save your work quickly and celebrate!

4.6. Planning output file names and formats

To be completed. In brief, set the outDir to 'output' and create it, and the 'fileNamePattern' to 'testdrawing1_%03d' then go to terminal and run the chart module as a command.

5. Creating a fund/index comparison chart

This example is slightly more complex because there are multiple series of data.

5.1. Planning:

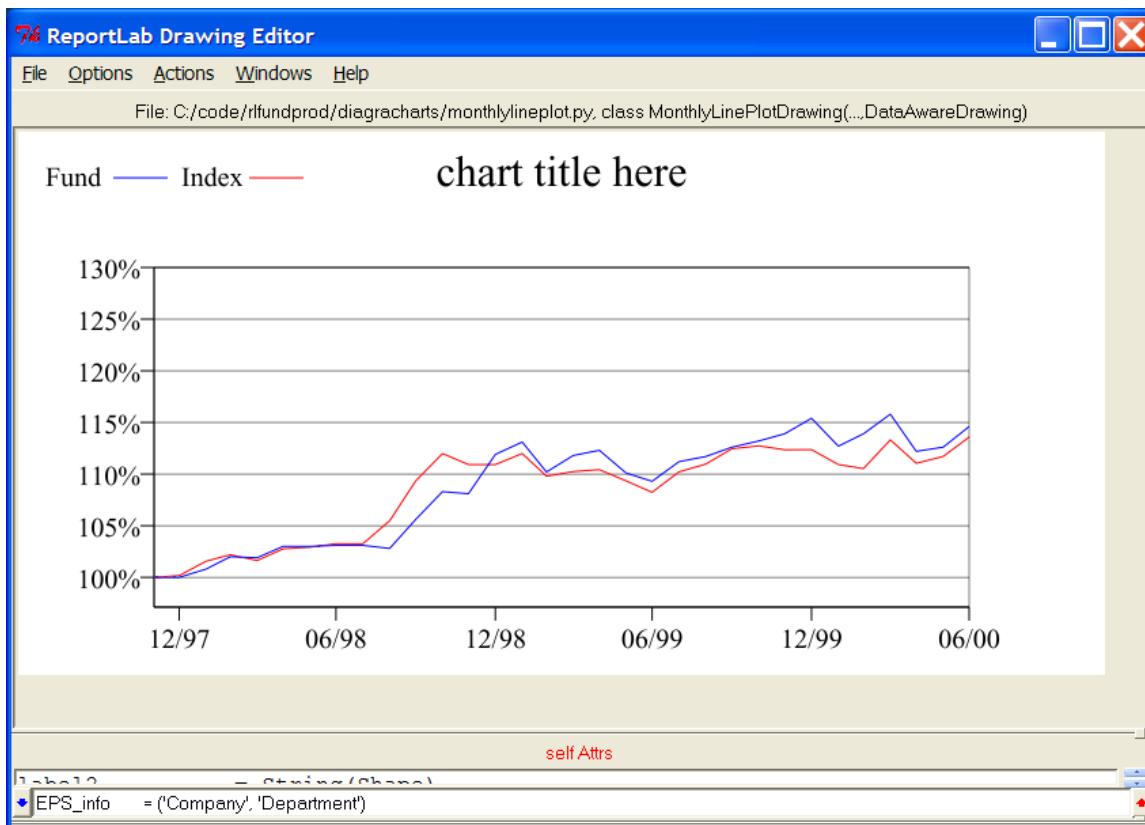
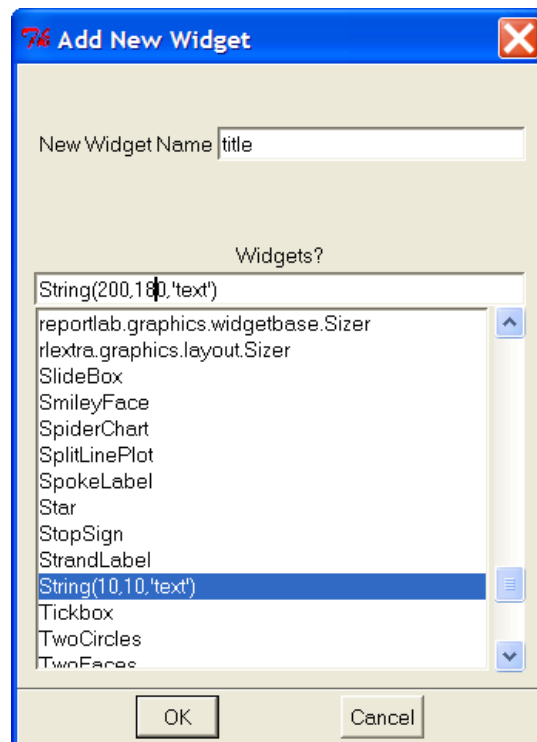
5.2. Laying out the chart

We'll go a little faster this time, not documenting every click. Refer back to the pie for screen shots if not sure what to do

1. File – New Drawing. Enter New Class Name = "MonthlyLinePlotDrawing", Base Class Name = "DataAwareDrawing". Click OK and save the module as "monthlylineplot.py"
2. Add a widget called "plot" of type "**GridLinePlot**" with Actions | Add Widget. *GridLinePlot is the basis for time series charts; it has a date axis and many nice details to do with date and value axis scaling.*
3. Double-click into the plot and set x=50, y=50, width=300, height=125
4. Add a text widget for the title. *Make sure you double-click the 'self Attrs' in red to backtrack to the top level of the drawing, or you'll end up adding the title to the plot, not the drawing, and it will not be visible.* There is a trick here to save time: you

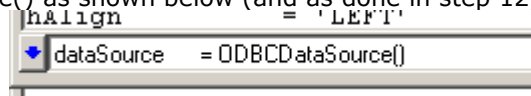
create a widget called 'title' of type String, but edit the position and text in the widgets dialog before adding it, so it starts in about the right place – see screenshot below.

5. Once it appears in the drawing, drill into it and set the font size to be, e.g., 16 points, and set **textAnchor = 'middle'**
6. Now we'll do the legend. Since we know in advance that there will be just two series and the line colors, widths etc. will not vary, we will simply create two swatches of colour and two text labels. Add two more strings called 'label1' and 'label2' with text 'Fund' and 'Index' at coordinates (20,180) and (60,180) respectively.
7. Now add a Line object called **"swatch1"** at coordinates (35,183,55,183). Drill down into it and set **strokeColor = blue** then step back up.
8. Repeat with another called "swatch2" at (85,183,105,83),
9. You should now have a chart looking like this:



5.3. Making data associations and testing

16. We will now set this up to connect to our sample database. Change 'dataSource' to 'ODBCDataSource()' as shown below (and as done in step 12, previous lesson):



17. Drill into the data source and set the following properties (the first 4 are the same as our previous lesson):

Driver	mysql
Host	'localhost' or leave as None <i>If using a remote server, enter the host name</i>
User	'root' if running a local server, or the user name
Password	Leave empty if running a server with no password, or enter the password
Name	'fundrep' if that's how the sample database was named
Sql	SELECT fund_id, periodEnd, fundReturn, indexReturn FROM fundrep_monthlyreturnseries ORDER BY fund_id, periodEnd

18. at top level set `outdir = 'output', filenamePattern = 'monthlylineplot%03d', formats = ['pdf','eps','png']` so that you get distinct named charts when running
19. Create two data associations by setting **dataSource.associations.size = 2**
20. Set the first association as
element00 = DataAssociation(column=0, target='chartId', assocType='scalar')
21. Set the second association (very carefully) as
element01 = DataAssociation(column=[(1, 2), (1, 3)], target='plot.data', assocType='tmatrix')
This can also be done by 'drilling down' and doing one property at a time. *What's happening here is a bit more complex. This plot type takes a series of lines as its drawing instructions – one for fund, one for index, and there could be more. Each line is an array of (date, value) pairs. So the bracketed-expression in 'column' says: compose the first series from column 1 (date) and column 2 (fund value); compose the second series from column 1 (date) and column 3 (index value); then put both together into the chart's data. If you had another column for, say, sector average, you might have **column=[(1, 2), (1, 3), (1, 4)]***
22. Go to the top level of the drawing and set `test = 1` to verify it connects. If it accepts the '1', and the chart updates, then the query worked. *Note that you can't do this until you have created the first association with the chartId, as it won't know where the first chart's data set begins and ends.*
23. Save the chart.
24. Go to a DOS or Unix prompt and execute the script e.g.
C:\code\rlfundprod\diagcharts> monthlylineplot.py
generating gif file output\monthlylineplot001.gif
generating EPS file output\monthlylineplot001.eps
...etc
generating gif file output\monthlylineplot024.gif
generating EPS file output\monthlylineplot024.eps

You should see output. *If it appears to 'hang' for a second, look in the directory; it may be creating charts saliently; then type "verbose = 1" in the properties editor and save and repeat.*

6. Creating a text table

Diagra can also be used (with some brute force effort) to automate complex tables. There are many "candidates for reuse" in financial publishing which consist of tables and need to be re-published with variable data each month. These can be set up as Drawings, even though no charts are involved, and automated to produce EPS files. As with the charts, this can save a lot of time and effort each month.

This was not one of our original design goals, and it is not very elegant in the present version. You might have to position a dozen lines and boxes by hand and fifteen text fields by hand. However, it is simple to connect these to a database.

Here we will show you how to create a table containing some data pulled out from MySQL database tables. In our next version of Diagra, we will be implementing tables objects as widgets similar to the previous widgets you have used earlier. In the meantime, this will be a somewhat prolonged exercise that will help you to familiarize yourself with the program and practice your data association skills.

6.1. Planning your work

As mentioned in the fourth chapter's beginning, you should thoroughly think about your objectives before starting any charts. This is particularly important for tables, as you have to position all the elements by hand, and want to get it right first time.

In this example, the objective is to create a **Discrete Years Table**, something the UK Financial Services Authority now requires in many publications. The principle is the same for other similar data tables. Here is an example of the chart we will create:

	2001-04-01 2002-03-31	2002-04-01 2003-03-31	2003-04-01 2004-03-31	2004-04-01 2005-03-31	30.04.04 30.04.05
2000-04-01 2001-03-31	-5.39	-6.56	-18.29	7.33	10.91

It is important to plan this carefully up front, using some other drawing tool to get your ideas right, as it is tedious to reposition all the elements multiple times in Diagra. *In the next version we intend to add support for tables, and perhaps even to click and rearrange things with a mouse, which will reduce the workload considerably.*

What you need to work out up front is (a) the overall size you want, and (b) the x and y coordinates in points of the different horizontal and vertical lines. The above drawing needs 7 x coordinates and 3 y coordinates.

6.2. Table Design

Now let's plan the data. We will need a fund ID of some kind; 5 columns for the values; 5 columns for the start dates of each financial year; and five columns for the end dates of each financial year. Let's plan ahead and add the fund name as well, just in case we get a request one day to add the fund name inside the graphic (and to help us tell which data came from which fund).

In the sample database there is a table **fundrep_discreteyears** that has the ideal data required for the this instance. On your MySQL Query Browser, double-click the '**fundrep**' database then double click the table '**fundrep_discreteyears**' which should generate an

SQL statement (SELECT * FROM fundrep_discreteyears f) for you in the editor. Just execute it as you did before.

The result of that execution, if everything was fine, should retrieve all the data from tables and normally look like the screen shot below.

	fundCode	fundName	fy5	fy4	fy3	fy2	fy1	start
1	HF001	Hyperfund Cash & Property Fund Acc	-5.39	-6.56	-18.29	7.33	10.91	
2	HF002	Hyperfund Global Opportunities Fund	-22.19	-13.62	-14.92	19.84	19.82	
3	HF003	Hyperfund Corporate Bond Fund	6.68	5.86	10.07	-2.05	9.03	
4	HF004	Hyperfund Managed Assets Fund I	22.32	3.02	-9.26	18.49	15.52	
5	HF007	Hyperfund Asia Pacific Portfolio Acc	-14.24	-27.89	-26.34	7.24	9.26	
6	HF008	Hyperfund UK Special Situations Fund	-7.17	-24.82	-21.38	20.41	7.70	
7	HF008	Hyperfund UK Smaller Companies Fund	3.68	-23.38	-20.08	39.65	18.58	
8	HF009	Hyperfund Financial Opportunities Fund	44.00	2.92	-4.05	21.31	22.66	
9	HF010	Hyperfund European Frontiers Fund R	16.89	1.57	-14.45	19.09	18.99	
10	HF011	Hyperfund UK Growth Fund	9.93	-19.58	-19.43	20.30	16.89	
11	HF012	Hyperfund High Income Fund	17.83	0.39	-8.25	13.00	16.87	
12	HF013	Hyperfund Transatlantic Fund R	2.96	-17.38	-21.25	30.33	20.75	
13	HF014	Hyperfund European Special Situations Fund	-5.33	-4.80	-16.89	22.16	27.64	
14	HF016	Hyperfund UK Mid 250 Fund R	9.34	-19.37	-36.06	19.15	17.35	
15	HF017	Hyperfund Far Eastern Fund	-8.96	-11.15	-28.17	23.93	10.32	
16	HF018	Hyperfund European Fund	-12.73	-5.55	-11.68	22.35	18.86	
17	HF019	Hyperfund Ethic Fund Acc	6.96	-12.16	-23.54	20.32	20.37	
18	HF020	Hyperfund Monthly Income Acc	-1.82	-31.08	-26.10	26.65	34.10	
19	HF021	Hyperfund Global Technology	-42.38	-36.61	-25.42	8.70	0.10	
20	HF022	Hyperfund Target Return Fund AM	0.00	0.00	2.27	3.44	12.54	
21	HF024	Hyperfund Emerging European Opportunities	0.00	0.00	0.00	38.61	33.73	

Here is an SQL column definition for that table:

```
DROP TABLE IF EXISTS `fundrep_discreteyears`;
CREATE TABLE `fundrep_discreteyears` (
  `id` mediumint(9) unsigned NOT NULL auto_increment,
  `fundCode` varchar(20) NOT NULL,
  `fundName` varchar(100) NOT NULL,
  `fy5` decimal(11,2) default NULL,
  `fy4` decimal(11,2) default NULL,
  `fy3` decimal(11,2) default NULL,
  `fy2` decimal(11,2) default NULL,
  `fy1` decimal(11,2) default NULL,
  `start5` varchar(20) default NULL,
  `start4` varchar(20) default NULL,
  `start3` varchar(20) default NULL,
  `start2` varchar(20) default NULL,
  `start1` varchar(20) default NULL,
  `end5` varchar(20) default NULL,
  `end4` varchar(20) default NULL,
  `end3` varchar(20) default NULL,
  `end2` varchar(20) default NULL,
  `end1` varchar(20) default NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note that we have declared all of the numeric and text fields as VARCHAR, i.e. as TEXT fields, not numbers and dates. This is because we want to display the dates and numbers exactly as given across all output media; it will save us having to fiddle around formatting numbers and dates in the drawing, and will avoid any rounding errors. It will also let you put in the text string you want (e.g. 'N/A', '-' or nothing) for

under open less than five years. This approach would be wrong in a performance-calculation database, but is often simplest in a reporting / publishing database.

In a real-world database, you might instead be deriving this from a query which joined together several tables, and using formulae in SQL to format the columns; but we'll start with it all in one table.

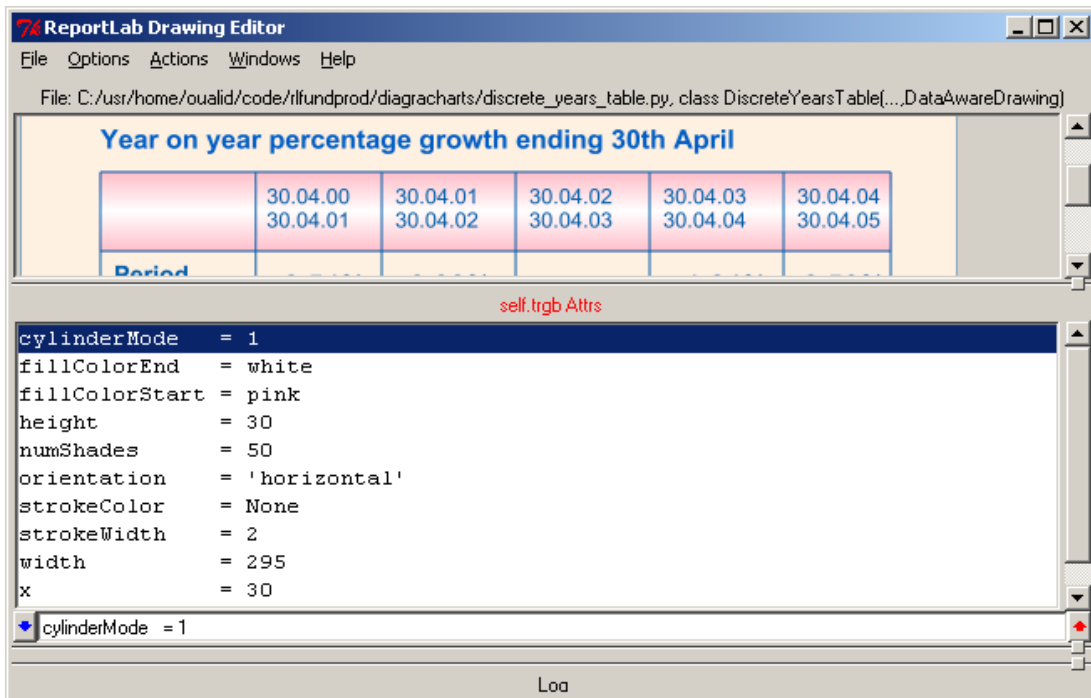
6.3. Create a chart module

See "4.3 Create a basic chart module" for more details.

6.4. Create the table

We do this by explicitly adding objects to the drawing using Actions | Add Widget. We'll leave this as an exercise but we added

1. 3 horizontal line objects called horline1, horline2, horline3
2. 7 vertical line objects called verline1..7
3. a String called 'title'
4. a ShadedRect widget called 'trgb' whose coordinates were set to fit the top half of the grid. ShadedRect has properties to do gradual shading between two colours, white and pink in this case; if you set cylinderMode = 1 then it will vary from pink to white and back again to create the effect below, which can also be used to create 3d-look bars in bar charts..



5. we created lots of String objects. We named these 'col1Data', 'col2Data' etc. for the strings to hold the numbers; 'col1LabelA' for the top date; and col1LabelB for the bottom date. Fifteen of these hold the data, and two more on the left for the
6. One thing we didn't do in this case, but which might well save a little time, is to use Rect objects instead of lines. If your lines get thick, a rectangle will avoid strange corner effects when one zooms in close to the chart.

6.5. Connect to mysql and associate with data

We will now connect to our MySQL database. The connection process is straightforward and does require you to input some parameters, as you've already done in the previous chapter five (5.3).

1. In the connection parameters, you should copy most of the data you have set for the properties, because they have not changed since. The only difference is to modify the `Sql` property as shown below in the table

Driver	Mysql
Host	'localhost' or leave as None <i>If using a remote server, enter the host name</i>
User	'root' if running a local server, or the user name
Password	Leave empty if running a server with no password, or enter the password
Name	'fundrep' if that's how the sample database was named
Sql	<code>SELECT * FROM fundrep_discretelyears</code>

2. And now go back to the parent chart object by double clicking on the red text `self.Attrs` in the editor.
3. Test that the connection is established by setting test equal to 1. (`test=1`).
4. Now we will make the data associations. If you have noticed in our sample table **fundrep_discretelyears** we have a table of 18 columns. In this example, we did not use the fundCode and therefore we will make 17 associations.
 - One to get the fund ID into the chart, which is used to distinguish the filenames
 - One to get the date data for the top date in `col1LabelA` that go in the first column for instance and so on...
 - One to get the date data for the bottom date in `col1LabelB` again that go in the first column and so on...
 - One to get the numeric values for the percentage data which go in the bottom table cell and so on...

5. So, drill down into the **'associations'** property of the `dataSource`

6. set `'size=17'` and hit Enter.

7. edit the first association like this

```
element00 = DataAssociation(column=0, target='chartId', assocType='scalar')
```

8. edit the third and up to the seventh association to bring the percentage values data as follows:

```
element02 = DataAssociation(column=3, target='col1Data.text', assocType='scalar')
```

```
element03 = DataAssociation(column=4, target='col2Data.text', assocType='scalar')
```

etc ...

9. now, edit the eighth up to the twelfth association to bring all dates that go in top date strings, like:

**element07 = DataAssociation(column=8, target='col1LabelA.text',
assocType='scalar')**

**element08 = DataAssociation(column=9, target='col2LabelA.text',
assocType='scalar')**

etc ...

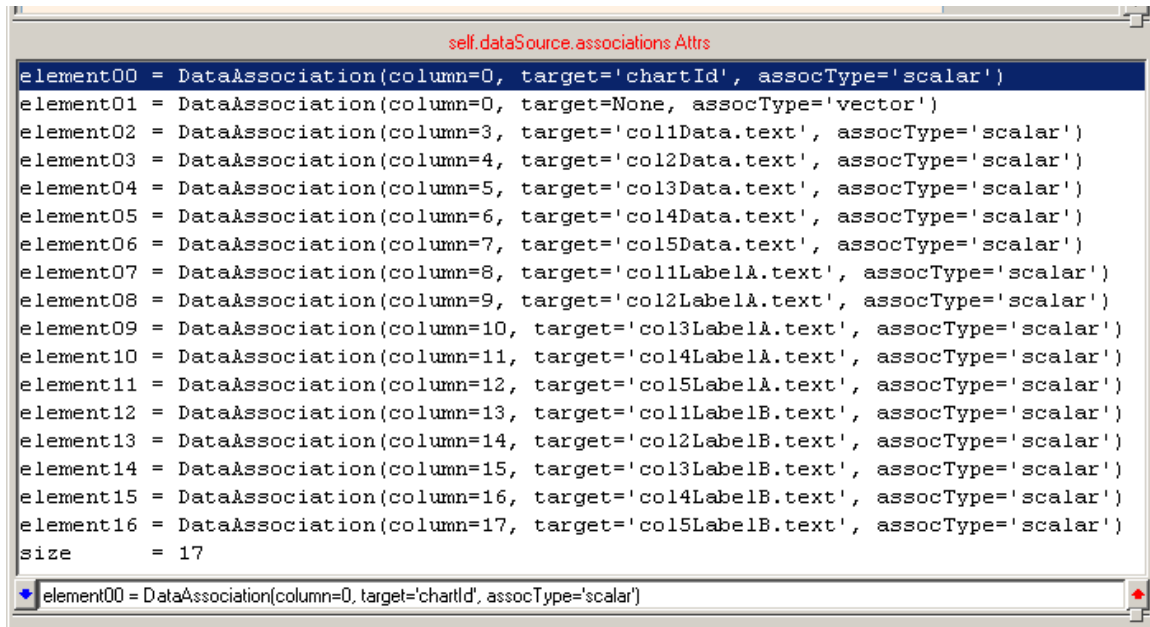
10. Finally, edit the thirteenth up to the seventeenth association to bring all dates that go in bottom date strings, like:

**Element12 = DataAssociation(column=13, target='col1LabelB.text',
assocType='scalar')**

**Element13 = DataAssociation(column=14, target='col2LabelB.text',
assocType='scalar')**

etc ...

here below is a screen shoot of what you should really ending up with. Notice that we did not use **element01** as we have reserved that association element for later in case we needed to include the fund names.



```
self.dataSource.associations Attrs
element00 = DataAssociation(column=0, target='chartId', assocType='scalar')
element01 = DataAssociation(column=0, target=None, assocType='vector')
element02 = DataAssociation(column=3, target='col1Data.text', assocType='scalar')
element03 = DataAssociation(column=4, target='col2Data.text', assocType='scalar')
element04 = DataAssociation(column=5, target='col3Data.text', assocType='scalar')
element05 = DataAssociation(column=6, target='col4Data.text', assocType='scalar')
element06 = DataAssociation(column=7, target='col5Data.text', assocType='scalar')
element07 = DataAssociation(column=8, target='col1LabelA.text', assocType='scalar')
element08 = DataAssociation(column=9, target='col2LabelA.text', assocType='scalar')
element09 = DataAssociation(column=10, target='col3LabelA.text', assocType='scalar')
element10 = DataAssociation(column=11, target='col4LabelA.text', assocType='scalar')
element11 = DataAssociation(column=12, target='col5LabelA.text', assocType='scalar')
element12 = DataAssociation(column=13, target='col1LabelB.text', assocType='scalar')
element13 = DataAssociation(column=14, target='col2LabelB.text', assocType='scalar')
element14 = DataAssociation(column=15, target='col3LabelB.text', assocType='scalar')
element15 = DataAssociation(column=16, target='col4LabelB.text', assocType='scalar')
element16 = DataAssociation(column=17, target='col5LabelB.text', assocType='scalar')
size = 17
```

11. now, go back to the parent object (double clicking on the red text) as you did in step 2.

12. then, set test=0 if it was previously set to 1, and set it back to 1 and hit Enter.

Well done!! You have created your first data aware chart table.

Also, make sure you have saved your work. Next, go to a terminal and run the chart module as a command to see the output.

7. Updating the database from data in Excel or files

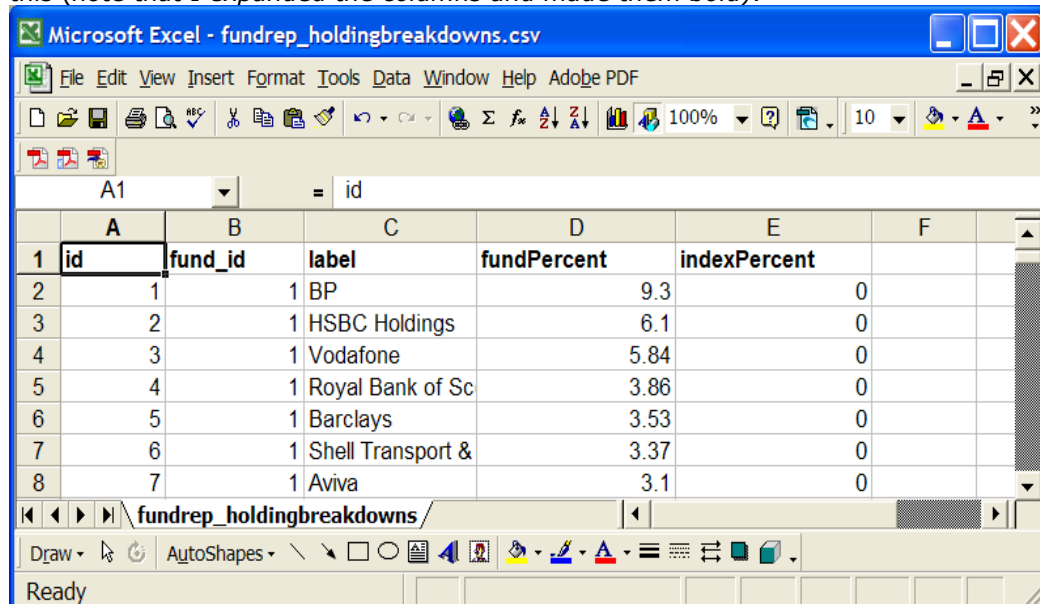
There are many ways to do this. One of the most versatile is the MySQL **load data** command. Most database vendors provide an equivalent. In addition, most database administration tools have an import or export tool which can control the various options.

7.1. Manual import of one data set

We will show you how to use the MySQL command from the Query Browser. One advantage of this is that it can be placed in a Unix or DOS script and fully automated. We'll presume you can prepare exactly the needed data set in a spreadsheet. We've shipped several example CSV files in the `diagracharts` directory for examples.

WARNING: this can get complex. The database server will give error messages about duplicate data, mismatched records in other tables if you have inter-table constraints, incorrect field types; and may misinterpret dates and other fields. If in doubt, ask a database developer; and always examine the imported data by eye afterwards!

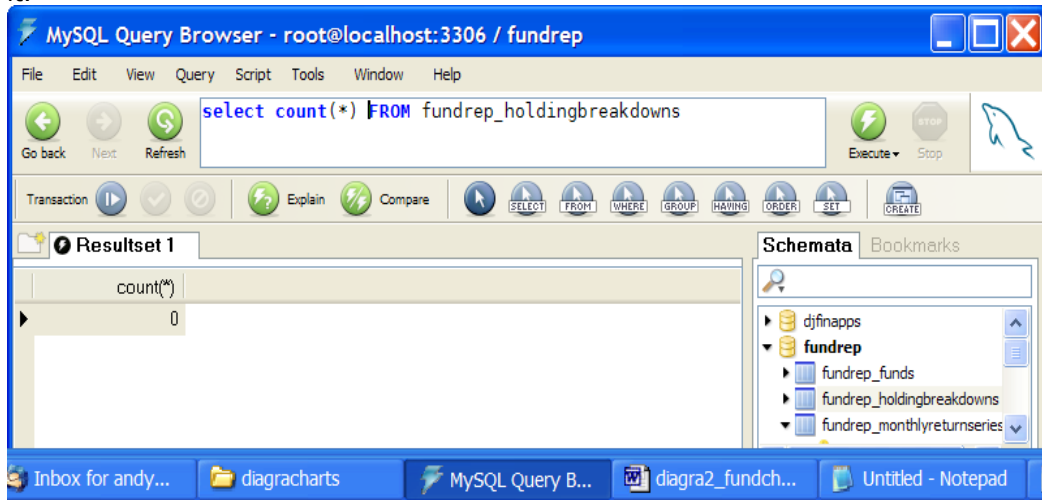
1. Start Excel
2. Open the file `fundrep_holdingbreakdowns.csv`, which should come up somewhat like this (note that I expanded the columns and made them bold):



	A	B	C	D	E	F
1	id	fund_id	label	fundPercent	indexPercent	
2	1	1	BP	9.3	0	
3	2	1	HSBC Holdings	6.1	0	
4	3	1	Vodafone	5.84	0	
5	4	1	Royal Bank of Sc	3.86	0	
6	5	1	Barclays	3.53	0	
7	6	1	Shell Transport &	3.37	0	
8	7	1	Aviva	3.1	0	

3. modify the data as you wish. The column structure must match the database table you wish to load it into, but the column names are not needed and don't have to match.
4. Choose "File | Save As". In the box saying "Save as type", choose "tab delimited" and call the file "myfile.txt". *We recommend tab delimited, because sometimes commas occur in text fields you might be loading, at which case things get more complicated. Tab delimited is tougher on the naked eye in a text editor, but much easier for the computer to parse, because you'll never want to have a tab character in, say, a fund objective.*
5. In the MySQL query browser, do a 'select * from tablename' command to reassure yourself that the column structure of the destination table really does match. Example: **"select * from fundrep_holdingbreakdowns"**.
6. If you have existing data in the table and want to delete it before reloading, execute a delete query. **Be very sure you have the right tablename!**
delete from fundrep_holdingbreakdowns

- To verify it's gone, you can ask for a record count with this query:
select count(*) from holdingbreakdowns. You should see a single cell with '0' in it.



- We now compose a LOAD DATA command to bring in the data. This command needs to be told (a) the full path to the filename we are loading from, (b) the table to go into, and (c) lots of options about delimiter types. This should work:

```
load data local infile 'c:/code/rlfundprod/diagracharts/myfile.txt'  
into table fundrep_holdingbreakdowns  
fields terminated by '\t'  
ignore 1 lines
```

The above says that the file is tab delimited, and the first row should be ignore. The Query Browser then prints about 200 warnings (one per row) about the precision of the data being truncated, presumably because the input data had a different number of decimal places to the target table; but works. (yellow icons in the pane at the bottom are warnings; red means errors which it cannot import).

- Do a further "select * from fundrep_holdingbreakdowns" to confirm your data is present

7.2. Automating imports

The above commands can be executed from DOS or Unix scripts. In addition, the **mysqlimport** command does similar things. If your input data is guaranteed to be consistent and correct, you can find a set of parameters to import it. The MySQL online manual has full help on the LOAD DATA command.

However, these commands are very picky and even experienced database frequently spend an hour scratching their heads trying to figure out exactly why data won't go in. Just remember that the database server's main job is only to allow in data which is exactly right!

In general, the usual "production" approach is to write scripts in languages like Python, Perl, Basic or Java to pump data in, which can apply many checks and can display meaningfully what is going wrong, and which can ensure the data is split off into the destination tables. ReportLab are happy to assist with this.

8. Windows and Mac OS X platform notes

To be completed – some Mac difference notes
