From Technologies to Solutions

iReport 3.7

Learn how to use iReport to create, design, format, and export reports

Shamsuddin Ahammad

PACKT PUBLISHING

iReport 3.7

Learn how to use iReport to create, design, format, and export reports

Shamsuddin Ahammad

PACKT PUBLISHING

BIRMINGHAM - MUMBAI

iReport 3.7

First published: March 2010

Production Reference: 1250210

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-847198-80-8

www.packtpub.com

Cover Image by Karl Swedberg (karl@englishrules.com)

Credits

Author
Shamsuddin Ahammad

Reviewer
Giulio Toffoli

Acquisition Editor
Douglas Paterson

Development Editor
Chaitanya Apte

Technical Editor
Siddhant Rai

Copy Editor
Leonard D'Silva

Indexer
Monica Ajmera Mehta

Editorial Team Leader
Mithun Sehgal

Project Team Leader
Lata Basantani

Project Coordinator
Srimoyee Ghoshal

Proofreader
Cathy Cumberlidge

Production Coordinator
Shantanu Zagade

Cover Work
Shantanu Zagade

About the Author

Shamsuddin Ahammad is a Senior Lecturer at Daffodil Institute of IT (DIIT), Bangladesh. He is the Course Coordinator of IT programs at DIIT. He has been teaching Java Programming, Database Systems, Systems Analysis and Design since 2002. He has experience in supervising hundreds of academic projects. Shamsuddin has a Masters degree in Management Information Systems from Daffodil International University, Dhaka. He obtained a BSc degree in Computing & Information Systems of NCC Education Ltd, UK and London Metropolitan University joint program from Daffodil Institute of IT. He is an additional article reviewer of the CONQUEST conference, Germany. He has been maintaining several blogs on Java technologies for years.

Extraordinary moral support of my respected parents and my loving wife Jesmin Rashid is the greatest inspiration to write this book.

My brother and sisters, my relatives, friends, and my colleagues at Daffodil Institute of IT have inspired me a lot in writing this book—special thanks to all of them. I'm very thankful to my friend Shahed Hasan for his warm support.

It is my great fortune to have worked with a great team of publishing professionals at Packt Publishing. My sincerest gratitude to Douglas Paterson, Srimoyee Ghoshal, Chaitanya Apte, and Siddhant Rai for their great cooperation in writing the book.

About the Reviewer

Giulio Toffoli is a Senior Software Engineer at JasperSoft Corporation, where he serves as the iReport project leader. He has been developing Java applications since 1999 and founded the iReport project in 2001. During this time, Giulio has enjoyed designing complex software architectures and implementing custom software solutions with a focus on desktop and multitiered, web-based, client-server applications using Java (J2EE/JEE) and open source technologies. Giulio has a degree in computer science from the University of Bologna and currently resides in Italy. Giulio has written several tutorials and articles about iReport and JasperSoft technologies. He is the author of The Definitive Guide to iReport, Vol. 2, edited by Apress, and co-author of JasperServer Ultimate Guide.

Table of Contents

Preface

iReport is an intuitive and easy-to-use visual report designer/builder for JasperReports, which is written in Java.

Users can visually edit complex reports with charts, images, and subreports, as iReport is integrated with the leading open source library—JasperReports. JasperReports is the world's most popular open source Java reporting library, but doesn't provide an adapted tool to visually design reports. iReport is a visual report designer built on JasperReports. A person without much confidence with XML might not be able take full advantage of the JasperReports library. iReport works perfectly for such people.

This book is a straightforward introduction to the iReport environment taking an example-oriented approach in developing your skills from scratch. It will guide you through developing a simple report to a dynamic enterprise level report using iReport.

This book is a beginner's tutorial, which shows you how to use iReport for creating reports in PDF, RTF, and so on, which can be sent over the web for immediate access.

It will guide you in developing various types of reports, using realistic examples based on a sample Inventory Management System. This book takes you through the main types of reports available in iReport, and shows you exactly how to create them. It shows you how to use different report templates, how to use special kinds of data operations to generate more powerful reports, combine data to produce master-detail reports, add images, control the layout and formatting of your report, and much more.

This book will also show you how to use NetBeans IDE for creating a Java project with reporting facilities. You will learn about report formatting and layout according to business requirements.

What this book covers

Chapter 1, *Introduction to iReport*, looks at what iReport is, what you can do with iReport, the history of iReport, and installing iReport.

Chapter 2, *Building your First Report*, covers creating a new data source or database JDBC connection, creating a report using the Report Wizard, using the design query to produce the SQL for the report, viewing the report using the built-in viewer, exporting the report in different formats like PDF, RTF, and so on.

Chapter 3, *Report Layout and Formatting*, covers configuring report properties, understanding bands, configuring and using report bands, maintaining the size, position and alignment of the report elements, modifying report fonts, using the library, using borders, and modifying element properties.

Chapter 4, *Using Variables*, discusses the use of variables, adding variables in a report, and writing variable expressions.

Chapter 5, *Using Parameters*, discusses the necessity of parameters, adding/modifying parameters, and modifying the SQL query for using the parameters.

Chapter 6, *Grouping Data in Reports,* discusses report groups and grouping data together.

Chapter 7, *Subreports*, covers what subreports are, creating subreports, compiling subreports, linking the main report to the subreport, and passing data between the main report and the subreport.

Chapter 8, *Crosstab Reports*, discusses what crosstab elements are and how to use crosstab elements in reports.

Chapter 9, *Charting,* covers how to create reports with pie charts, 3D pie charts, and bar charts.

Chapter 10, *Working with Images*, covers how to display images in reports from the database, how to display static images from the hard drive, and how to set background images in a report.

Chapter 11, *Calling Reports from Java Applications*, discusses the JasperReports library for calling iReport from your Java application.

Chapter 12, *iReport in NetBeans*, discusses installing iReport plugins in NetBeans and creating reports from within the NetBeans IDE.

Appendix, *A Sample Database*, explains a sample database design and the development of an inventory management system.

What you need for this book

The following software must be installed for this book:

- iReport 3.7.x

  `http://sourceforge.net/projects/ireport/files/`
- JDK 6

  `http://java.sun.com/javase/downloads/index.jsp`
- NetBeans

  `http://netbeans.org/downloads/index.html`
- MySQL 5

  `http://dev.mysql.com/downloads/mysql/5.5.html`
- MySQL GUI Tools

  `http://dev.mysql.com/downloads/gui-tools/`

Who this book is for

This book is for people new to iReport, business intelligence reporting tool users, and developers who have working experience in Java. This book shows you how to develop a simple report, reports with charts, and summary reports in iReport. Thus this book is a perfect choice for both the initial learners and the experts, who produce extensive reports for business applications.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "If you want the instance to be on top of other internal frames, call the `setSelected` method by entering `Boolean true` as the argument."

A block of code is set as follows:

```
SELECT * FROM Sales,Customer
WHERE Sales.customerNo=Customer.customerNo
AND Sales.salesNo = $P{salesNo}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
SELECT name,sum(salesQuantity), sum(unitSalesPrice*salesQuantity)
FROM Product, Sales, SalesLine
WHERE Product.productCode=SalesLine.productCode
AND Sales.salesNo=SalesLine.salesNo
AND salesDate BETWEEN $P{startDate} AND $P{endDate}
GROUP BY name
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "If the **Verify Certificate** dialog box appears, click **Continue**".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book on, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code for the book

Visit `http://www.packtpub.com/files/code/8808_Code.zip` to directly download the example code.

The downloadable files contain instructions on how to use them.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **let us know** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Introduction to iReport

iReport is a reporting tool, developed in Java that helps users and developers design reports visually. Through a rich and simple-to-use user interface, iReport provides the most important functions to create complex reports easily, thus saving a lot of time.

iReport uses the JasperReports library inherently to create reports. JasperReports is, in a sense, the core of iReport. JasperReports is the most popular open source reporting library for Java technology, and iReport is a visual report designer for JasperReports. JasperReports has hundreds of features, but it itself doesn't provide any tool to visually design reports. A person without much confidence with XML might not be able take full advantage of the JasperReports library. iReport works for such people as well as for the expert report developers.

A report produced in iReport can be integrated in your open source or commercial application to generate reports, display them on screen, or export them in several formats including PDF, OpenOffice, DOCX, and many others. Alternatively, you can transfer the result through a web application or send the final document directly to a printer.

It is extremely easy to integrate JasperReports in any Java application using iReport. However, if you need an environment to use the reports without having to write a custom application, you may consider using JasperServer. JasperServer provides:

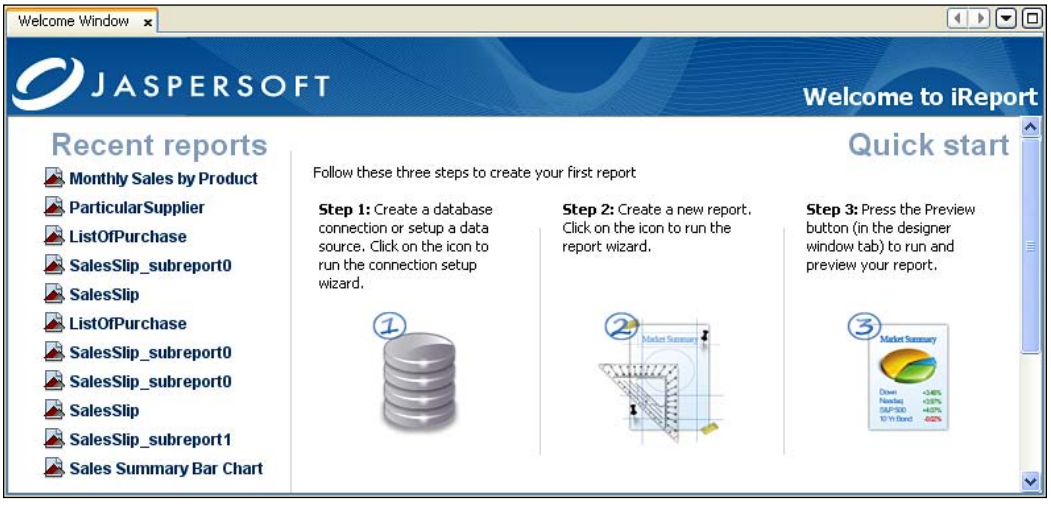
- A web-based interface to manage, schedule, and run the reports
- A repository to store all the report resources, such as images, fonts, data sources, and much more
- A security service to decide who can execute which report
- A web services API to execute the reports from external applications

iReport features

iReport can connect to any database and acquire data from different data sources. It can export the reports to PDF, XHTML, OpenOffice, MS Word, MS Excel, XML, Text, and many more. To understand what you can do with iReport, let's have an overview of the features.

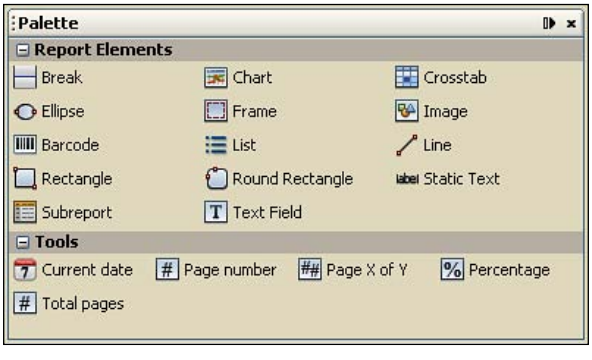
Simple to use GUI

The User Interface (UI) of iReport is very user friendly. It provides some easy steps to generate reports using the Report Wizard.

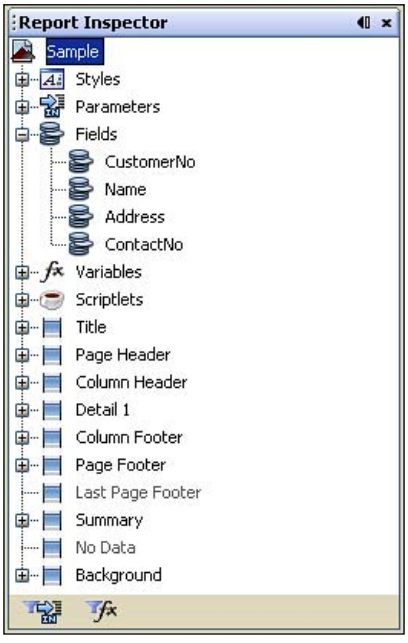


The UI features of iReport include:

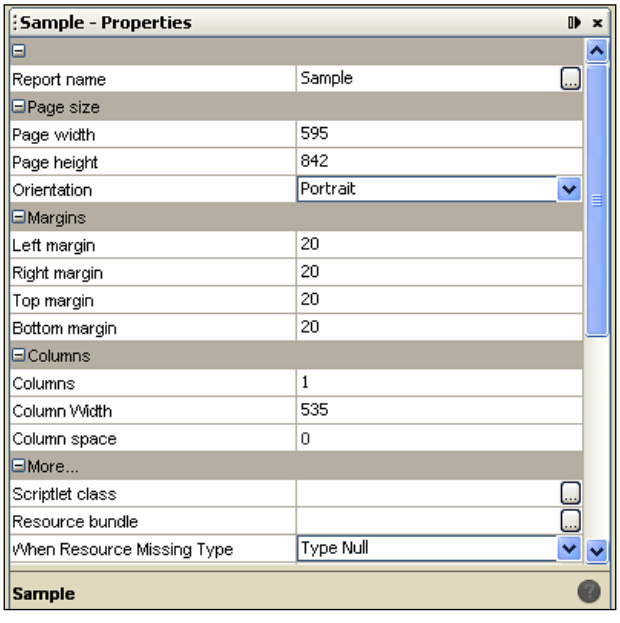
- Drag-and-drop facilities for report elements
- The **Palette** window consisting of the **Report Elements** and **Tools**



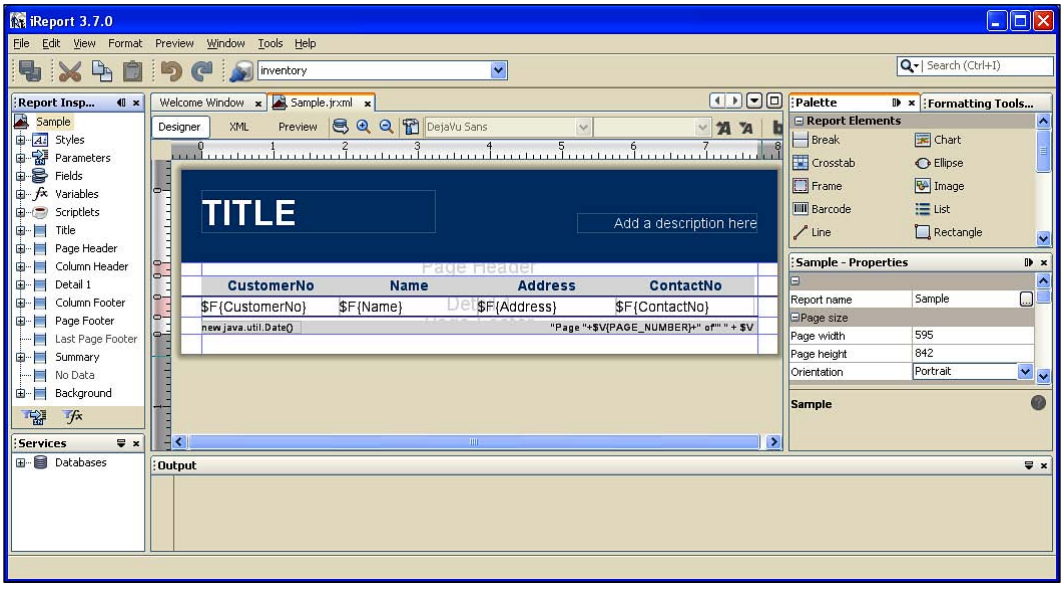
- Easy navigation of report objects through the **Report Inspector**



- The **Properties** window for setting properties of report elements, as shown next:



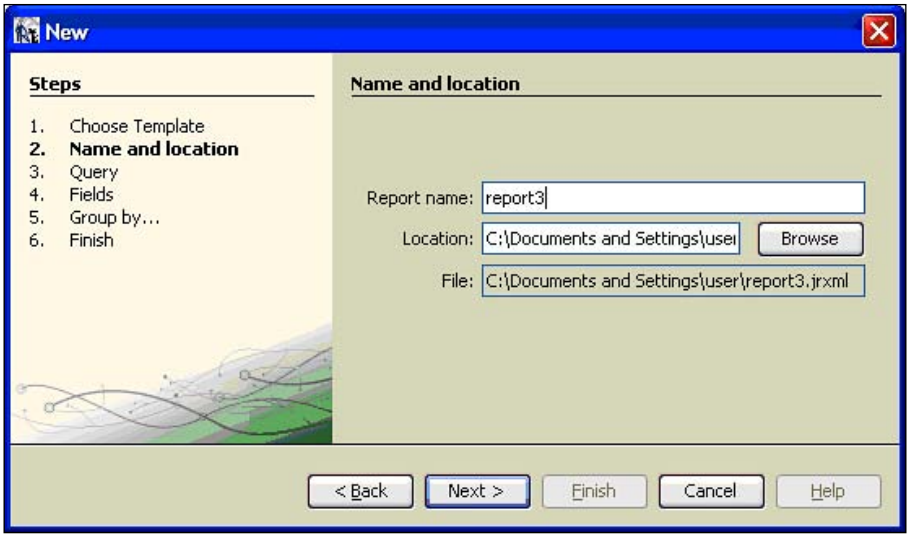
- **Undo** and **Redo** support
- Dockable and configurable interface window



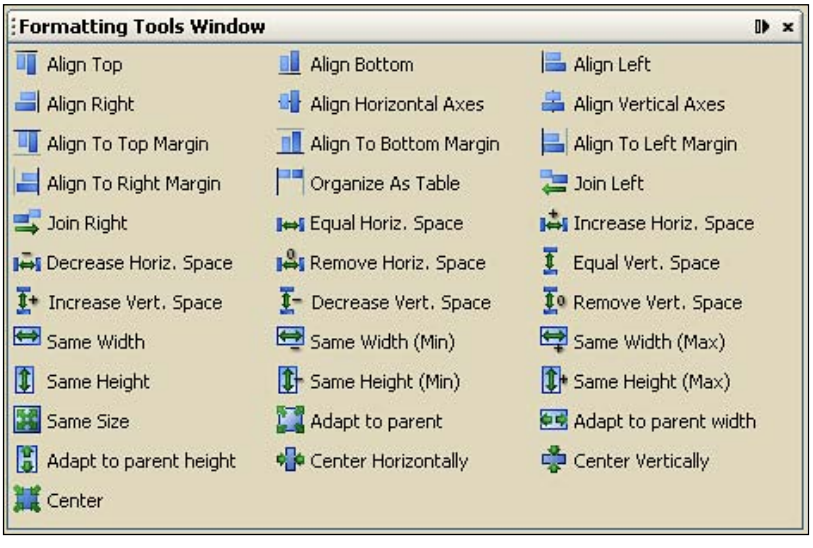
Report designer and tools

You can design your report using a smart report designer. The following are some of the features of the designer:

- Using the Report Wizard, the report can be built in some easy steps within a very short time. After that, there are many other editors that can be used to modify/add more report features. It helps to quickly create new reports, subreports, define group data, and create group headers and footers.

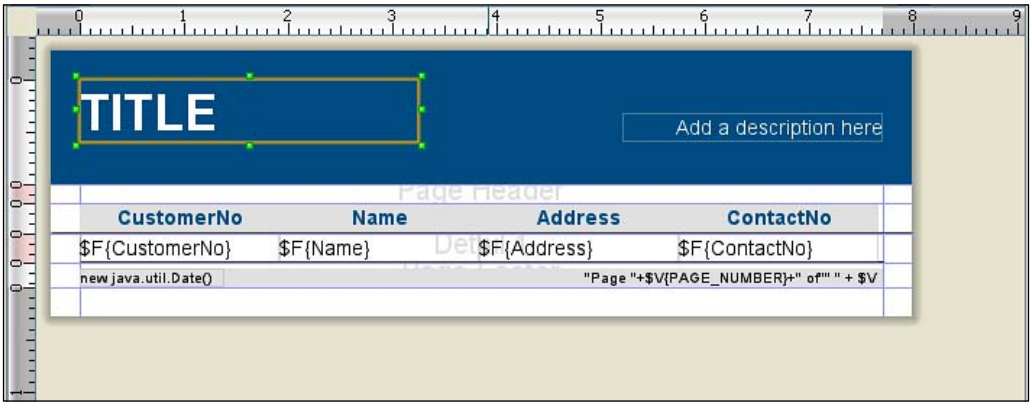


- Built-in editors for hyperlink, padding and borders, numbers, and date patterns.
- Using a mouse or keyboard to position, align, and resize objects.
- Comprehensive set of formatting tools.

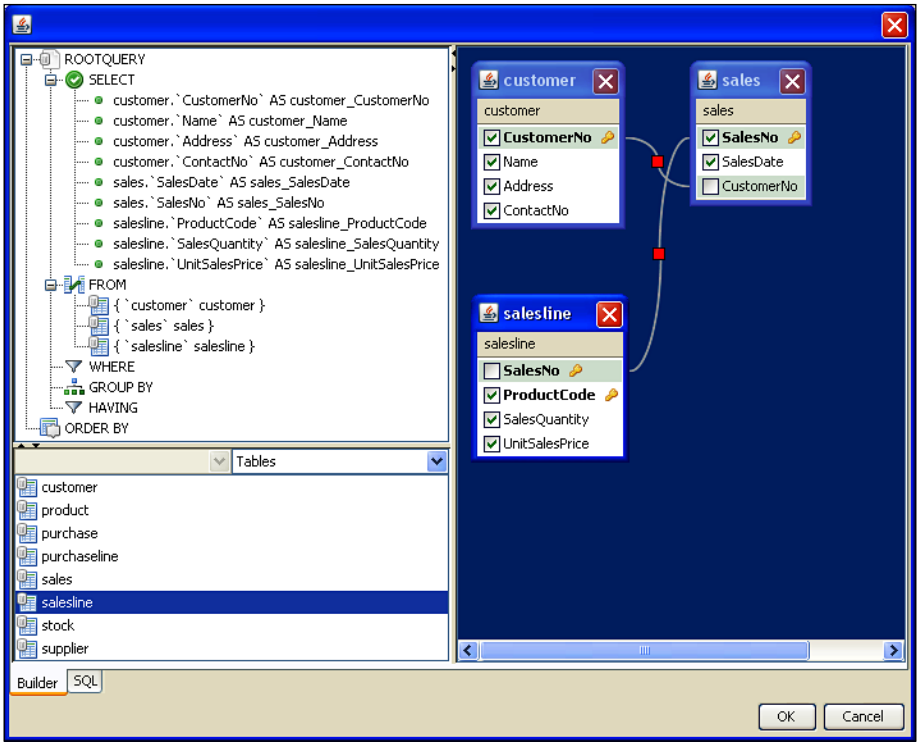


- Contextual menu for performing common operations quickly.
- Resizable bands for header, footer, summary, background, and so on, with multiple details.

- Copy and paste elements.
- Grids and rulers for formatting elements.



- Real-time design error reporting.
- Crosstab and subreport designer.
- Built-in query designer.

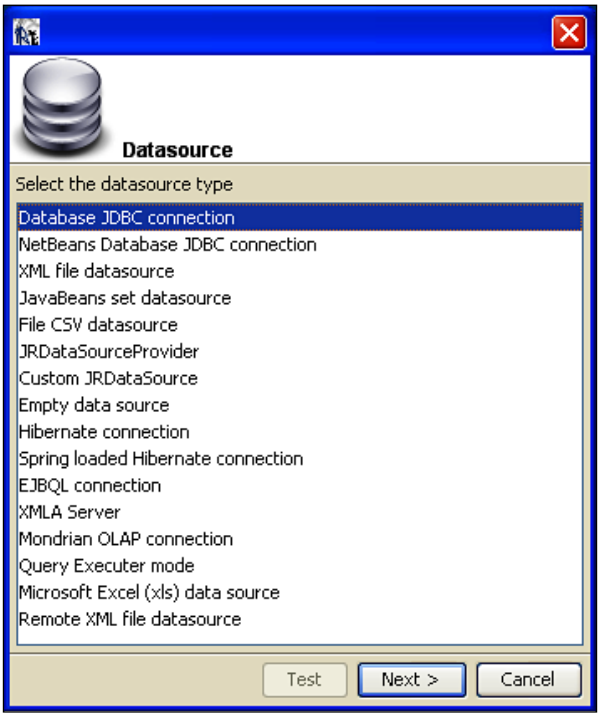


- Variables to perform calculations at different report levels, including `sum`, `average`, `count`, `min`/`max`, custom calculations, and standard deviation.
- Grouping data and elements.
- Page setup options.
- Complete set of charts including **Pie**, **Pie 3D**, **Bar**, **Bar 3D**, **YX Bar**, **Stacked Bar**, **Stacked Bar 3D**, **Line**, **XY Line**, **Area**, **YX Area**, **Scatter**, **Bubble**, **Time Series**, **High Low**, **Candlestick**, **Gantt**, **Meter**, **Thermometer**, and **Multi Axis** charts.

- Graphic elements including images, frames, lines, rectangles, round rectangles, ellipses, and so on.
- Label and text fields with HTML and RTF formatting support.
- Scriptlets support.

Data sources

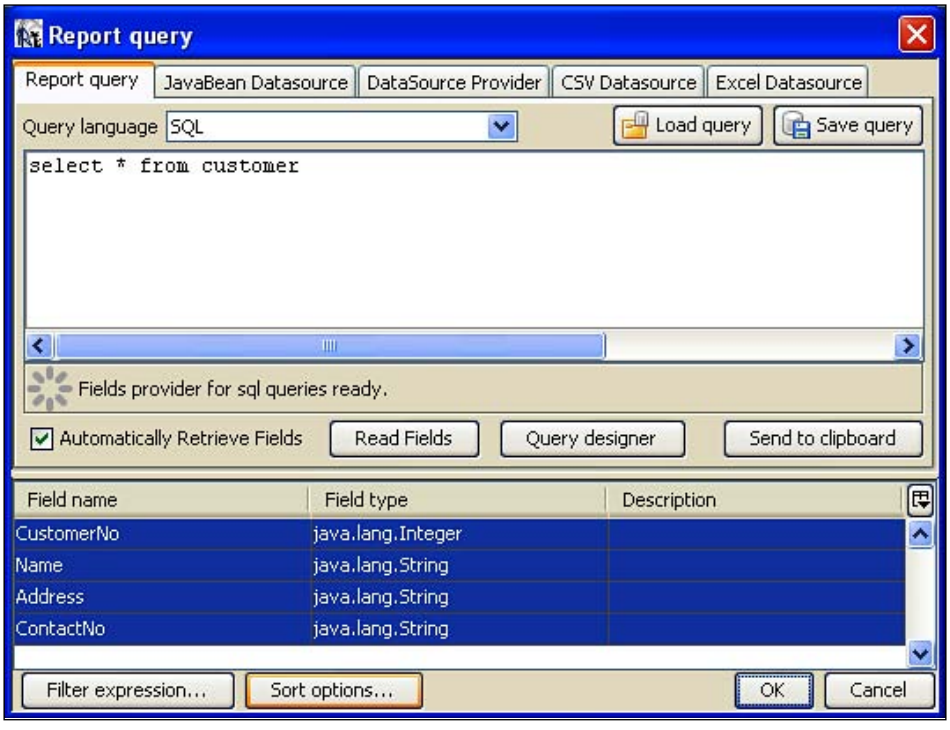
There are a significant number of data sources that you can connect to from iReport, as shown in the following screenshot:



Some of the features of iReport data sources are as follows:

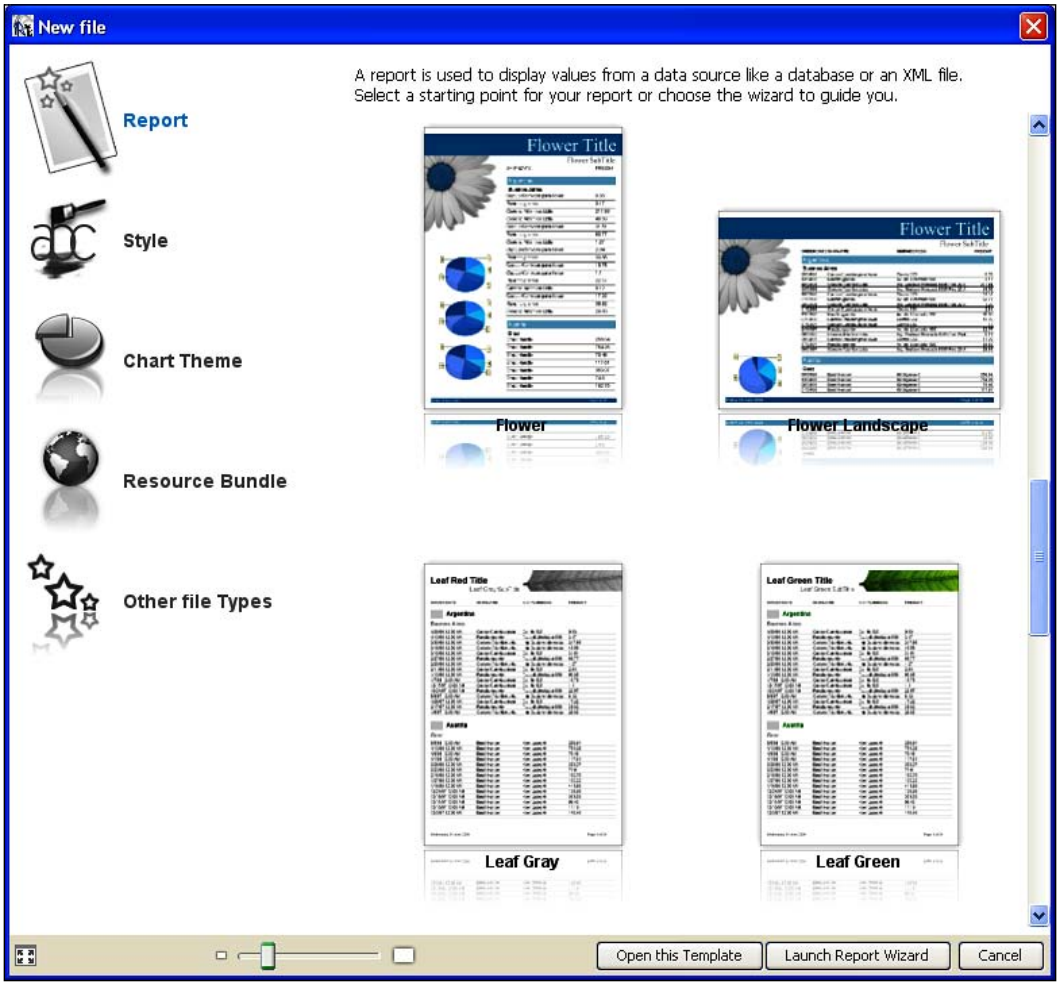
- Support for all relational databases accessible via JDBC and other data sources, including local and remote XML files, collections of JavaBeans, CSV files, hibernate connections, EJBQL connections, XMLA servers, custom data sources, and so on.
- Built-in support for SQL, HQL (Hibernate Query Language), MDX, and XPath with the integrated SQL query builder and the MDX designer.

- Field mapping tools for JavaBeans, XML, and CSV files.
- Ability to use multiple data sources for subreports and list components.
- Automatic retrieval of SQL fields.



Report templates

iReport has many built-in report templates, which you can use very easily with the Report Wizard. These templates provide automatic layout and orientation of report elements.



Features of the iReport templates include:

- Support for custom templates with the ability to import template libraries
- **Style**, **Chart Theme**, **Resource Bundle**, and **Other file Types** editor

Export and preview

In iReport, you get a one-click preview of your reports. The report viewer has the functionality to export the report to many other formats. Some of the features are as follows:

- Integrated PDF, HTML, XHTML, XLS, XLSX, RTF, DOCX, Text, ODT, XML preview
- Support for report exporting to PDF, HTML, XHTML, XLS, RTF, DOCX, Text, CSV, OpenOffice, XML, PNG with a comprehensive set of export options for each
- Support for multiple character encodings
- Setting up and automatic launch of external report viewers

iReport Classic vs iReport NB

iReport is available for download at `http://sourceforge.net/projects/ireport/files/`. There are mainly two versions available: Classic version and NB version. iReport 3.0.0 is the last release of the old classic version so far, and all the later releases are NB versions. iReport NB is available as a standalone application (based on the NetBeans RCP) and as a NetBeans plugin for NetBeans IDE.

Downloading and installing iReport

When downloading, you should select the file appropriate for your platform. Note that Sun Java 1.5.0 or later is required to run iReport.

- **Windows**: `iReport-3.7.0-windows-installer.exe`

  Win32 Installer, Windows XP, Vista, and Windows 7 are supported.
- **Mac OS X**: `iReport-3.7.0.dmg`

  Apple Mac OS X disk image file.
- **Linux**: `iReport-3.7.0.tar.gz`

  Generic `tar gz` distribution.
- **Other platforms**: `iReport-3.7.0.zip`

  Generic ZIP distribution for all the other platforms.
- **Plugin for NetBeans IDE 3.x**: `iReport-3.7.0-plugin.zip`

  This distribution allows us to execute iReport as a plugin for NetBeans IDE.

References

- *Getting Started*, by Giulio Toffoli, available at `http://jasperforge.org//website/ireportwebsite/IR%20Website/ir_getting_started.html?header=project&target=ireport`
- *iReport Features*, available at `http://jasperforge.org/website/ireportwebsite/IR%20Website/ir_features.html?header=project&target=ireport`.

Summary

We have understood what iReport is used for. Now, let's start working with it. Go through the coming chapters and discover the various reporting ideas.

2
Building Your First Report

We will develop our first simple report using iReport, and before doing so, we will create a connection/data source that will be used to fill the reports.

In this chapter, we will learn about:

- Creating a new data source or database JDBC connection
- Creating a report using the wizard
- Using the design query feature to produce the SQL query for the report
- Viewing and exporting the report using the built-in viewer

So let's get on with it!

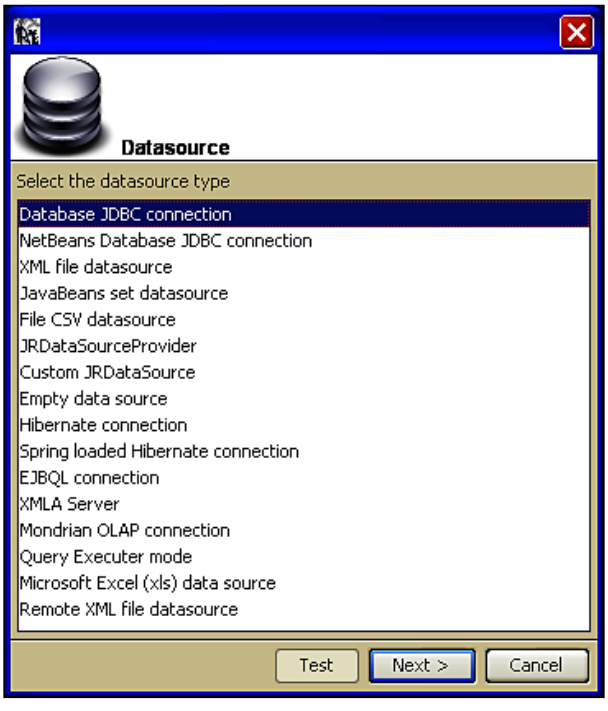
Creating a connection/data source

Before going to create the connection, a database should be set up. Designing the database used in this book for creating reports is explained in detail in the Appendix. However, for the purpose of this book, the SQL query for the database used for creating reports can be downloaded from the Packt website. Now, we are going to create a connection/data source in iReport and build our first report in some easy to follow steps:

You need to create the connection/data source just once before developing the first report. This connection will be reused for the following reports.

1. Start iReport.
2. Press the **Report Datasources** button in the toolbar. You will see a dialog box similar to the following screenshot:

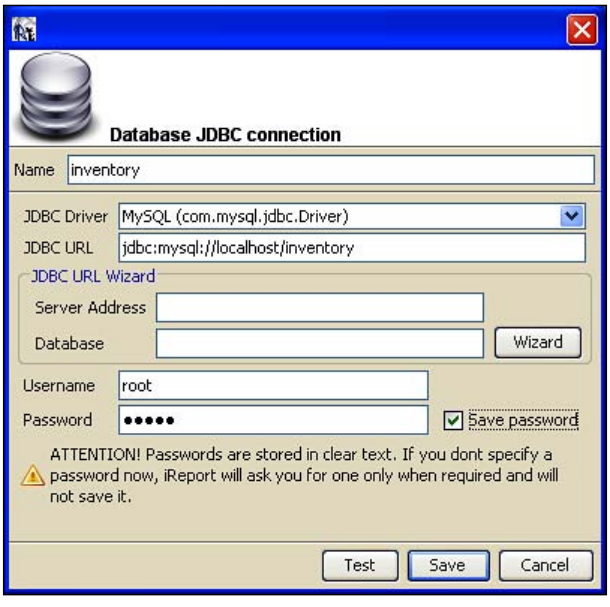
3. Press the **New** button. Another dialog box will appear for selecting the data source type. There are several types to choose from, according to your requirement. For now, choose **Database JDBC connection**, and press **Next >**.



4. Another dialog box will appear to set up the **Database JDBC connection** properties. Give a sensible name to the connection. In this case, it is **inventory**.
5. Choose the **JDBC Driver** from the list, according to your connection type and/or your database. In this case, it is **MySQL (com.mysql.jdbc.Driver)**.
6. Write the **JDBC URL**, according to the driver you have chosen. For this tutorial, it is **jdbc:mysql://localhost/inventory**.

In the previous code for connecting to a database from a Java program using JDBC—`jdbc` is the connection protocol, `mysql` is the subprotocol, `localhost` is the MySQL server if it runs on the same computer, and `inventory` is the database name.

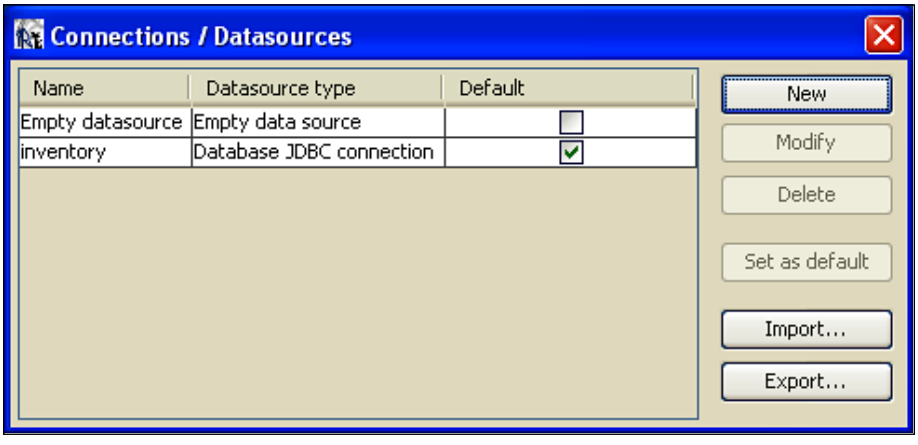
7. Enter the **Username** and **Password**. Generally, for a MySQL server, the username is **root** and you have set a customized password during the installation of the MySQL server. The screenshot is as follows:



8. Press **Test** to confirm that you have set all the properties correctly. If all the settings are correct, then you will see a message that says **Connection test successful!**.

You can save the password by checking the **Save Password** checkbox, but be warned that iReport stores passwords in clear text. Storing passwords in clear text is a bad thing for us, isn't it? If you do not specify a password now, iReport will ask you for one only when required and will not save it.

9. Now save the connection. You will see that the newly created connection is listed in the **Connections / Datasources** window. If you have more than one connections, then you can set one as the default connection. In order to do this, select the connection and press **Set as Default**.



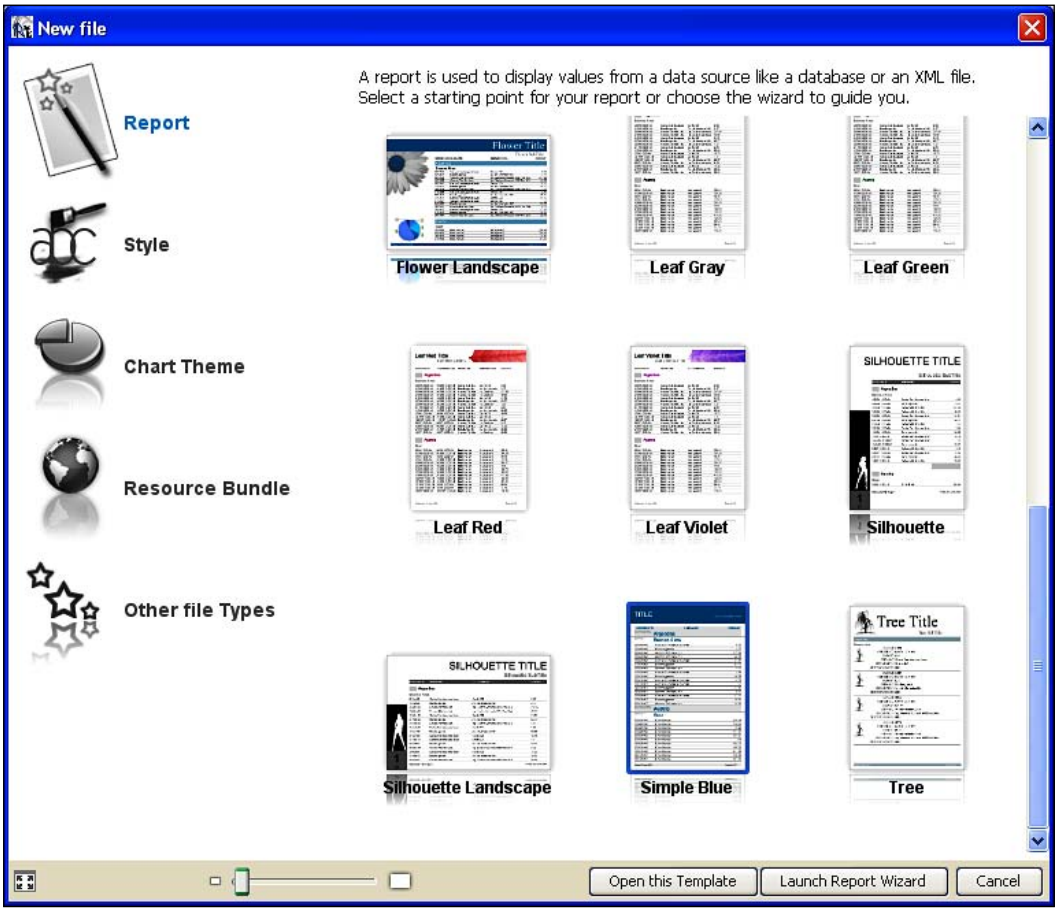
When we execute the report with an active connection, the reports are filled with data from the database or other data sources. We can also see the report output with empty data sources, which has, by default, a single record with all fields set to null. An empty data source is used to print a static report. However, in order to choose the tables and columns from a database automatically using the Report Wizard, we need to connect to a database/data source first. To do this, we must create a connection/ data source.

Building your first report

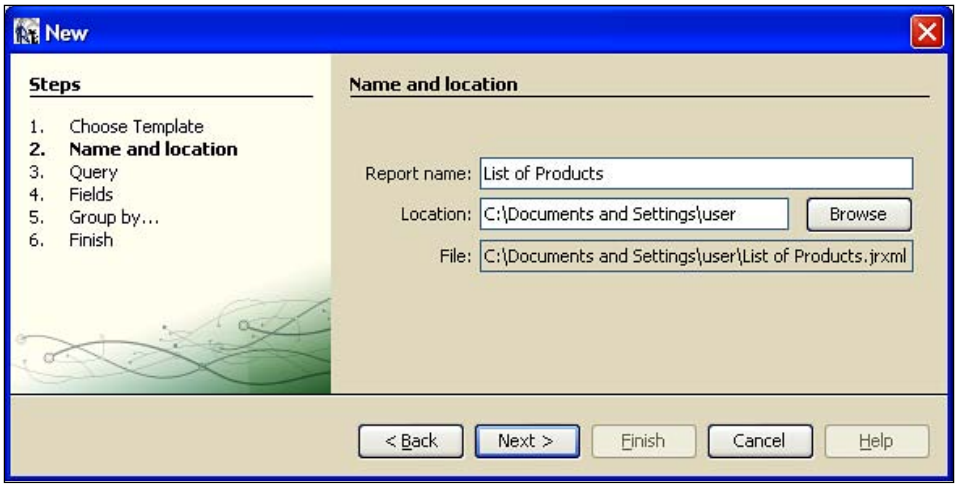
Having set up a connection, we are ready to build our first report. We will keep it very simple, just to be familiar with the steps required for building a report. We will create a report that lists out all the products; that is, we will show all the rows of the product table of our database.

Follow the steps listed and build your first report:

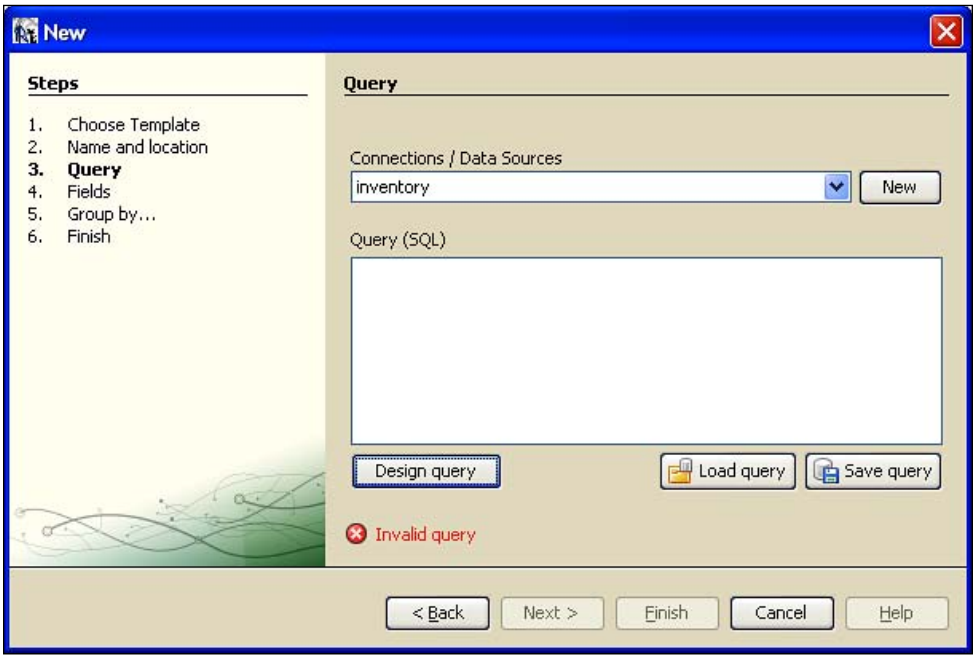
1. Go to the **File** menu and click **New…**. You will see a dialog box like the following screenshot:



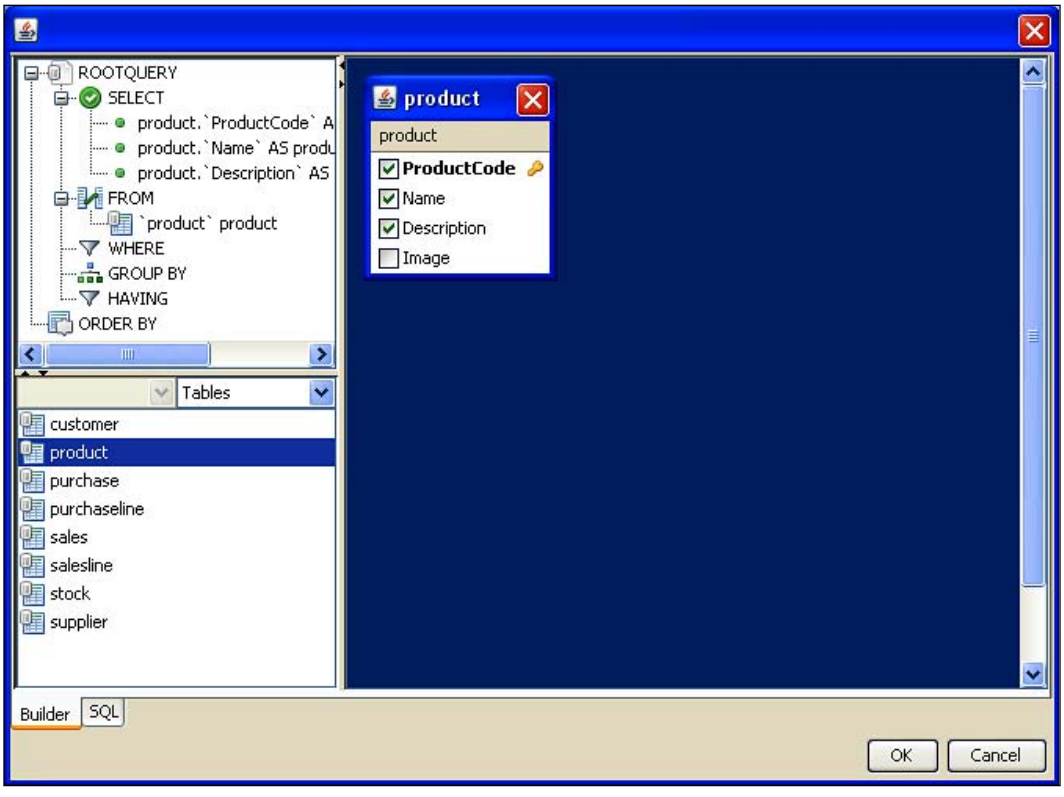
2. From the list of **Report** templates, select **Simple Blue** and press **Launch Report Wizard**.



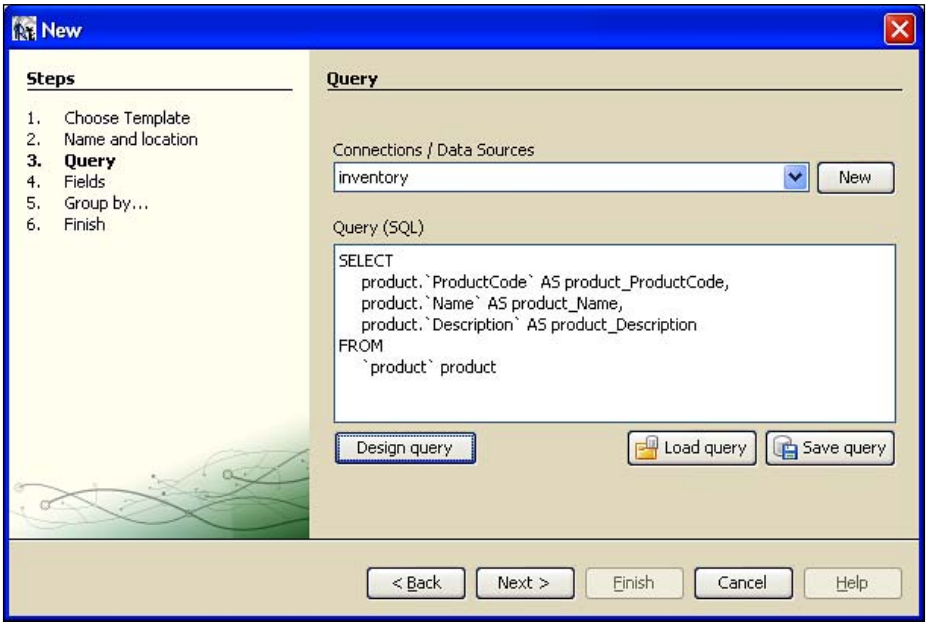
3. Enter **Report name** as **List of Products** and press **Next >**.
4. Now you will specify the query to retrieve the report fields. Select your connection from the **Connections / Data Sources** drop-down list.



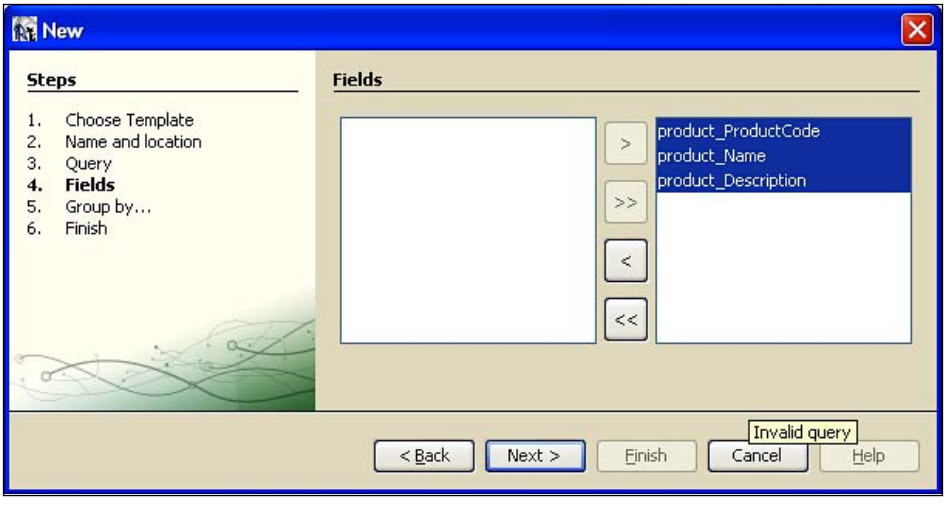
5. Write the SQL query for the report you want to develop. In our case, it is `SELECT ProductCode, Name, Description FROM Product`.
6. Alternatively, you can use the **Design query** option if you want to design the query graphically by selecting tables and columns easily, without writing the SQL commands. For this, double-click on the table name, select the fields from the table, and then press **OK**.



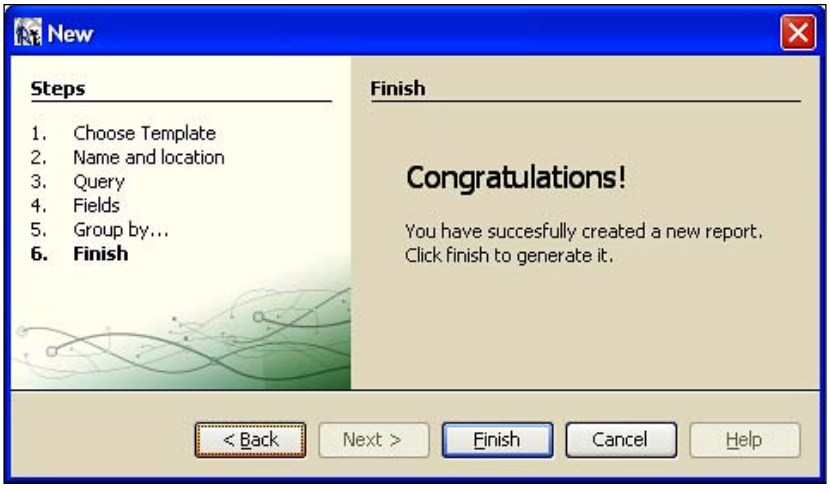
7. After writing the SQL query or designing the query, press **Next >**.



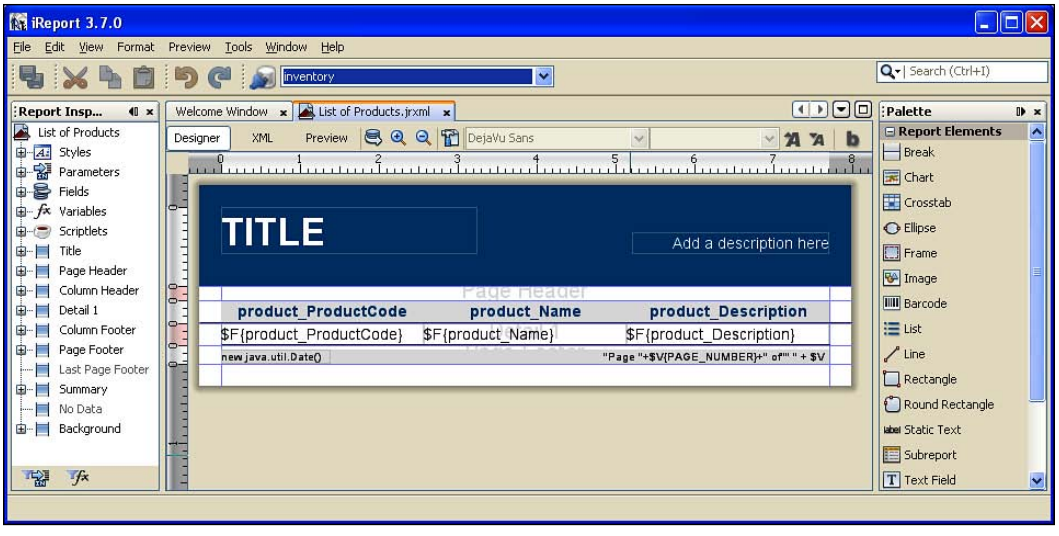
8. In this step, we will select the fields that we want to show in the report. We may select all the fields or choose some of them. For this example, select all the fields (**ProductCode**, **Name**, and **Description)** by clicking on **>>** (double arrow). Note that the fields listed here are based on the query specified in the previous step.



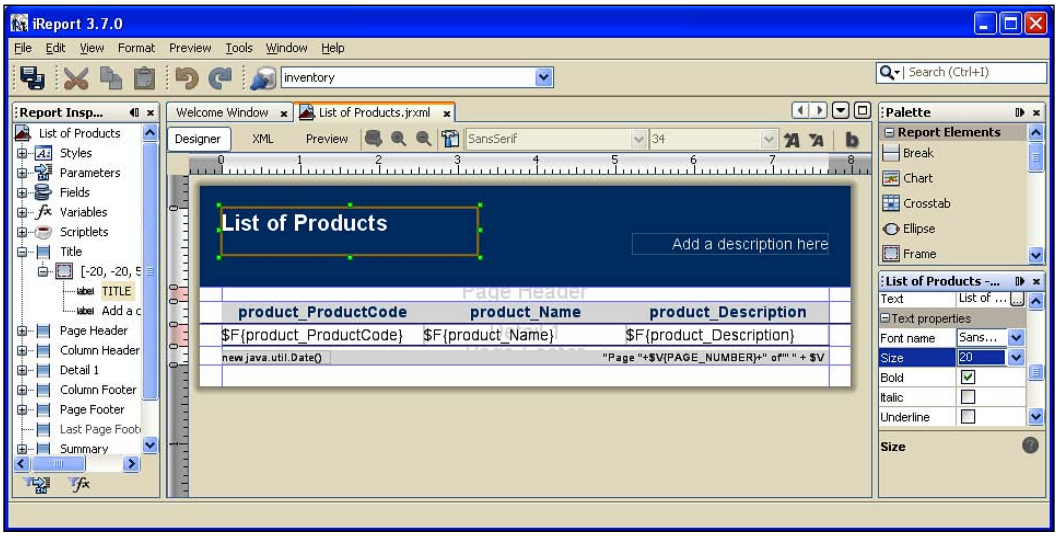
9. After selecting the fields, press **Next >**.
10. Now we are in step **5** (**Group by…**) of the wizard. Ignore this just for now; we will learn about grouping later. Press **Next >** to proceed.
11. We are at the last step of the Report Wizard. Now press **Finish**.



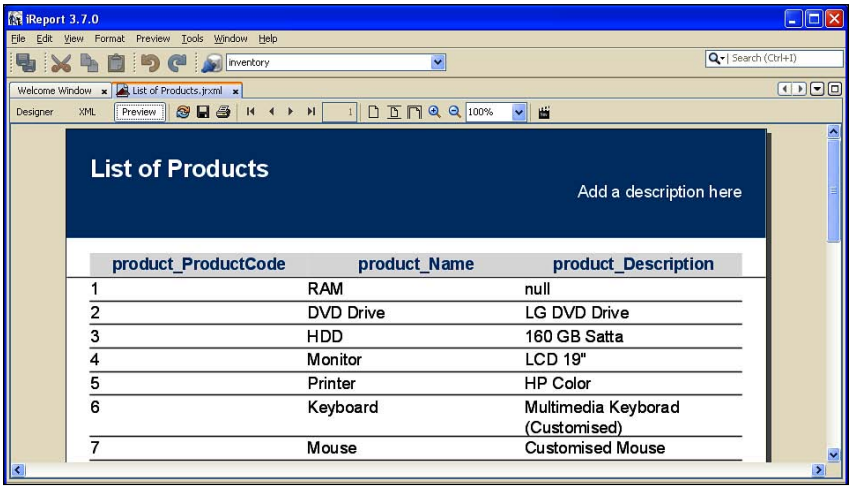
12. You will see the following output:



13. To change the title, select **TITLE**, and enter **List of Products** in the **Text** box of the **Properties** window, at the bottom-right corner. You can change the font name, size, and style according to your requirements from the same window. In this example, **Font name** is **SansSerif**, **Size** is **20**, and style is **Bold**.



14. It's time to see the output of your first report. Just press the **Preview** button at the top of the report design. You will see a report similar to the following one:



product_ProductCode	product_Name	product_Description
1	RAM	null
2	DVD Drive	LG DVD Drive
3	HDD	160 GB Satta
4	Monitor	LCD 19"
5	Printer	HP Color
6	Keyboard	Multimedia Keyborad (Customised)
7	Mouse	Customised Mouse

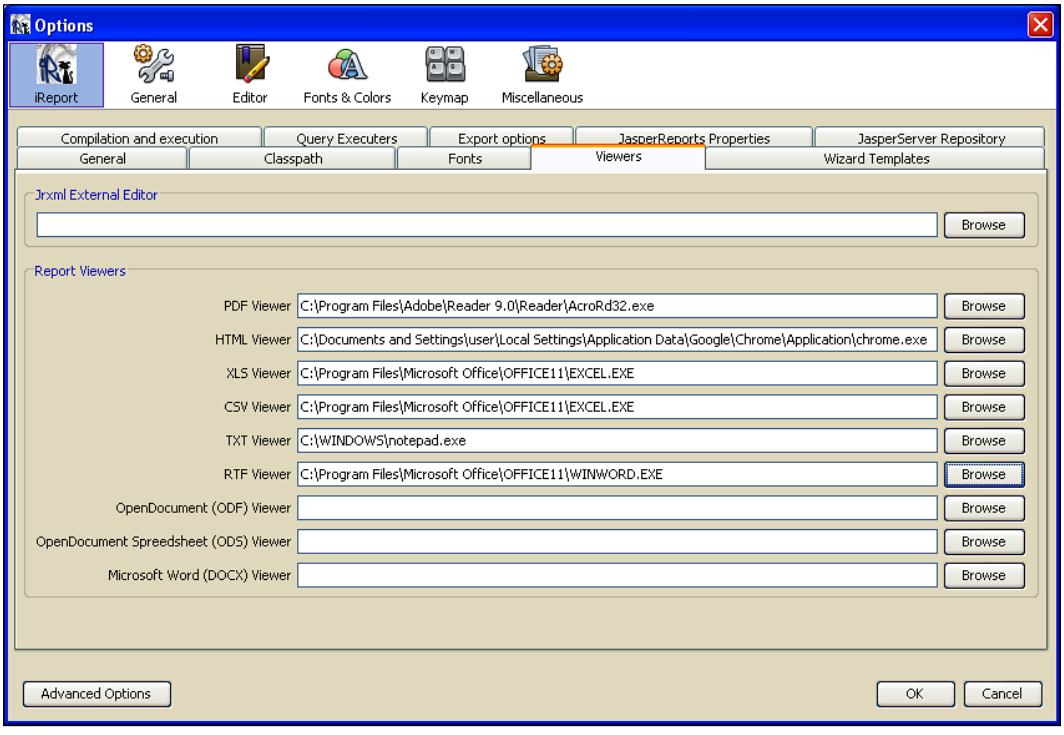
To build the report easily, we can use the Report Wizard, where the database columns can be selected from the list, and the layout of the report can be set very easily by choosing templates. Note that for choosing the layout, the tabular format is generally suitable if your report shows more than one row from the database table, and the column format is suitable if the query returns only a single row. If the number of columns in the query result is quite large, then you can choose the landscape template.

Viewing and exporting the report

We can view our report using JRViewer or any other supported external programs, such as Acrobat Reader (PDF preview), Microsoft Word (RTF preview), any browser (HTML preview), and so on. You have seen the JRViewer preview. There is a built-in toolbar in the JRViewer; here you can save the report in various formats, such as PDF, RTF, HTML, CSV, XML, and so on. To do this, just press the **Save** button, and then choose **PDF/ RTF/ HTML/ CSV/ XML** or others from the **Files of type** drop-down list.

You can print the report from the viewer by just pressing the **Print** button in the toolbar. If your report contains more than one page, then you can navigate through the report pages using these four buttons: **First Page**, **Previous Page**, **Next Page**, and **Last Page** from the toolbar. You can resize the page to **Fit the page**, **Fit the page width**, or **Actual size**. You can also zoom in or zoom out. All options are available in the toolbar—you just have to choose the appropriate button.

You can change the report viewer from the **Preview** menu to see the preview in another application. You can choose from PDF, HTML Java 2D, RTF, or other preview options. However, before seeing these previews, you have to set the external programs for the appropriate viewers. You can do this from **Tools** | **Options** | **iReport** | **Viewers**. Browse the required programs for each viewer. Then choose the appropriate viewer from the **Preview** menu, and preview the report again.



Summary

In this chapter, we had a look at creating connections and building simple reports.

Specifically, we covered:

- Creating a database JDBC connection
- Using the Report Wizard
- Using the design query feature for designing a query without writing the SQL command
- Choosing the appropriate layout and template
- Viewing report in JRViewer and other external programs

Now that we've learned about generating a report design structure using the Report Wizard, we're ready to design ahead to get a professional (or required) look in the generated reports. This will be covered in the next chapter.

3
Report Layout and Formatting

In this chapter, we will concentrate on the designing of the report after the initial layout is generated using the Report Wizard, which we learned in Chapter 2, *Building Your First Report*. You saw that the report title was not sensible; even the column names were not as sensible as those stated in the database table's columns. We sometimes give short names in database table columns, though we should follow naming standards, that is we don't use spaces or other characters in names to follow the naming standards. However, in a report, we must give sensible column headings for the user. Note that in database tables, the column headings are identifiers, but in the report, the column headings are not identifiers. Besides these, we may want to give a suitable header and footer, choose fonts, change page size, margin, and so on in our report. We will learn about these types of elements in this chapter.

In this chapter, we will cover:

- Configuring report properties
- Understanding bands
- Configuring and using report bands
- Maintaining the size, position, and alignment of the report elements
- Modifying report fonts
- Using the library
- Using borders
- Modifying element properties

Setting up the report pages

In Chapter 2, we created a report `List of Products` that we will now continue to use for achieving the learning objectives of this chapter.

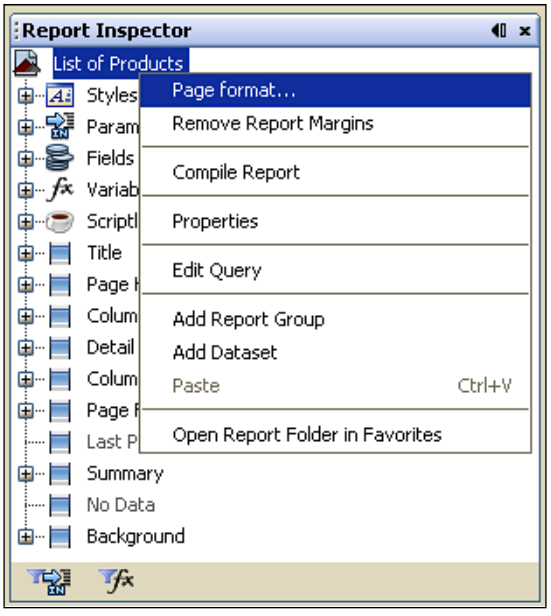
If we use the Report Wizard for generating reports, some default report properties (page size, margins, and so on) are set automatically. For our software or reports, we may need to modify the default report properties at times, so that it fulfills the user (or software) requirements.

Configuring the page format

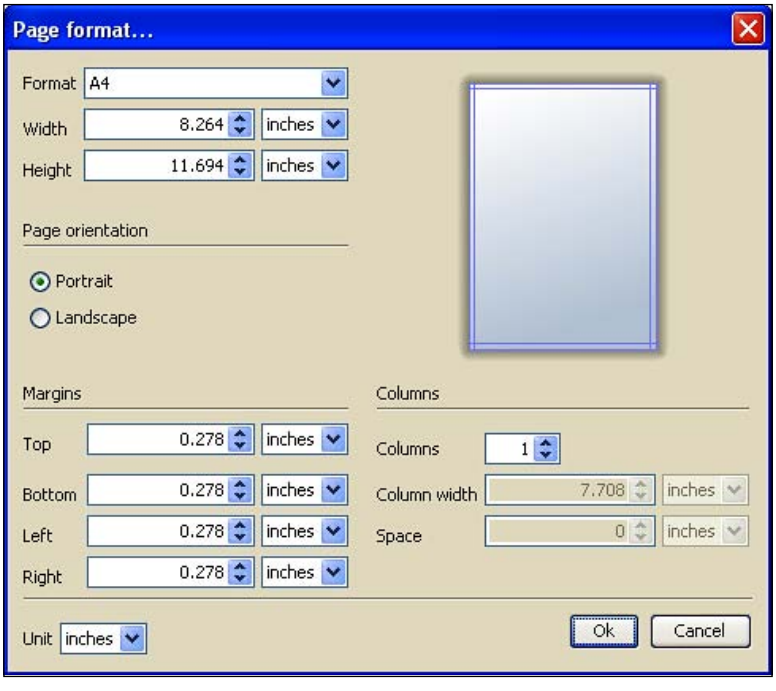
We can follow the listed steps for setting up report pages:

1. Open the report **List of Products**.
2. Go to menu **Window | Report Inspector**. The following window will appear on the left side of the report designer:

3. Select the report **List of Products**, right-click on it, and choose **Page Format…**.



4. The **Page format…** dialog box will appear, select **A4** from the **Format** drop-down list, and **select Portrait** from the **Page orientation** section.



5. You can modify the page margins if you need to, **or leave it as it is to have** the default margins. For our report, you need not change the margins.
6. Press **OK**.

Page size

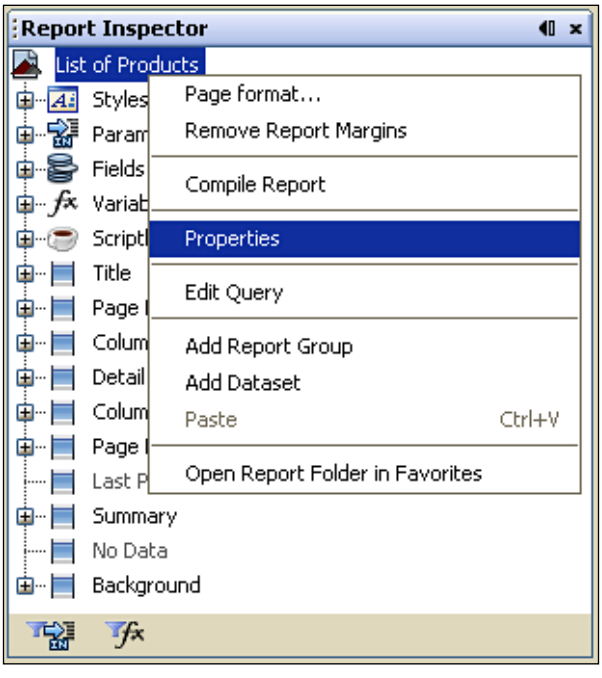
You have seen that there are many preset sizes/formats for the report, such as **Custom**, **Letter**, **Note**, **Legal**, **A0** to **A10**, **B0** to **B5**, and so on. You will choose the appropriate one based on your requirements. We have chosen **A4**. If the number of columns is too high to fit in **Portrait**, then choose the **Landscape** orientation.

If you change the preset sizes, the report elements (title, column heading, fields, or other elements) will not be positioned automatically according to the new page size. You have to position each element manually. So be careful if you decide to change the page size.

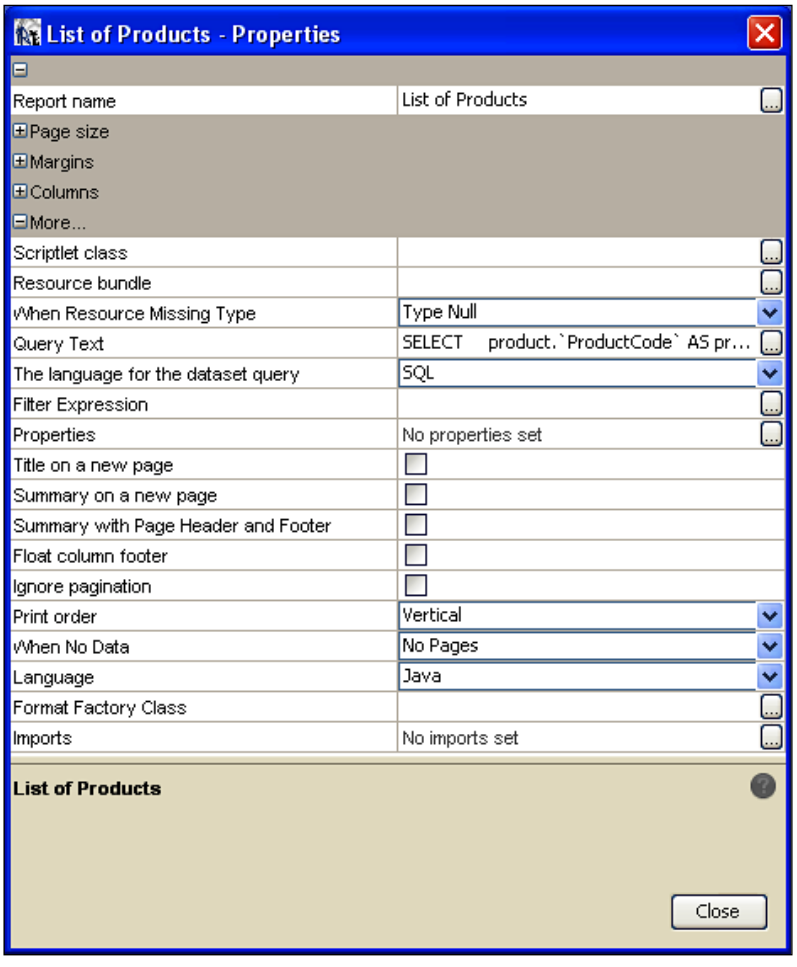
Configuring properties

We can modify the default settings of report properties in the following way:

1. Right-click on **List of Products** and choose **Properties**.



2. We can configure many important report properties from the **Properties** window.



You can see that there are many options here. You can change the **Report name**, **Page size**, **Margins**, **Columns**, and more. We have already learnt about setting up pages, so now our concern is to learn about some of the other (**More...**) options.

What are the different checkboxes?

You'll see **Title on a new page**, **Summary on a new page**, **Floating column footer**, **Ignore pagination** checkboxes in the **More...** tab in the report **Properties** window.

What will happen if you check these options?

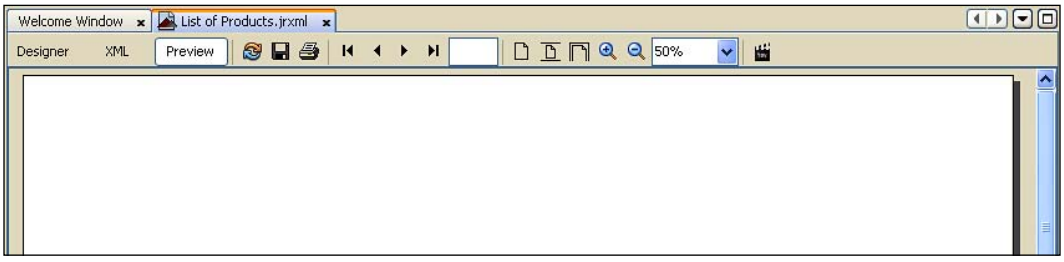
Title on a new page	If this option is checked, then the report title will be shown on a new page, that is, the report title will be shown on the first page without any data, and the report data will be shown on subsequent pages.
Summary on a new page	If the report contains a summary, generally it is shown just after the data, but if this option is checked, then the summary will be shown on a new page. Don't worry about the report summary, as you will learn about the report summary in the coming chapters.
Floating column footer	If a report has column footer, then it is generally shown at the bottom of the page. However, if this option is checked, then column footer will be shown at the bottom of the column, even if it is in the start/middle of the page.
Ignore pagination	If this option is checked, then the whole report will be shown in a single page. Your page size settings (height of the page) will not work if you check this option. Page height will depend on the report data. If data is less, then the page height will be small, but the height will be increased if data is more. This option is suitable for POS printing where the page height depends on the products the customer purchases.

When there is no data

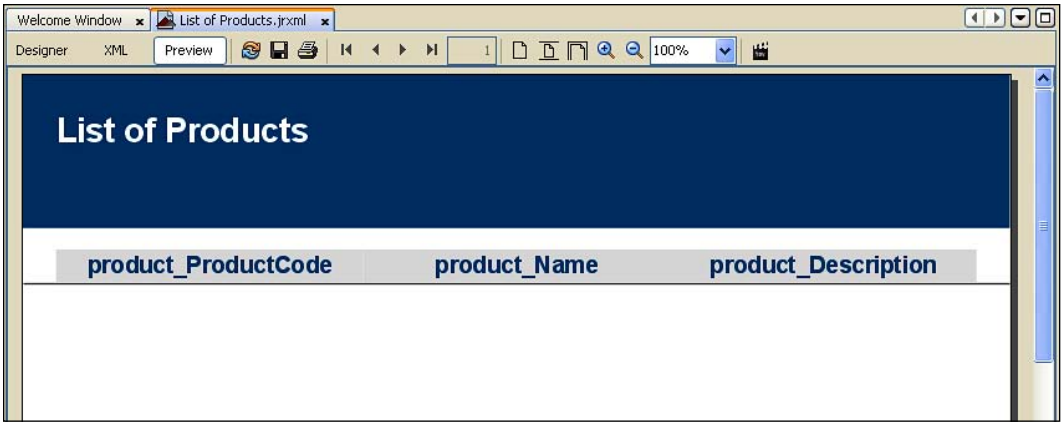
When we execute the report, it is filled with data from the data source (database or others). But sometimes there may be no data (depending on the query). What will happen then? We may deal with this situation using the **When No Data** option in the **More...** section of the report **Properties** window. There are four options :

- **No pages**: If the report has no data, then just a dialog box will be shown with the message **The document has no pages** and the report viewer will not be shown.

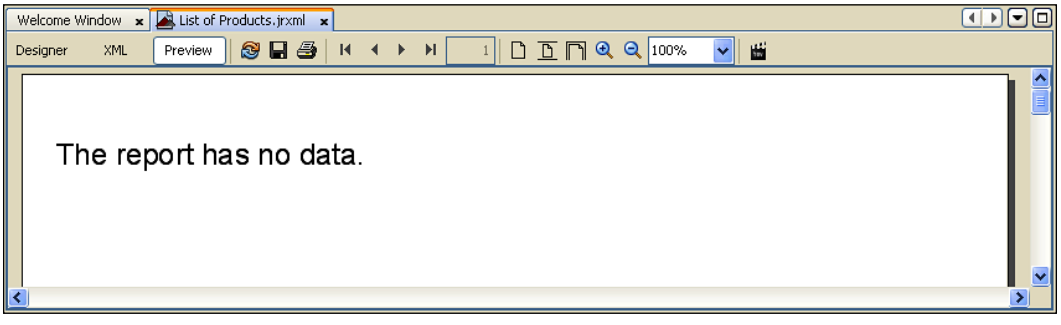
- **Blank Page**: The report viewer will be shown with a blank page like the following screenshot:



- **All Sections, No Detail**: All sections of the report will be shown without data, that is, you will see just the report structure.



- **No data Section**: If no data is available, then a customized message will be shown in the report page of the report viewer. In order to do this, just checking the checkbox is not enough, you have to configure the **NoData** band. You will learn about bands in the next section of this chapter.



Configuring bands, formatting reports and elements

A complete report is structured by composing a set of sections called **bands**. Each band has its own configurable height, a particular position in the structure, and is used for a particular objective. The available bands are: **Title**, **Page Header**, **Column Header**, **Detail 1**, **Column Footer**, **Page Footer**, **Last Footer**, and **Summary**.

A report structured with bands is shown in the following screenshot:

Besides the mentioned bands, there are two special bands which are **Background** and **No Data**.

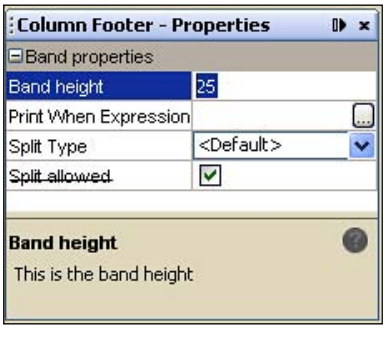
Band	Description
Title	Is the first band of the report and is printed only once. Title can be shown on a new page. You can configure this from the report properties discussed in the previous section of this chapter. Just to review—go to report **Properties** | **More...** and check the **Title on a new page** checkbox.
Page Header	Is printed on each page of the report and is used for setting up the page header.
Column Header	Is printed on each page, if there is a detail band on that page. This band is used for the column heading.
Detail	This band is repeatedly printed for each row in the data source. In the **List of Products** report, it is printed for each product record.

Band	Description
Column Footer	Is printed on each page if there is a detail band on that page. This band is used for the column heading. If the **Floating column footer** in report **Properties** is checked, then the column footer will be shown just below the last data of the column, otherwise it will be shown at the bottom of the page (above the page footer).
Page Footer	Is printed on each page except the last page, if **Last Page Footer** is set. If **Last Page Footer** is not set, then it is printed on the last page also. This band is a good place to insert page numbers.
Last Page Footer	Is printed only on the last page as a page footer.
Summary	Is printed only once at the end of the report. It can be printed on a separate page if it is configured from the report **Properties**. In the following chapters, we will produce some reports where you will learn about the suitability of this band.
Background	Is used for setting a page background. For example, we may want a watermark image for the report pages.
No Data	When no data is available for the reports, this band is printed if it is set as the **When no data** option in the report **Properties**.

Showing/hiding bands and inserting elements

Now, we are going to configure the report bands (setting height, visibility, and so on) and format the report elements.

1. Select **Column Footer** from the **Report Inspector**. You will see the **Column Footer - Properties** on the right of the designer.
2. Type **25** in the **Band height** field.

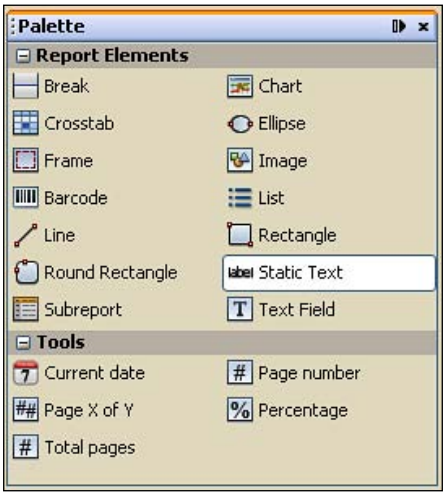


3. Press *Enter*. Now you can see the **Column Footer** band in your report, which was invisible before you set the band height.

A band becomes invisible in the report if its height is set to zero. We have already learned how to change the height of a band. We can also make a band invisible using the **Print When Expression** option. If we write `new Boolean(false)` in **Print When Expression** of a band, then that will make the band invisible, even though its height is set to greater than zero. If we write `new Boolean(true)`, then the band will be visible. It is `true` by default.

4. Drag a **Static Text** element from the **Palette** window and drop it on the **Column Footer** band. Double-click on **Static Text** and type **End of Record**, replacing the text **Static Text**.

If the **Palette** window is not visible, then go to the **Window** menu and select **Palette**.

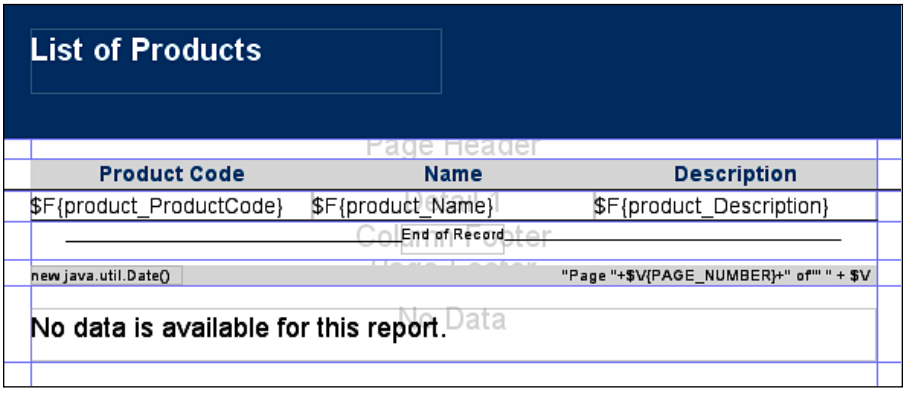


5. Select the static text element (**End of Record**). Go to **Format | Position** and then choose **Center**. Now the element has been positioned in the center of the **Column Footer** band.

6. In the same way, insert two **Line** elements. Place one element at the left and another at the right of the static text.
7. Select both the lines. Go to **Format** | **Position**, and then choose **Center Vertically**. The lines are now positioned in the center of the **Column Footer** vertically.

Use the *Ctrl* or *Shift* key to select more than one element.

8. Select both the lines and go to **Format** | **Size** and then choose **Same Width**. Now both the lines are equal in width.
9. Select the static text element (**End of Record**) and the left line. Now go to **Format** | **Position** and choose **Join Sides Right**. This moves the line to the right, and it is now connected to the static text element.
10. Repeat the previous step for the right line and finally choose **Join Sides Left**. Now the line has moved to the left and is connected with the static text element.
11. In the same way, change the column headers as you want by double-clicking the labels on the **Column Header** band. Now, the columns may be **Product Code**, **Name**, and **Description**.
12. Now your report design should look like the following screenshot:



13. **Preview** the report, and you will see the lines and static text (**End of Record**) at the bottom of the column.

By default, the **Column Footer** is placed at the bottom of the page. To show the **Column Footer** just below the table of data, the **Float column footer** option must be enabled from the report **Properties** window.

List of Products

Product Code	Name	Description
1	RAM	null
2	DVD Drive	LG DVD Drive
3	HDD	160 GB Satta
4	Monitor	LCD 19"
5	Printer	HP Color
6	Keyboard	Multimedia Keyborad (Customised)
7	Mouse	Customised Mouse

End of Record

Sizing elements

We can increase or decrease the size of an element by dragging the mouse accordingly. Sometimes, we need to set the size of an element automatically based on other elements' sizes. There are various options for setting the automatic size of an element. These options are available in the format menu (**Format** | **Size**).

Size Options	Description
Same Width	This makes the selected elements of the same width. The width of the element that you select first is used as the new width of the selected elements.
Same Width (max)	The width of the largest of the selected elements is set as the width of all the selected elements.
Same Width (min)	The width of the smallest of the selected elements is set as the width of all the selected elements.
Same Height	This makes the selected elements of the same height. The height of the element that you select first is used as the new height of the selected elements.
Same Height (max)	The height of the largest of the selected elements is set as the height of all the selected elements.
Same Height (min)	The height of the smallest of the selected elements is set as the height of all the selected elements.
Same Size	Both the width and the height of the selected elements become the same.

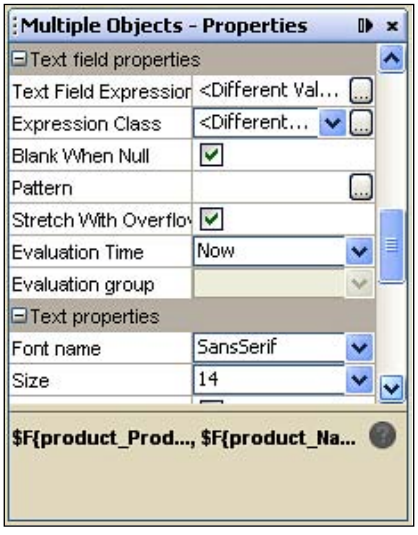
Positioning elements

We can change the position of elements easily by using the drag-and-drop feature. However, for automatic positioning of an element based on bands/cell or other elements, we can use the options in **Format | Position**.

Position	Description
Center Horizontally (band/cell based)	The selected element is placed in the center of the band horizontally.
Center Vertically (band/cell based)	The selected element is placed in the center of the band vertically.
Center (in band/cell)	The selected element is placed in the center of the band both horizontally and vertically.
Center (in background)	If the **Background** band is visible and if the element is on the **Background** band, then it will be placed in the center both horizontally and vertically.
Join Left	Joins two elements. For joining, one element will be moved to the left.
Join Right	Joins two elements. For joining, one element will be moved to the right.
Align to Left Margin	The selected element will be joined with the left margin of the report.
Align to Right Margin	The selected element will be joined with the right margin of the report.

Handling null values

In the report, you can see a **null** value for **Description** of **RAM**. This is because in the database this field is null. To display a blank when it is null in the database, select all the fields (**$F{ProductCode}**, **$F{Name}**, **$F{Description}**), and then check **Blank when null** from the **Multiple Objects - Properties**.



Now, the report output will be as shown in the following screenshot (**Description** of **RAM** is blank):

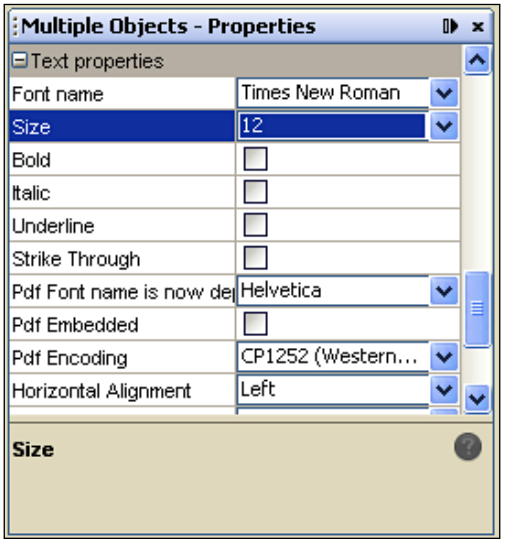
List of Products

Product Code	Name	Description
1	RAM	
2	DVD Drive	LG DVD Drive
3	HDD	160 GB Satta
4	Monitor	LCD 19"
5	Printer	HP Color
6	Keyboard	Multimedia Keyborad (Customised)
7	Mouse	Customised Mouse

End of Record

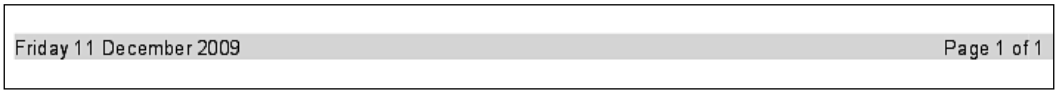
Font settings

We will now change the report font for the data elements. Select all the fields, find **Font name** from **Properties**, and choose **Times New Roman** as **Font name**, and **12** as **Size**.

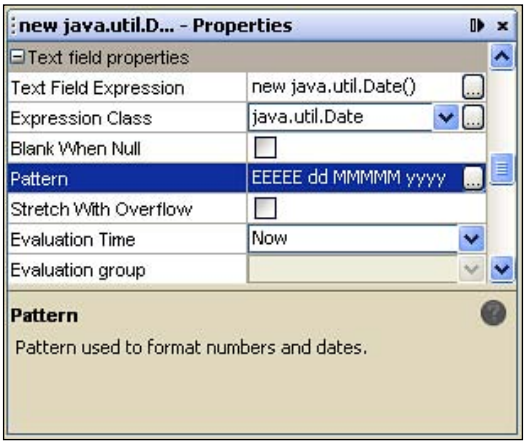


Creating text field pattern

In the **Page Footer** band, you can see the current date and the page numbers, as shown in the following screenshot:

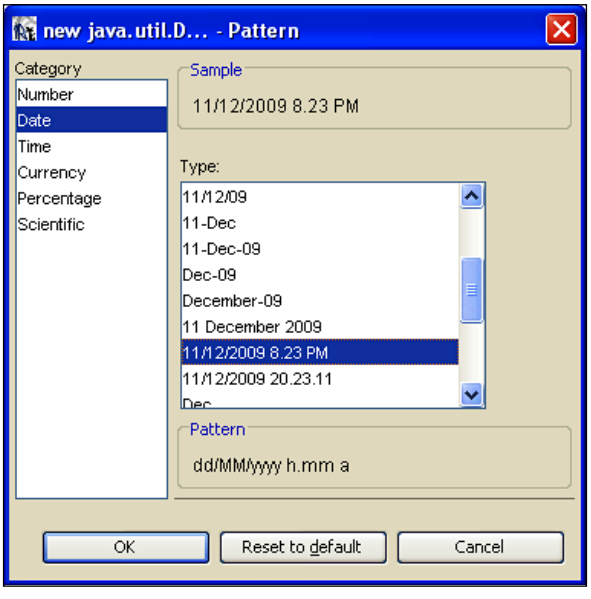


We will modify the elements of **Page Footer** now. To change the date format, select date element (`new java.util.Date()`), go to **Properties** | **Text Field Properties**, and see the **Pattern**.

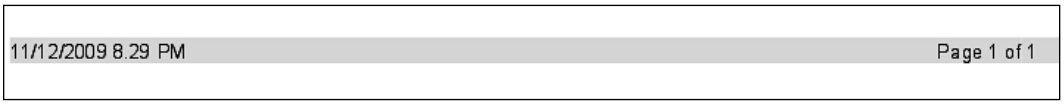


Now, follow the steps listed to change the date pattern:

1. Open the pattern editor by clicking the button next to **Pattern**.
2. Select **Date** from **Category**.
3. Select **dd/MM/yyyy h.mm a** from the **Type** window.



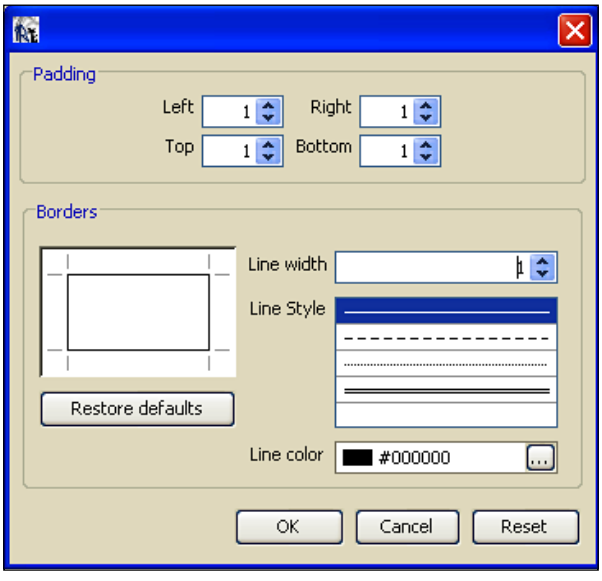
4. Press **OK**. The date pattern is changed to the output, as shown in the following screenshot:



Setting borders

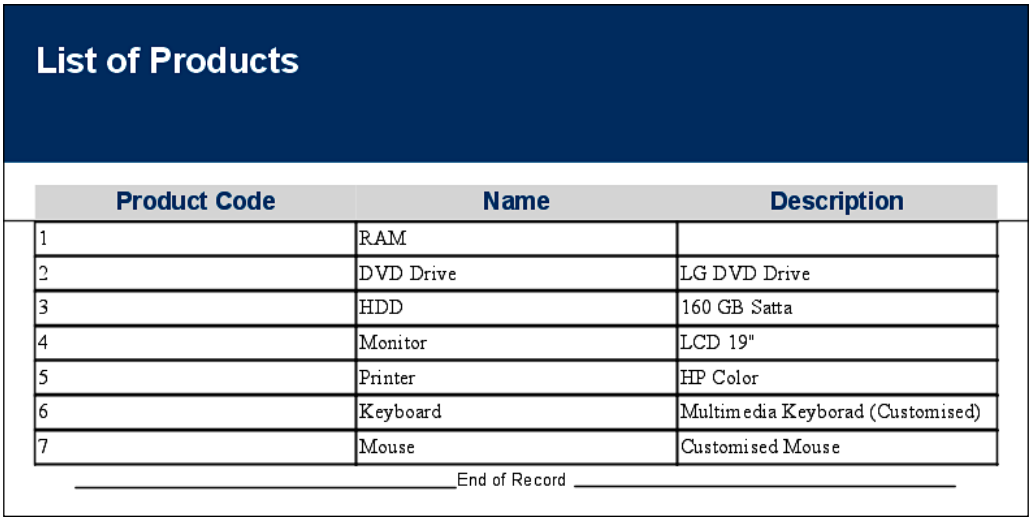
Now we will set borders for the field elements in some easy steps:

1. Select all fields.
2. Right-click and select **Padding and Borders**.



3. Set the **Left**, **Right**, **Top**, and **Bottom Padding** as **1**.
4. Set **1** as the **Line width**.
5. Select **Line color**.
6. Press **OK**.

7. **Preview** the report to see the output, as shown in the following screenshot:



List of Products

Product Code	Name	Description
1	RAM	
2	DVD Drive	LG DVD Drive
3	HDD	160 GB Satta
4	Monitor	LCD 19"
5	Printer	HP Color
6	Keyboard	Multimedia Keyborad (Customised)
7	Mouse	Customised Mouse

End of Record

Using tools for current date and inserting page numbers

Current date and page numbers are automatically set on the **Page Footer** band, if the report is created from the wizard. However, to add these manually to the report, we can use built-in tools in the **Palette**.

From the library options, drag the **Page X of Y** to the **Page Footer** band of your report, where you want to show the page numbers. Again, drag the **Current Date** in the same way.

Summary

We learned a lot in this chapter about formatting the report and its elements.

Specifically, we covered:

- Bands
- Setting the size of elements
- Setting the position of elements
- Setting the pattern for fields
- Setting the fonts of elements
- Using the built-in tools
- Setting the border for the elements

Now that we've learned about designing the report, we're ready to learn more about the types of reporting, and this is covered in the coming chapters.

4
Using Variables

In the previous chapters, we have retrieved data from the database and just viewed those after designing the report, but we haven't done any other processing using the retrieved data. Sometimes, we need further processing of data. For example, we have retrieved the quantity and unit price of all products of a particular sale. Now, we might want to calculate the total by multiplying quantity and unit price. We may also want to calculate the grand total of that sale. It is mentioned that the total and the grand total are not stored in the database. To calculate these during the runtime of the report, we need to use **variables**.

You might already know that a variable stores data, which can be modified during program execution. Thus a variable is called a named memory location or a unit of storage. In a report, any processing result may be stored in a variable, which is displayed in the report or used to process the data further.

In iReport, the report variable is a special object that holds value during the runtime of the report, based on the expression and other setups (`Reset` type, `Reset` group, `Increment` type, `Increment` group, and so on).

As you develop a report with the use of variables, you will be able to:

- Understand the use of variables
- Add variables to a report
- Write variable expression

Reviewing the database tables

Among the database tables, the following will be used:

Tables	Product	Sales	SalesLine
Attributes	ProductCode	SalesNo	SalesNo
	Name	SalesDate	ProductCode
	Description	CustomerNo	SalesQuantity
	Image		UnitSalesPrice

Creating a basic report

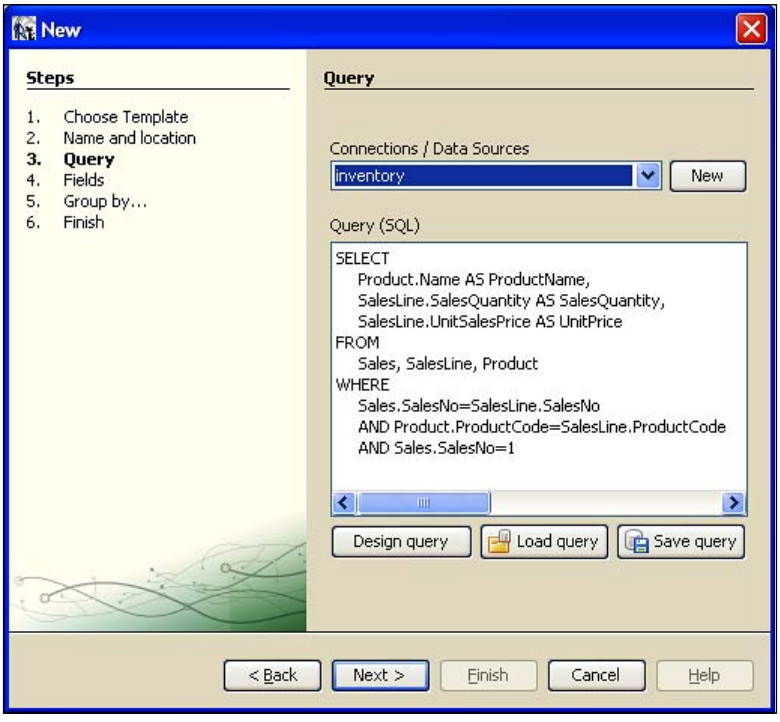
Now, we will create a report that shows details of **Sales No : 1** (one), that is, the report will show the product name, quantity, unit price, total, and grand total. Here "total" and "grand total" don't exist in the database tables. So, we will use iReport variables to calculate these.

Let's first create a report in some basic steps, as we did in Chapter 2, *Building Your First Report*. The SQL command we will use here is as follows:

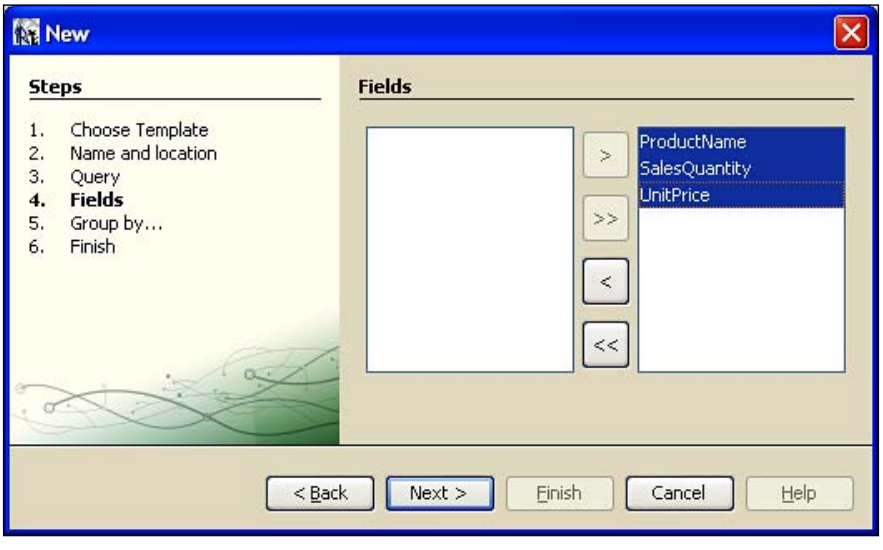
```
SELECT
     Product.Name AS ProductName,
     SalesLine.SalesQuantity AS SalesQuantity,
     SalesLine.UnitSalesPrice AS UnitPrice
FROM
     Sales, SalesLine, Product
WHERE
     Sales.SalesNo=SalesLine.SalesNo
     AND Product.ProductCode=SalesLine.ProductCode
     AND Sales.SalesNo=1
```

1. Go to **File | New…**.
2. Select the **Simple Blue** template.
3. Press the **Launch Report Wizard** button.
4. Choose the **Report name** and **Location**. Here, the report name can be **SalesDetails**, then press **Next**.

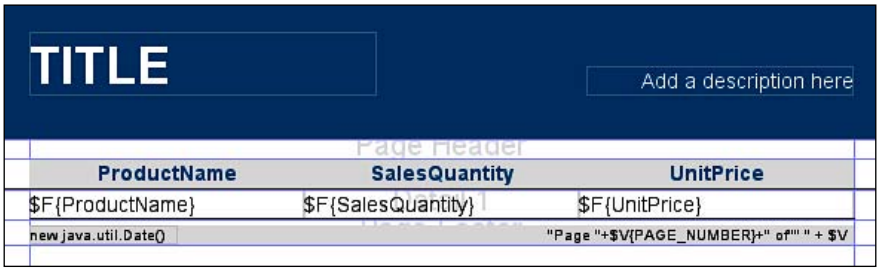
5. Select the **inventory** connection, write the SQL command, and press **Next >**.



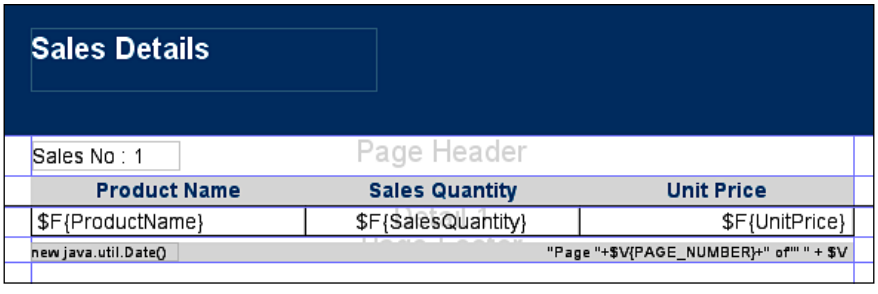
6. Select all fields and press **Next >**.



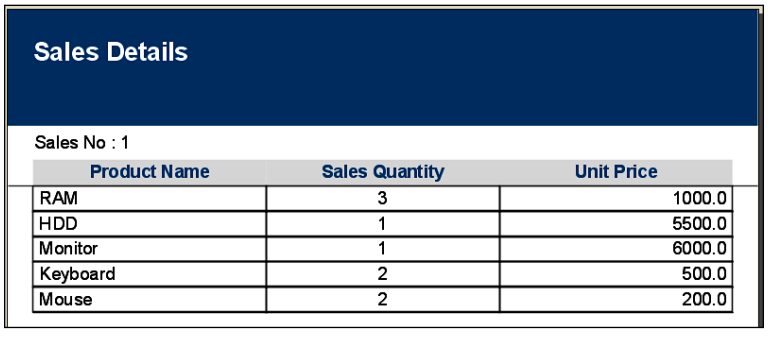
7. Again, press **Next >** without choosing any group in the **Group by…** window and press **Finish**.
8. The report designer window will open, and we will see that the design of the report is like the following one:



9. Make some changes in the design as you learnt in Chapter 3, *Report Layout and Formatting*. Change the **Title** to **Sales Details**, remove **Add a description here**, put a **Static Text** as **Sales No : 1** in **Page Header** band, give space between words in the field name, change the **Horizontal Alignment** of the fields as required from the **Properties** window, and give borders to the fields. Now, the report will look like the following one:



10. **Preview** the report to see the output, as shown in the following screenshot:



Adding variables

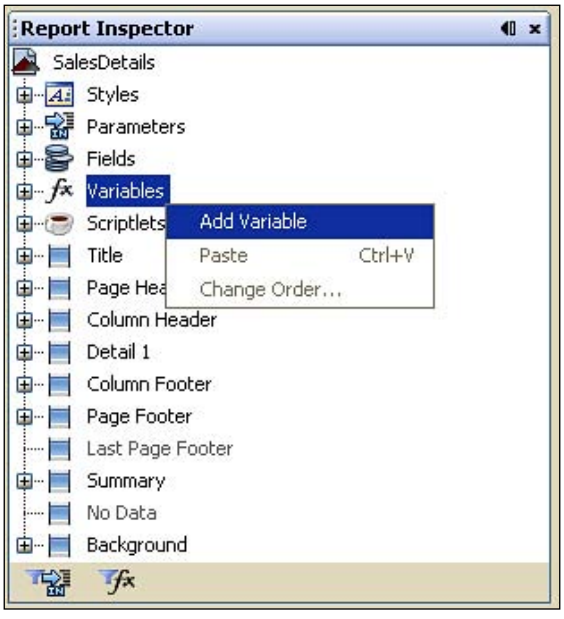
Before adding variables, we need to understand a few terms:

- **Variable name**: Each variable must have a name to identify it, in the places where it is required.
- **Variable class**: Each variable has a particular type, thus the particular Java type must be selected for the variable in iReport.
- **Calculation**: Variables can perform built-in calculations, such as sum, average, lowest, highest, standard deviation, variance, count, distinct count, and so on. If there are no built-in calculation types for an operation, we can build the expression using the wizard as we will do for our first variable "Total". However, for the grandtotal, we will use the built-in calculation type `sum`.
- **Reset type**: For variables, we can specify the reset type at which they are reinitialized. The default type is `Report`, which we will use for our variables "Total" and "GrandTotal". This reset type means that the variable is initialized only once at the beginning of the report, and that it performs the specified calculation until the end of the report execution. We can also choose a lower level of reset type for our variables to perform the calculation at the page, column, or group level. For example, if we want to calculate the total price in each report page, we would choose the page reset type.
- **Increment type**: If we want a variable value to increase automatically, we can choose the suitable increment type option. For example, a variable serial will increase its value by one after each table row. The available increment type options are: report, column, page, group, and none.
- **Variable expression**: Variable is used for various calculations. In its expression, a variable can have references to fields, other variables, parameters, a Java statement, and so on.

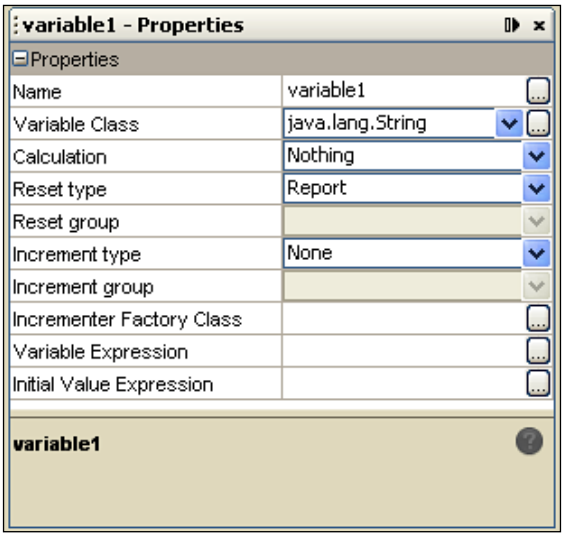
Now it's time to use variables in our report.

Adding total variable

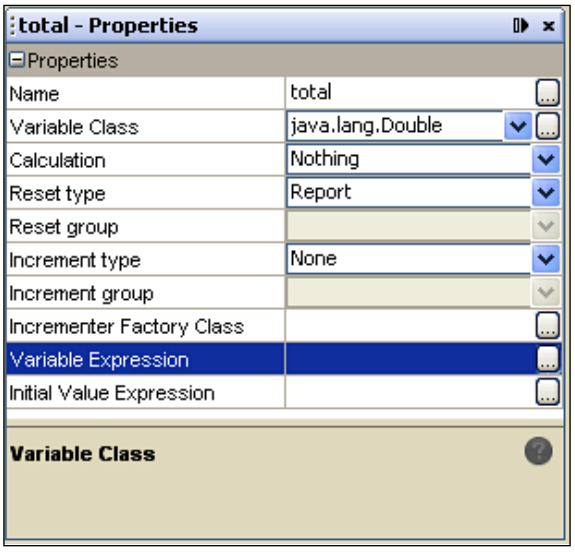
1. From the **Report Inspector** window, select **Variables**, right-click on it, and select **Add Variable**.



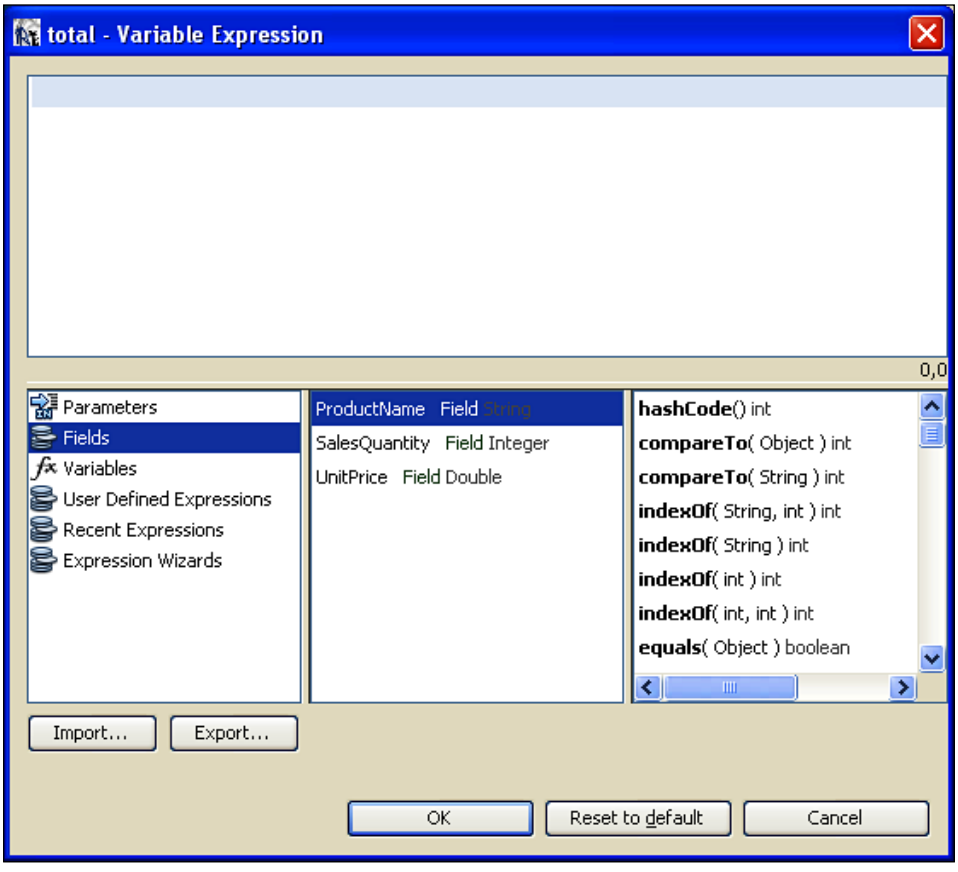
2. You will see a variable **Properties** window on the right of the designer window.



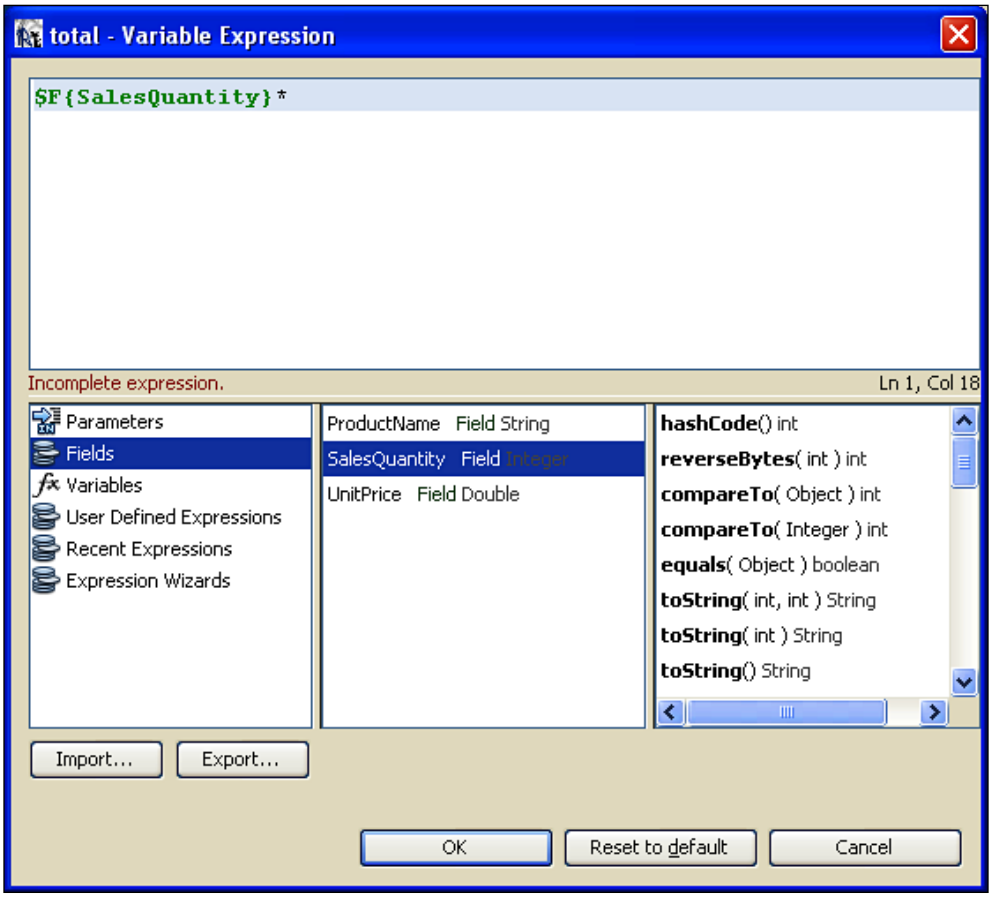
3. Enter **total** as the **Name**.
4. Select **java.lang.Double** as **Variable Class**. This is to note that we will multiply the quantity by unit price. As the data type of unit price is `double`, the result of multiplication will be of type `double`.
5. Choose **Nothing** as **Calculation**, as no built-in function is available here to multiply the fields.
6. **Reset type** is **Report**.
7. There is no group in this report. Hence **Reset group** and **Increment group** are disabled here.
8. **Increment type** should be **None**, as the **total** variable will not be increased automatically.



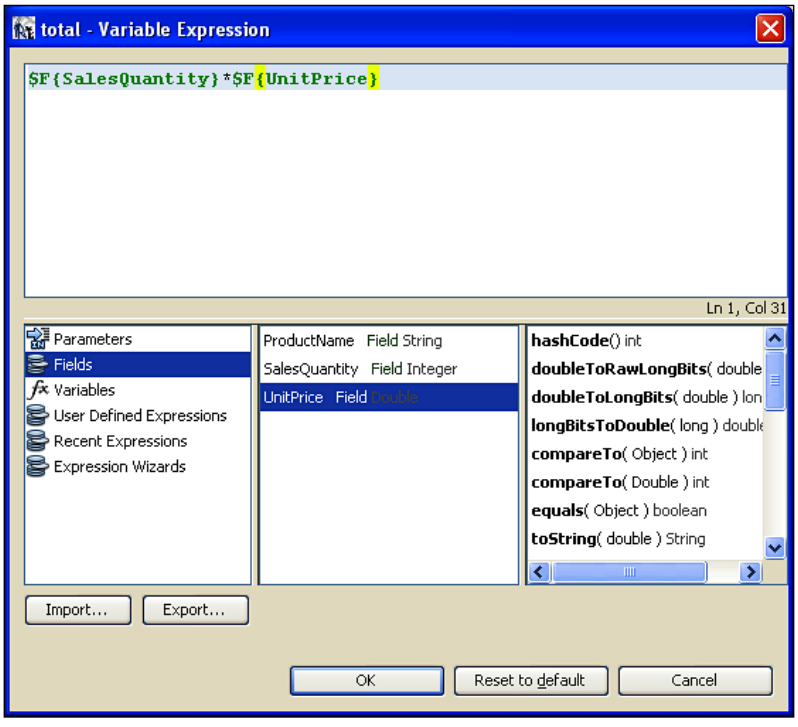
9. Now, we will create the variable expression; actually this is where we will multiply the columns. Click on the button near **Variable Expression** to open the custom editor.



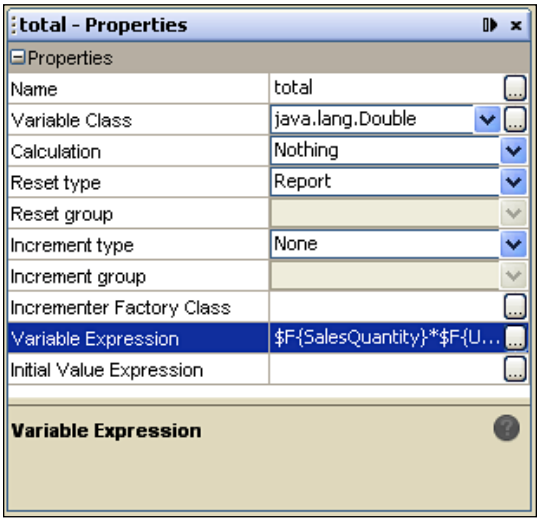
10. Select **Fields** | **SalesQuantity,** double-click on it, and insert a * (multiply) symbol.



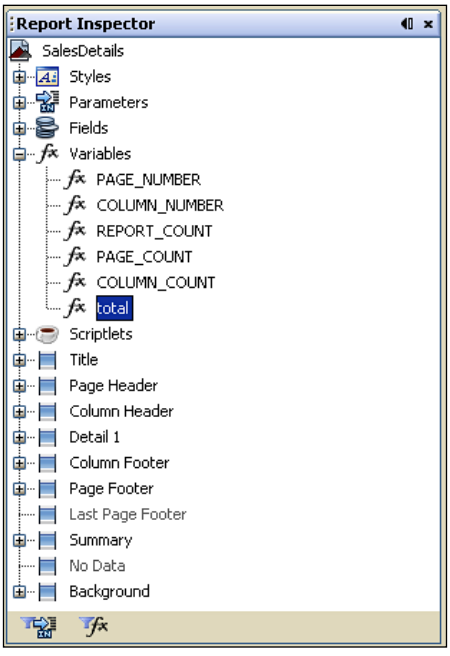
11. Again, select **UnitPrice**, and double-click on it. See that a line of code—`$F{SalesQuantity}*$F{UnitPrice}`—has been created above.



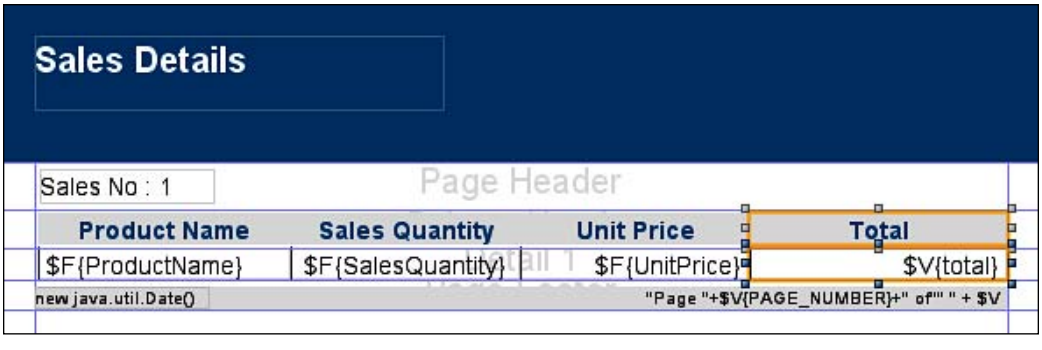
12. Press **OK**.



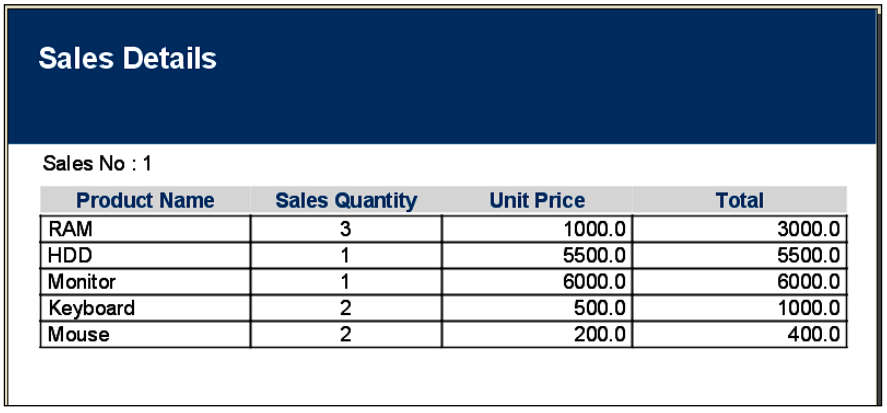
13. See that **total** is listed as a variable in the **Report Inspector**.



14 Now, we need to modify the design of the report so that we can place the **total** variable on the report. Select all the column headers and fields on the report design and decrease the size and position all the elements on the left side of the report, so that we can place a column header and **total** variable, as shown in the following screenshot. For the column header **Total**, copy **Unit Price** and paste it, and then position it by modifying the text. In order to place the variable, drag the **total** variable to the report from the **Report Inspector**. Remember to set the same font for the variable **total**. Change the **Horizontal Alignment** of **total** to **Right** from the **Properties** window.



15. **Preview** the report to see the output, as shown in the following screenshot:



Sales Details

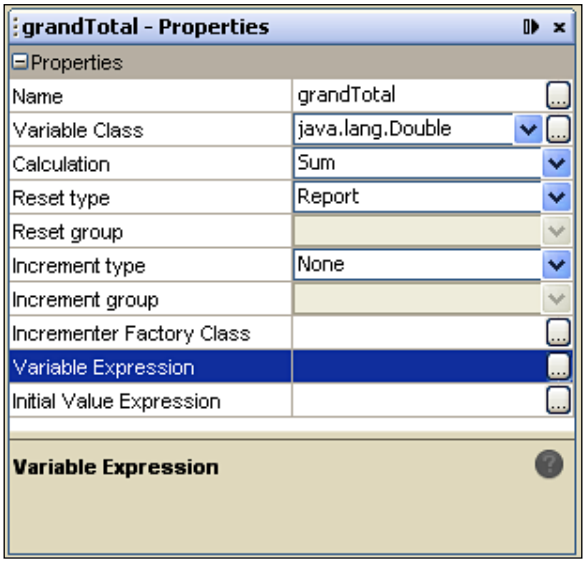
Sales No : 1

Product Name	Sales Quantity	Unit Price	Total
RAM	3	1000.0	3000.0
HDD	1	5500.0	5500.0
Monitor	1	6000.0	6000.0
Keyboard	2	500.0	1000.0
Mouse	2	200.0	400.0

Adding a grand total

Now, we will add another variable, **grandTotal**, which is the sum of all totals, and will be displayed at the bottom of the report.

1. Again, select **Variables** in the **Report Inspector** window, right-click on it, and choose **Add Variable**.
2. Enter the variable **Name** as **grandTotal**, select **java.lang.Double** from the **Variable Class** options, and select **Sum** from the **Calculation** options.



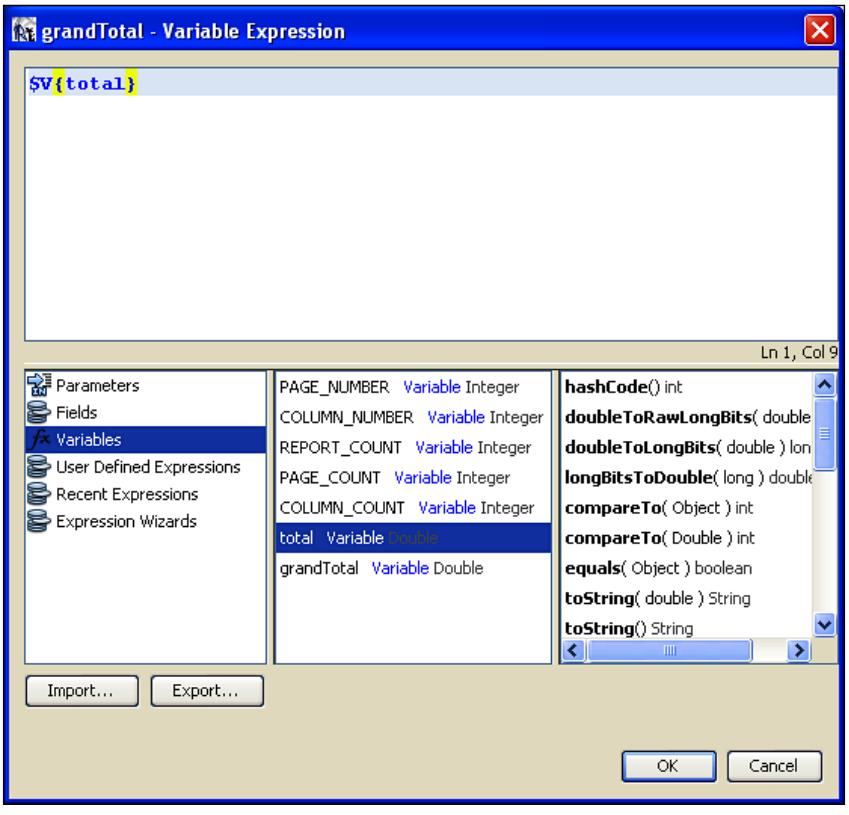
grandTotal - Properties

Properties

Name	grandTotal
Variable Class	java.lang.Double
Calculation	Sum
Reset type	Report
Reset group	
Increment type	None
Increment group	
Incrementer Factory Class	
Variable Expression	
Initial Value Expression	

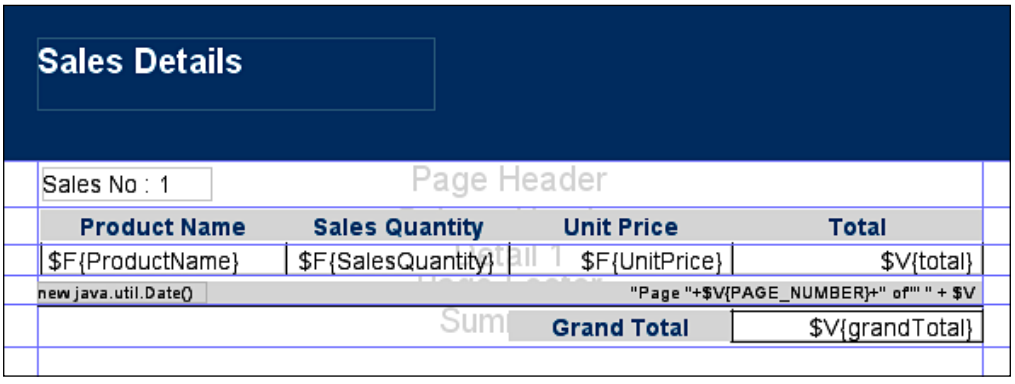
Variable Expression

3. Open the **Variable Expression** editor. Now select **Variables** from the **Objects and expressions** section and just double-click on **total Variable**.

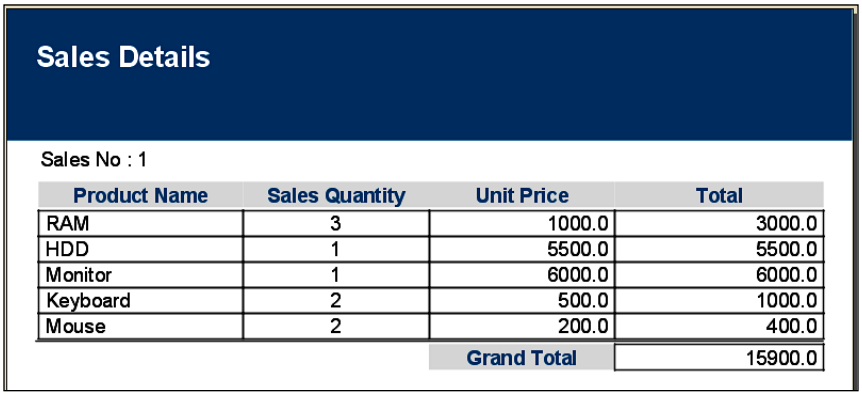


4. Press **OK**.
5. Set **25** as the **Band height** for **Summary** band, so that we can place the `grandTotal` variable in the **Summary** band. To understand bands and how to change the band height, refer Chapter 3.
6. Place a line and a static text **Grand Total** on the **Summary** band.

7. Drag the **grandTotal** variable from the **Variables** dialog box to the summary band. Now the report design looks like the following screenshot:



8. **Preview** the report, and you will see the output, as shown in the following screenshot:



Sales Details

Sales No : 1

Product Name	Sales Quantity	Unit Price	Total
RAM	3	1000.0	3000.0
HDD	1	5500.0	5500.0
Monitor	1	6000.0	6000.0
Keyboard	2	500.0	1000.0
Mouse	2	200.0	400.0
		Grand Total	15900.0

Summary

In this chapter, we learned about variables. Specifically, we have covered:

- How to add variables
- How to define the expressions
- How to show variables in a report
- The significance of the variable `Reset` type and `Increment` type.

Now that we've learned about building reports, designing reports, and variables, we're ready to use parameters, which is the topic of the next chapter.

5
Using Parameters

In the previous chapter, we created a `SalesDetails` report, which shows the **Product Name**, **Unit Price**, **Sales Quantity**, **Total**, and **Grand Total** of **Sales No : 1** (One). This report shows the data of only Sales No. 1. So what will we do if we want to see the details of Sales No. 2, or 3, or others? Of course, we will not develop reports for each Sales No. because thousands of Sales Nos. will be generated. We will use a technique where we will develop only one report, which will work for any Sales No. In the `SalesDetails` report of Chapter 4, *Using Variables,* we hardcoded the `SalesNo` in the SQL commands. That's why it works only for Sales No. 1. In this chapter, we will use parameters instead of hardcoded data in the report. Recall that we wrote the following SQL query in the `SalesDetails` report:

```
SELECT
  Product.Name AS ProductName,
  SalesLine.SalesQuantity AS SalesQuantity,
  SalesLine.UnitSalesPrice AS UnitPrice
FROM
  Sales, SalesLine, Product
WHERE
  Sales.SalesNo=SalesLine.SalesNo
  AND Product.ProductCode=SalesLine.ProductCode
  AND Sales.SalesNo=1
```

In this chapter, we will use a parameter instead of `1` as in the previous code. We will just modify the previous report by replacing the static `1` (one) by a parameter. Specifically, in this chapter, we shall learn about the following:

- Necessity of parameters
- Adding/modifying parameters
- Modifying the SQL query for using the parameters

So let's get on with it.

What is a parameter?

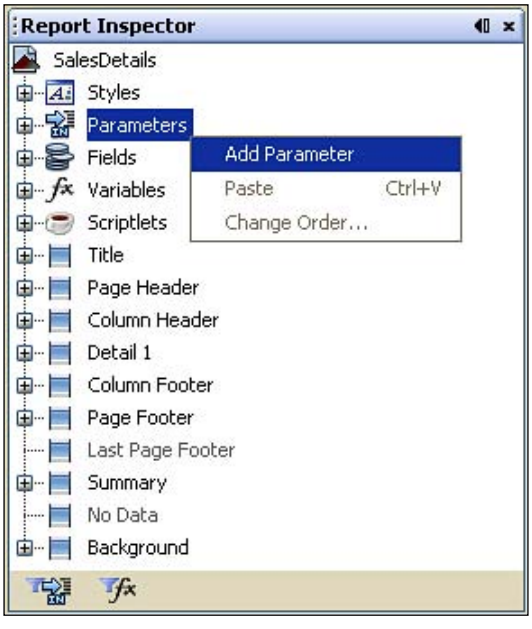
Parameters are types of report objects, which are used just for passing data to the report engine. The data may be passed from another report, from any program, or directly from the user. The passed data is then used for filling the report dynamically. Parameters are very useful for passing data to the report that cannot normally be found in the report data source. For example, we can dynamically change the title of the report. An important aspect is the use of parameters in the query string of the report in order to be able to further customize the data retrieved from the database. Actually, parameters act as dynamic filters in the query that supplies data for the report.

Adding parameters in the SalesDetails report

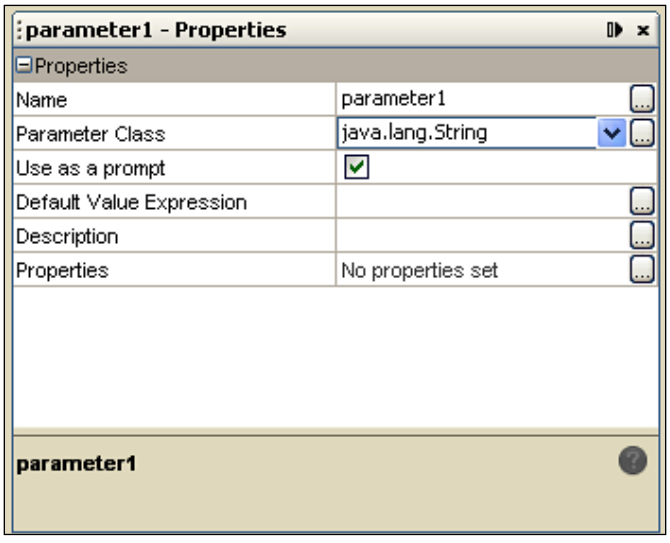
Adding parameters is the first thing to do when using parameters in a report. We need to name each parameter of a report. This can be any valid identifier.

We are going to add a parameter in our `SalesDetails.jrxml` report so that it can show data dynamically based on a user's input. When the report is executed, the user will give a Sales No. as input, and the report will show details of the given Sales No.

1. Open the `SalesDetails.jrxml` report, which you developed in Chapter 4.
2. Go to **Report Inspector**, select **Parameters**, right-click on it, and choose **Add Parameter**.



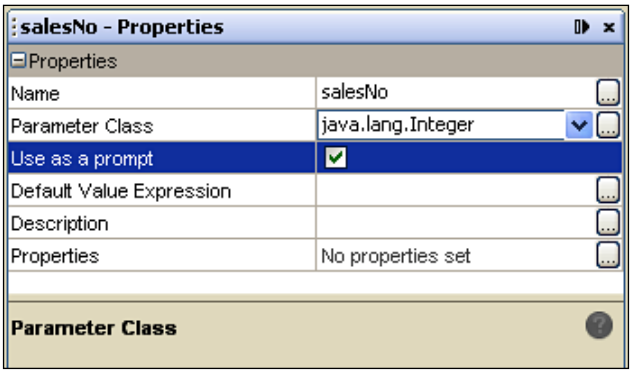
3. A parameter is added, and the **Properties** window is visible on the right side of the designer.



4. Enter **salesNo** as the parameter **Name**.
5. Select **java.lang.Integer** as **Parameter Class**.

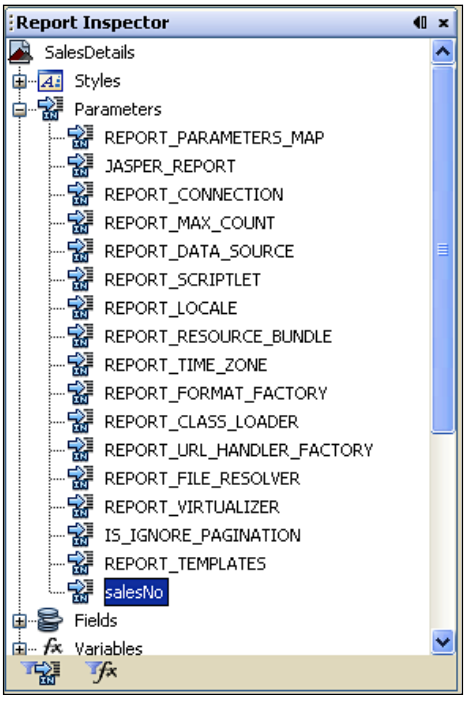
The parameter class type depends on the database attribute type. You need to choose the corresponding Java type for the attribute type. For example, if the attribute type is `varchar` in the database, then you have to choose **java.lang.String** as the parameter class type.

6. Check the **Use as a prompt** checkbox.



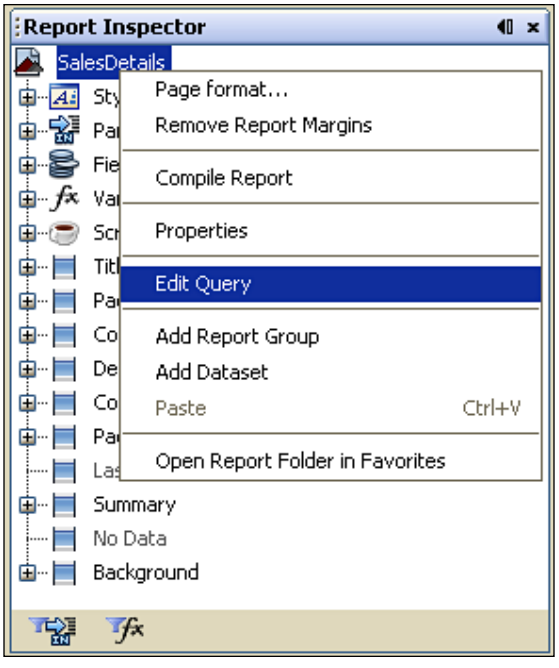
If you want the user to input the parameter, then you have to check the **Use as a prompt** checkbox. If the parameter is passed from other fields, variables, or other reports, then this checkbox should not be checked. In that case, you can set a default value (if no value is passed, the default value will be used for filling the report). If you want to set `1` as the default value for the `SalesNo`, write `new java.lang.Integer(1)` in the **Default Value Expression** area. This is a Java expression. If the report language is **Groovy**, then you can just write **1** as an expression.

7. Now, you can see that a new parameter **salesNo** is added in the **Report Inspector** | **Parameters** section as shown in the following screenshot:



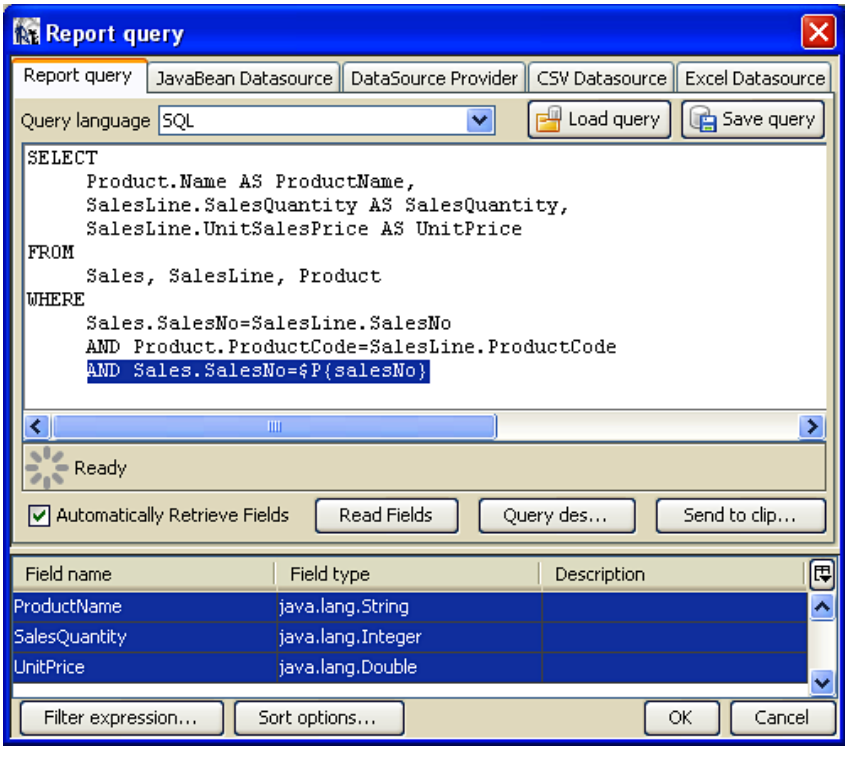
If you want to modify the parameter name, class type, the prompt option, or the default value option, go to **Report Inspector** | **Parameters** and double-click on the parameter from the list.

8. Now we will modify the SQL query. Select **SalesDetails** from the **Report Inspector**, right-click on it, and select **Edit Query**.



9. Replace the query with the following one:

```
SELECT
  Product.Name AS ProductName,
  SalesLine.SalesQuantity AS SalesQuantity,
  SalesLine.UnitSalesPrice AS UnitPrice
FROM
  Sales, SalesLine, Product
WHERE
  Sales.SalesNo=SalesLine.SalesNo
  AND Product.ProductCode=SalesLine.ProductCode
  AND Sales.SalesNo=$P{salesNo}
```



If you don't use the added parameter in your report query, the report data will not have any effect. Use the parameter in the `WHERE` clause of the SQL commands to filter the query result based on the parameter value.

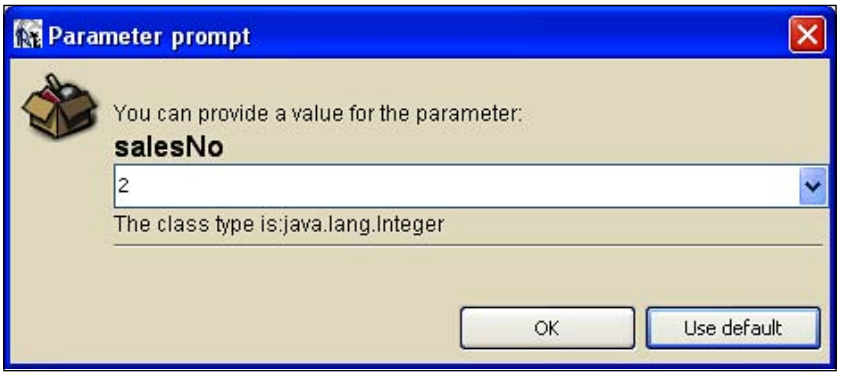
We have deleted `1` from the last line of the query. Instead, added `$P{salesNo}`.

`$P{salesNo}` is case sensitive. `$P` stands for parameter and `salesNo` within the curly bracket is the parameter name which you give when you are adding a parameter.

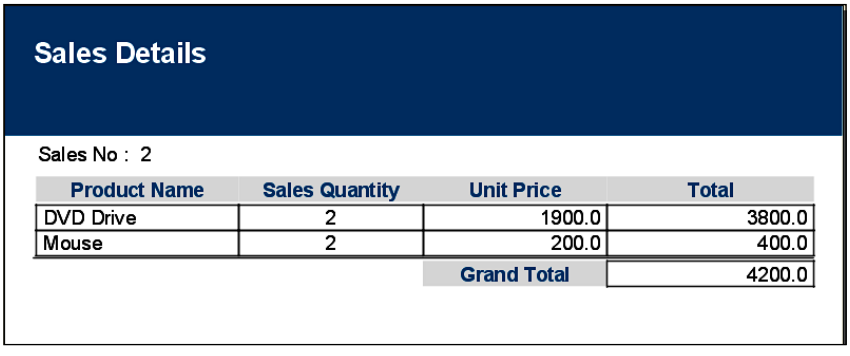
10. Press **OK**.
11. Delete **1** from the static text **Sales No : 1**.

12. Drag the **salesNo** from the parameter list of the **Report Inspector** into the report beside the static text **Sales No :**.

13. **Preview** the report with an active connection. It will prompt you for the input of **salesNo**.



14. Input **2** as the **salesNo**, and press **OK**. It will show the details of **Sales No : 2**, as shown in the following screenshot:



Sales Details

Sales No : 2

Product Name	Sales Quantity	Unit Price	Total
DVD Drive	2	1900.0	3800.0
Mouse	2	200.0	400.0
		Grand Total	4200.0

Using more than one parameter

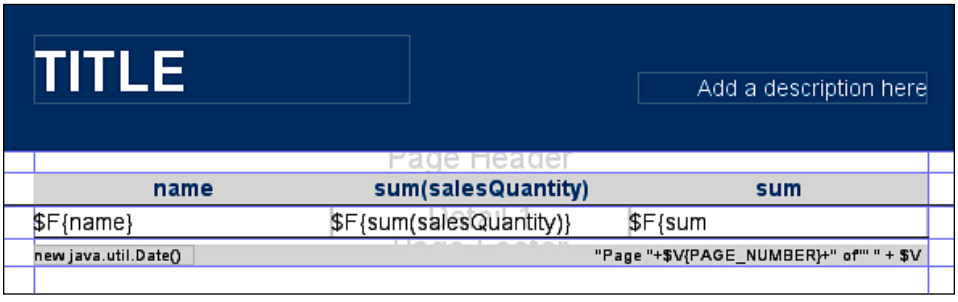
We are going to produce another report that uses two parameters of the `java.util.Date` type. More often, we need to produce daily, weekly, monthly, or yearly reports. A single report, which we are going to produce now, can be used for all periodical reports.

Here we will produce a report, which shows the sales summary data between two dates. Just follow the listed steps:

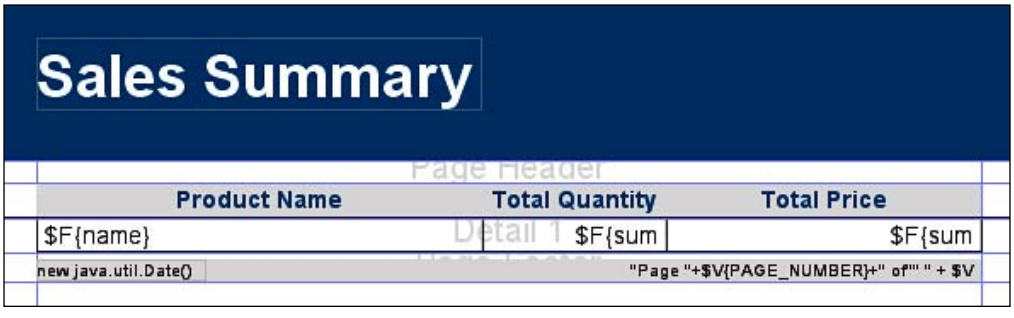
1. Create a report as previously done in Chapter 2, *Building Your First Report,* using the following SQL command in the **SQL query** section:

```
SELECT name,sum(salesQuantity), sum(unitSalesPrice*salesQuantity)
FROM Product, Sales, SalesLine
WHERE Product.productCode=SalesLine.productCode
AND Sales.salesNo=SalesLine.salesNo
GROUP BY name
```

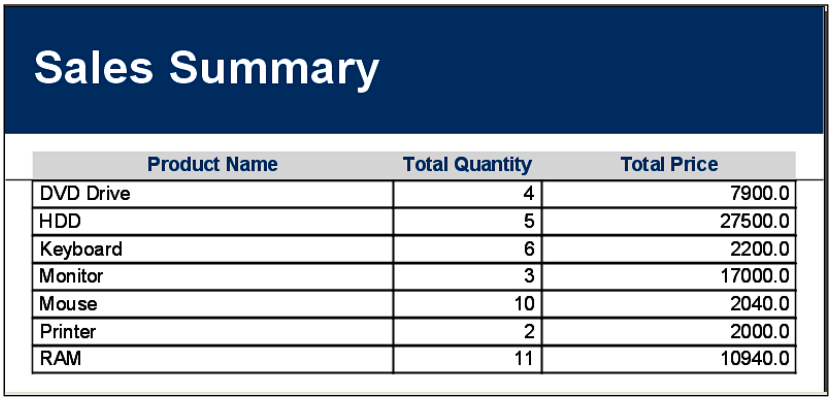
2. Now, the report design looks like the following screenshot:



3. Change the report title to **Sales Summary** and remove the **Add a description here** field.
4. Change the column headings to **Product Name**, **Total Quantity**, and **Total Price**. Change the **Horizontal Alignment** of quantity and price to **Right** from the **Properties** window. Give borders to the fields. Now the report design looks like the following screenshot:



5. **Preview** the report with an active connection and see the output, which is similar to the following screenshot. The report shows all the database data without filtering the data according to date.

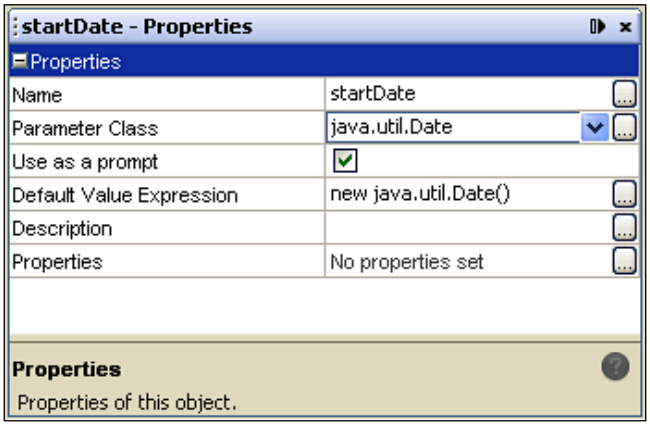


Sales Summary

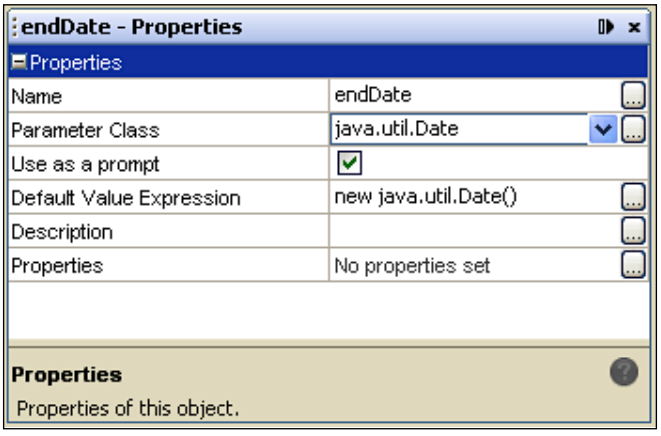
Product Name	Total Quantity	Total Price
DVD Drive	4	7900.0
HDD	5	27500.0
Keyboard	6	2200.0
Monitor	3	17000.0
Mouse	10	2040.0
Printer	2	2000.0
RAM	11	10940.0

Now, we want to filter the data date-wise, that is, the start date and end date will be given as input, and the report will show data of sales between the start date and end date. For this, we need to create parameters. Create two parameters following the steps previously listed.

1. For the first parameter, give **startDate** as the parameter **Name**, choose **java.util.Date** as the **Parameter Class** type, and check the **Use as a prompt** checkbox.

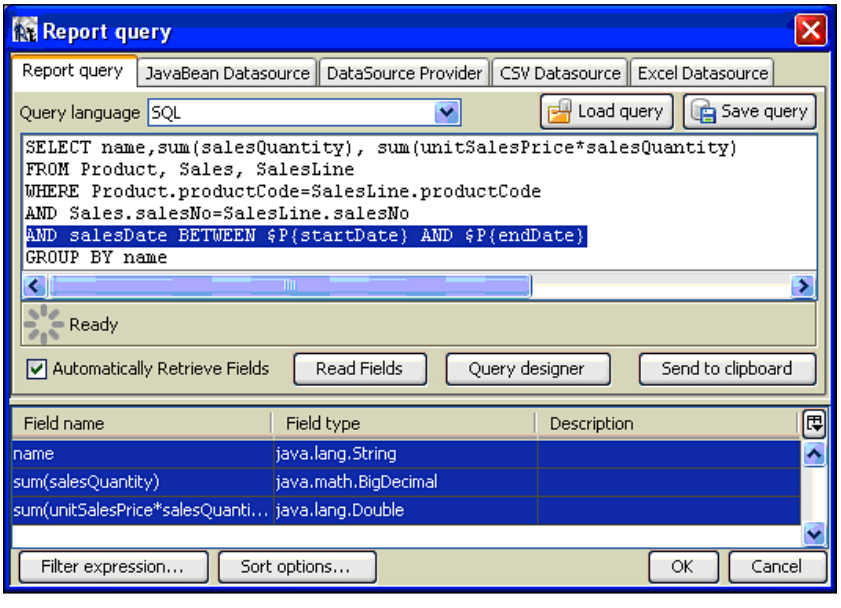


2. In the same way, create another variable named **endDate**.

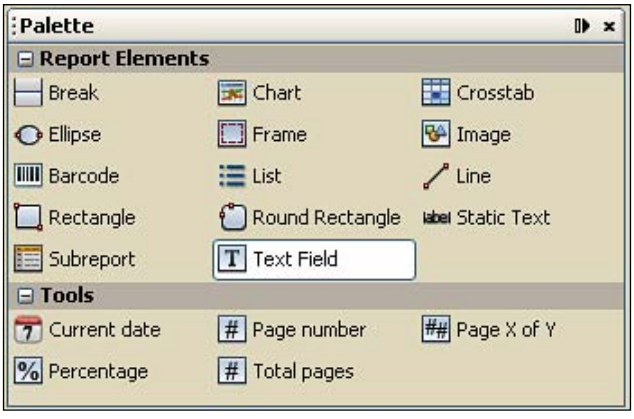


3. Now, you need to modify the query to filter data using the **Edit Query** option. Replace the query with the following one:

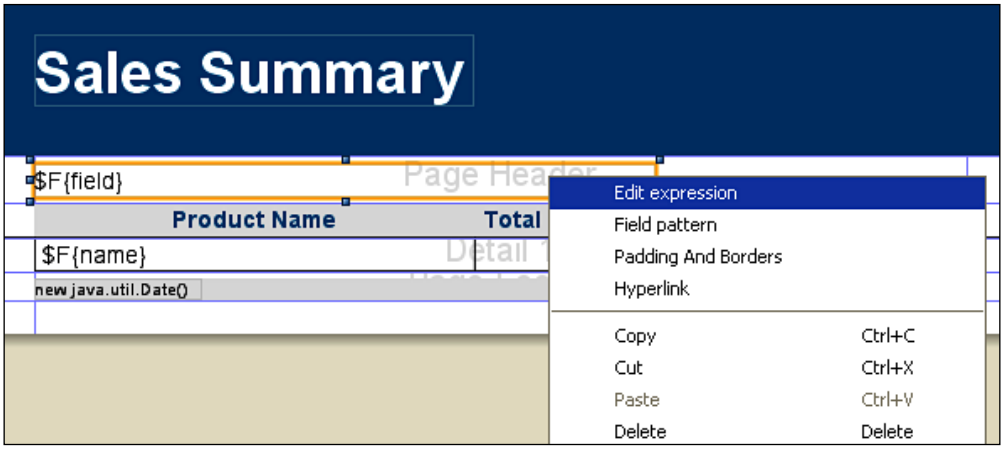
```
SELECT name,sum(salesQuantity), sum(unitSalesPrice*salesQuantity)
FROM Product, Sales, SalesLine
WHERE Product.productCode=SalesLine.productCode
AND Sales.salesNo=SalesLine.salesNo
AND salesDate BETWEEN $P{startDate} AND $P{endDate}
GROUP BY name
```



4. Now, we will display the value of the parameters **startDate** and **endDate** in the **Page Header** band. Increase the page header band height appropriately.
5. From the **Palette**, drag a **Text Field** and drop it on the **Page Header** band.

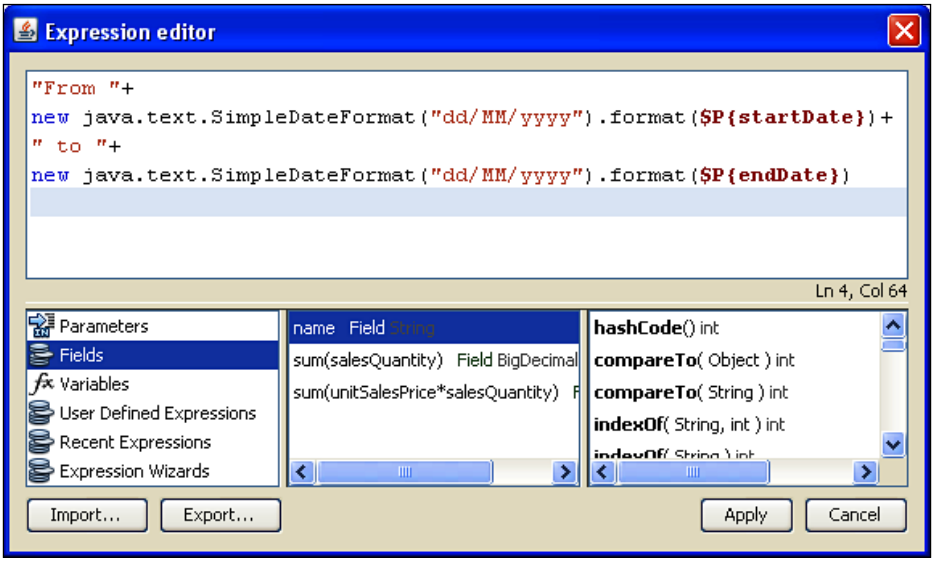


6. Select the text field, right-click on it, and then press **Edit expression**. The **Expression editor** will open.



7. Now, write the following code in the **Expression editor**:

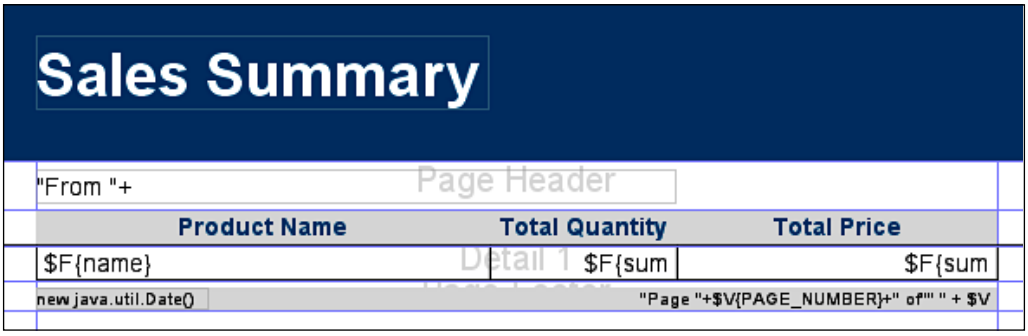
```
"From "+
new java.text.SimpleDateFormat("dd/MM/yyyy").format($P{startDate})+
" to "+
new java.text.SimpleDateFormat("dd/MM/yyyy").format($P{endDate})
```



8. Press **Apply**.

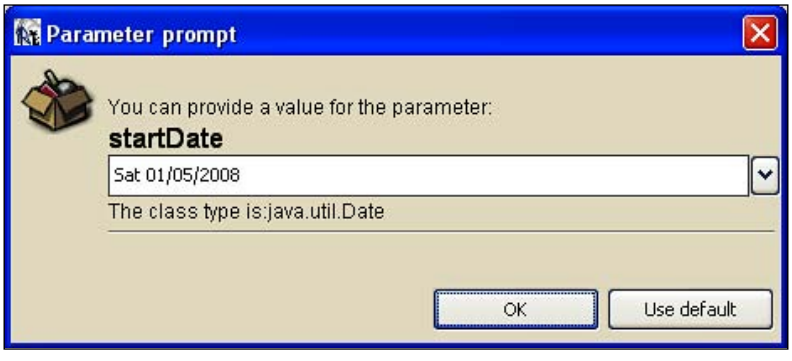
Here, we dragged a **Text Field** on the **Page Header** band and edited the text field expression. Actually, we need to show a total of four elements on the **Page Header** band, which are: a static text **From**, the value of the parameter **startDate**, another static text **to**, and the value of the parameter **endDate**. **Text Field** is a suitable element to show more than one element together. We can add several strings in the text field expression. In our case, we first added the string, `From`, then added the start date after formatting it as `dd/MM/yyyy`, and then added another string, `to`, and the end date respectively. For formatting the date parameter, we used the method format of Java class `java.text.SimpleDateFormat`.

9. Now, the report design looks like the following screenshot:

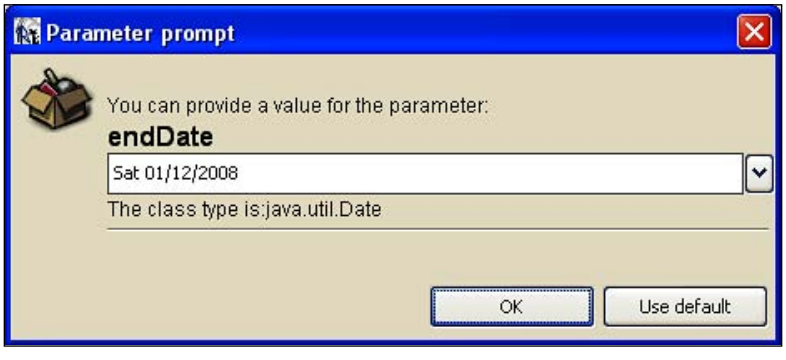


10. **Preview** the report with an active connection.

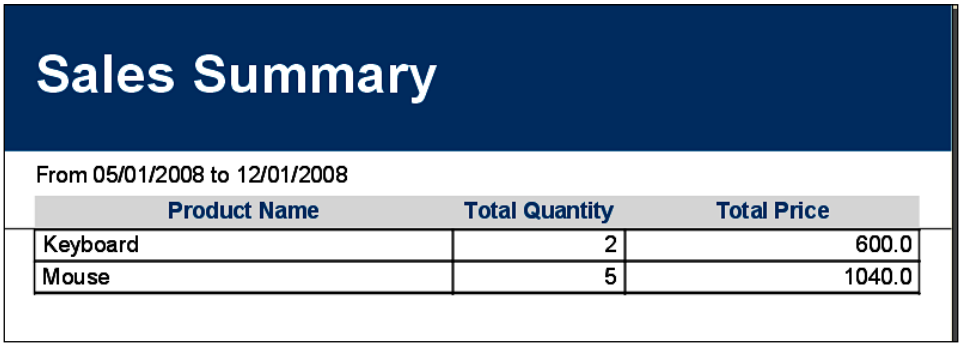
    Enter the **StartDate**:



    Enter the **endDate**:



The preview will be as shown in the following screenshot:



Summary

We learned a lot in this chapter about parameterized reports.

Specifically, we have covered:

- The necessity of report parameters
- Adding parameters
- Modifying a report query
- Displaying parameters in the report
- Modifying parameters

Now, it's time to learn about report groups, which is the topic of the next chapter.

6
Grouping Data in Reports

Report groups are a flexible way to show grouped data based on one or more certain fields, or even on generic expression, that is, a group can be defined based on the first letter of the employees. "List of Employees by Department" is another example of grouping employee data by department. The report may look like the following one:

Department: Sales

Employee ID	Employee Name
1001	John Smith
1002	Andrew Simmonds
1005	James

Department: Production

Employee ID	Employee Name
1003	Leong Chu
1004	Mike
1006	Collin
1007	Stoover

Department: Accounts

Employee ID	Employee Name
1008	Mark
1009	Sharif Hasan

In this chapter, we will cover the following topics:

- How to build a **Group by** report
- Understanding report groups

So let's get on with it.

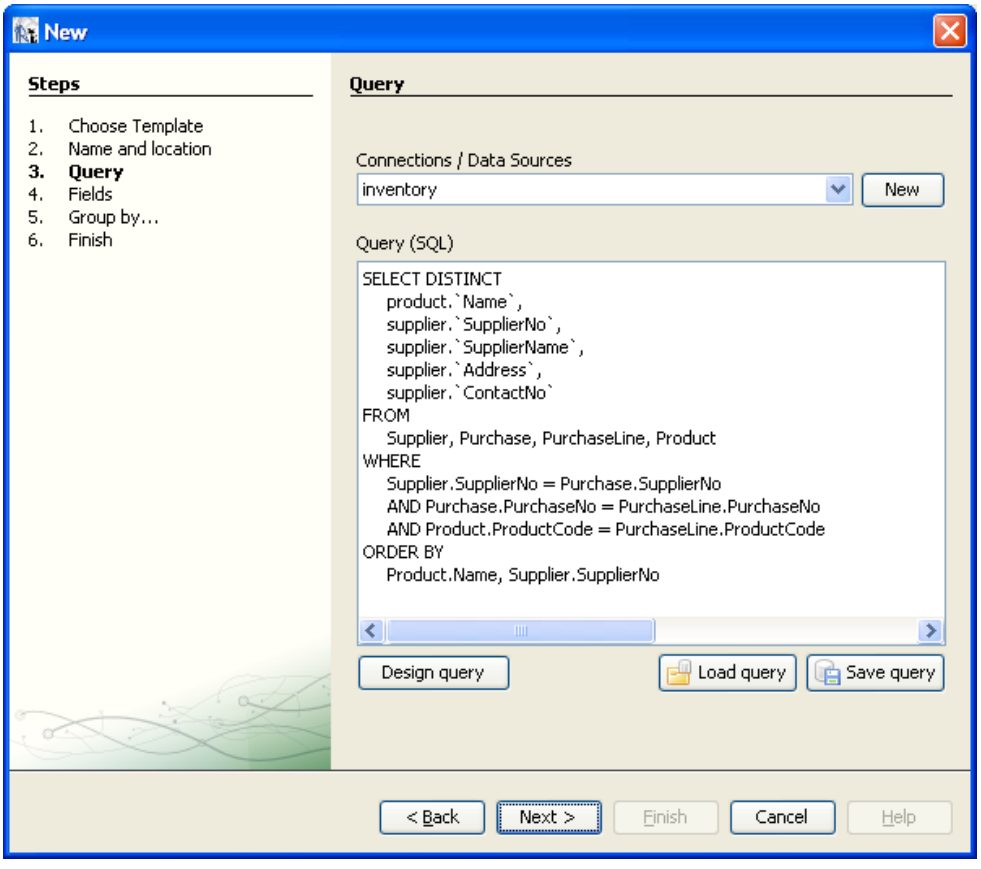
Building a Group by report

Here we will build a report to show the list of suppliers of each product. The suppliers of a particular product will be grouped together, thus the report will have a view of all the suppliers of a particular product, that is, the manger (suppose the report is for a manager) will know which product is supplied by which supplier.

Follow the listed steps:

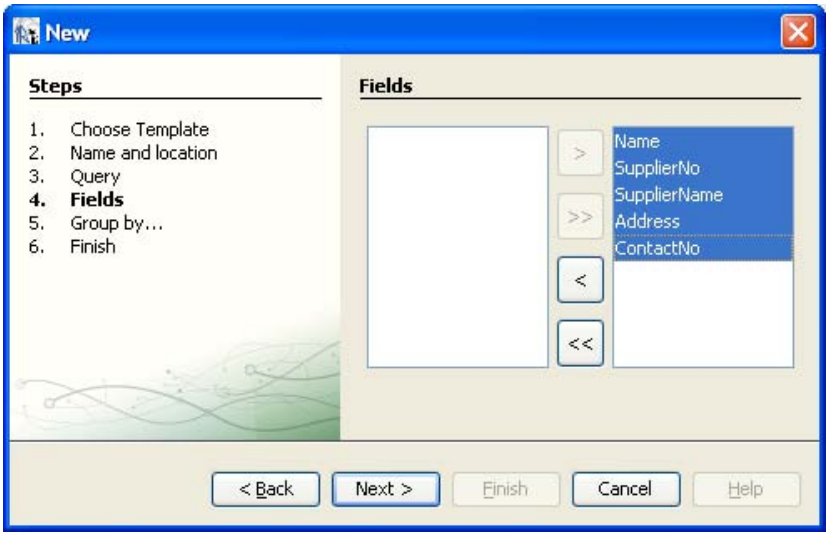
1. Go to **File** | **New...** | **Report**, select **Simple Blue**, and press **Launch Report Wizard**.
2. Name the report as **SupplierProduct**, and press **Next >**.
3. Write the SQL query in the following way:

```
SELECT DISTINCT
     product.`Name`,
     supplier.`SupplierNo`,
     supplier.`SupplierName`,
     supplier.`Address`,
     supplier.`ContactNo`
FROM
     Supplier, Purchase, PurchaseLine, Product
WHERE
     Supplier.SupplierNo = Purchase.SupplierNo
     AND Purchase.PurchaseNo = PurchaseLine.PurchaseNo
     AND Product.ProductCode = PurchaseLine.ProductCode
ORDER BY
     Product.Name, Supplier.SupplierNo
```

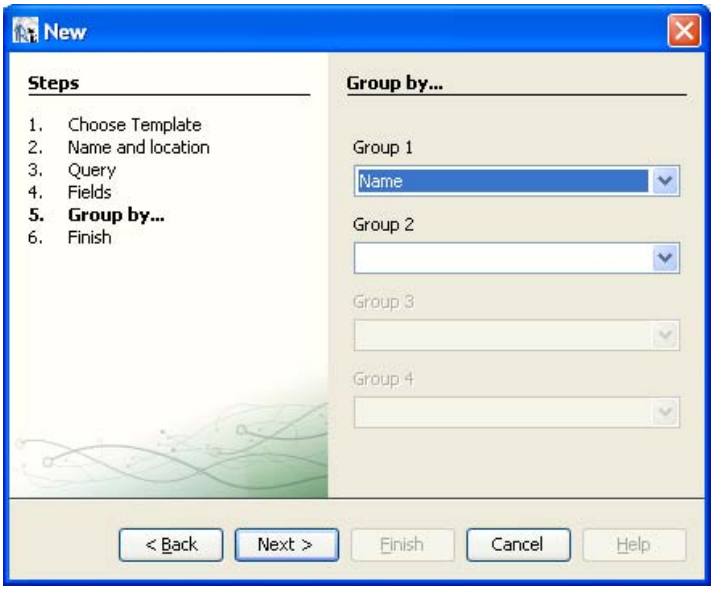


iReport doesn't perform any command on data by default, so if you group data by fields, and if the values for these fields are A B A B, then four groups are created against the expected two (one for each record with the field value set to A, and one for the record numbers 2 and 3, with the field value set to B). If you present iReport the ordered data (A A B B), then all will work fine. That's why we have applied the `ORDER BY` clause in the SQL query. Note that sorting by field can be performed by JasperReports. Anyway, it is not suggested when working with a large amount of data. If SQL is used, then the `ORDER BY` clause is for sure the better way to sort data.

4. Press **Next >**.



5. Select all the fields, and press **Next >**.
6. Select **Name** from the **Group 1** options.



7. Press **Next >**, and then press **Finish**.
8. Now, you will see the report design, as shown in the following screenshot:

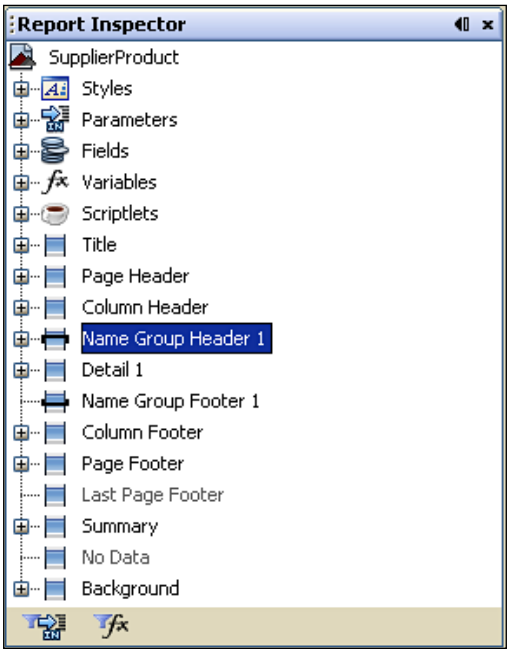
9. **Preview** the report with an active connection, and see the following output:

Name	**Mouse**			
1		Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
Name	**Printer**			
1		Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
2		ABC Supplies	Dhanmondi, Dhaka	0234567891
Name	**RAM**			
1		Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
2		ABC Supplies	Dhanmondi, Dhaka	0234567891
3		XYZ Company	52 Gabtali, Dhaka	0345678912
4		Smart Computers Ltd.	New Elephant Road, Dhaka	0456789123

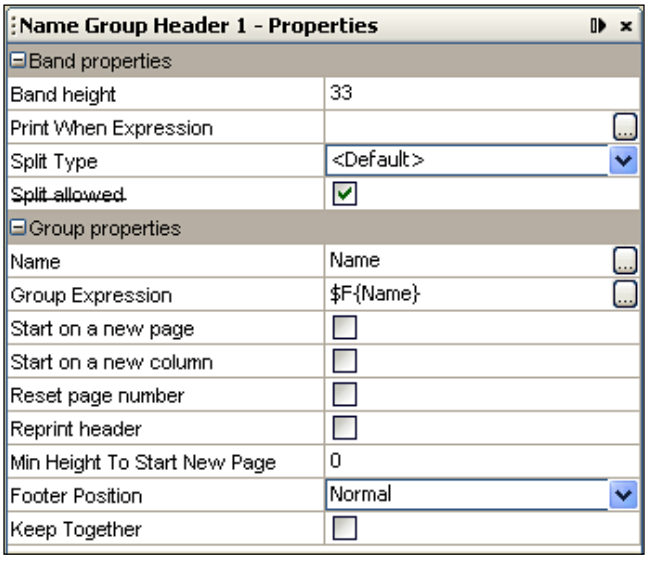
We have just created a **Group by** report where all the suppliers are grouped by product. Using this kind of representation gives us an overall view of a particular data item. For example, in this report, we get all the supplier details of a particular product together. This is the product view of the product–supplier relationships. If we present the entire product list of a particular supplier, then it will be a supplier view of the product–supplier relationship.

Modifying group properties

To modify the group properties, select **Name Group Header 1** in the **Report Inspector**, and see the group **Properties** on the right side of the designer window.



The group **Properties** window is as shown in the following screenshot:



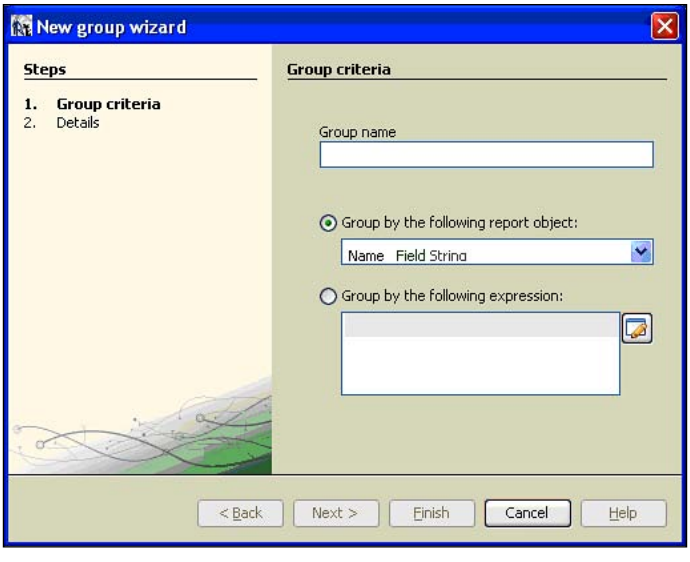
The following options are available:

Option	Description
Name	This is the name of the group. You can modify the name by just clicking on the button next to the name.
Group Expression	For grouping the records, the expression is defined here. The expression may be the value of a field or a more complex expression.
Start on a new page	If this checkbox is checked, then the group starts on a new page.
Start on a new column	If this checkbox is checked, then the group starts on a new column.
Reset page number	When the group changes, the page number is reset if this option is checked.
Reprint header	If you want to reprint the group header on each page, check this option.
Min Height to Start New Page	If the value is greater than **0** (zero), it is considered as the minimum height required to keep the group on the current page.
Footer Position	This option allows us to specify where to place the group footer. The available place options are **Normal**, **Stack at bottom**, **Force at bottom**, and **Collate at bottom**.
Keep Together	This is a flag that prevents the group from splitting on two separate pages/columns.

Managing report groups

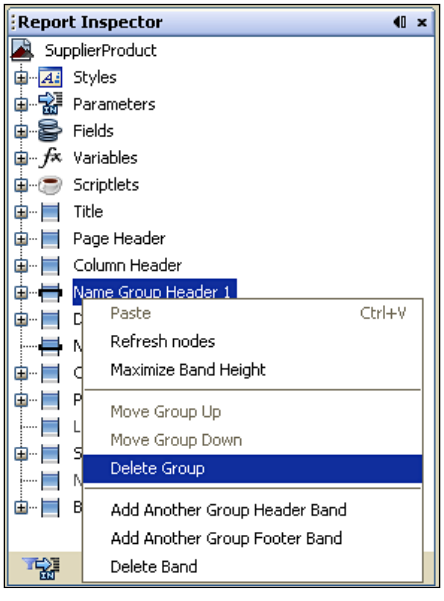
If you need to group data, then you can add one or more groups. You have already seen that you can use the Report Wizard to define the **Group by** fields. You can also include new groups in a report without the wizard. For this, select the report from the **Report Inspector**, right-click on it, and press **Add Report Group**.

You will see a dialog box like the one in the following screenshot:



Now, give a group name, select the report object for grouping data or define an expression. If we had to define the report groups that we have done previously without the Report Wizard, then we would name the group as **Name**, select the report object **Name Field**, and press **Next >**. In the next dialog, you will see options for adding a group header and footer.

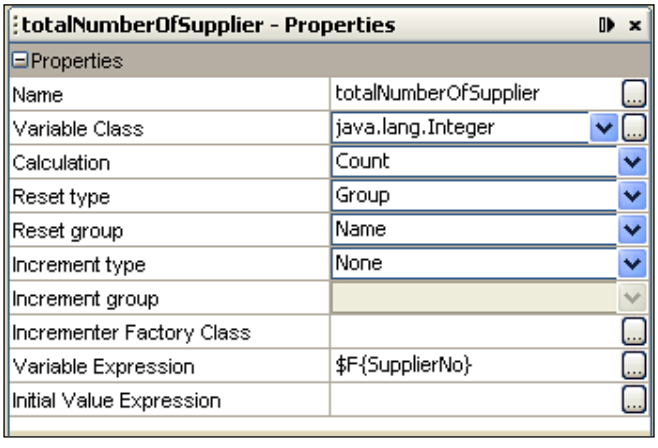
You can have as many groups as you want in a report. The order of groups declared in a report design is important because groups contain each other. One group contains the following group, and so on. **Move Group Up** and **Move Group Down** are applicable when there are more than one report group in a report. You can also remove the groups from the report. You can get the move up, move down, and delete options for a group by right-clicking on the group in the **Report Inspector**.



Variables for the group

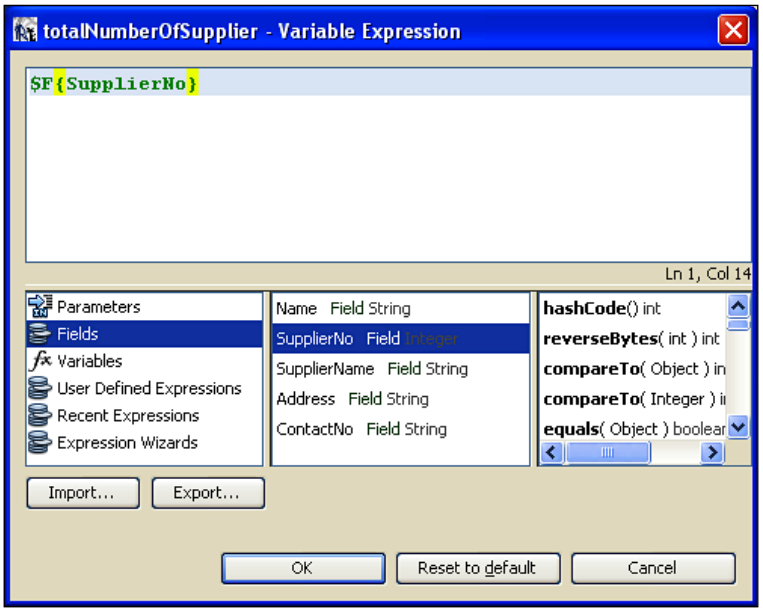
In Chapter 4, *Using Variables*, we learnt about the use of variables. In that chapter, we saw that a variable contains a particular value (maybe after some calculations) and that value remains the same for the whole report during execution time. However, we can use variables that can hold different values depending on the group. Actually, the variable holds a value for a particular group and the value is reset if the group is changed. We will add such a variable in our report. Suppose we want to show the number of suppliers for each product after the supplier details are shown. Just follow the listed steps:

1. Create a variable, and set the properties as follows:

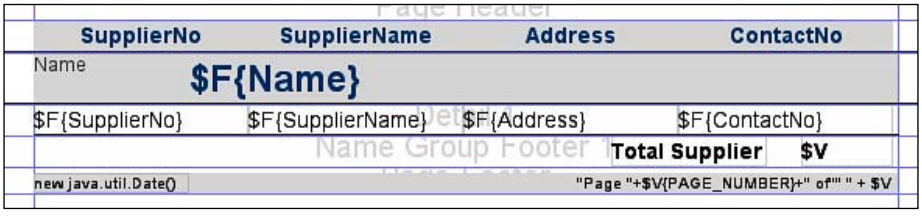


2. Enter **totalNumberOfSupplier** as the variable name.
3. Select **java.lang.Integer** as the **Variable Class** type.
4. Select **Count** from the **Calculation** options, as we will actually count the number of suppliers.
5. Set **Reset type** to **Group**. Here, **Name** is selected as **Reset group** as we have only one group in this variable. We have done this because the variable's value will change if the group is changed.
6. Open the variable expression editor by clicking on the button on the right side of **Variable Expression**.

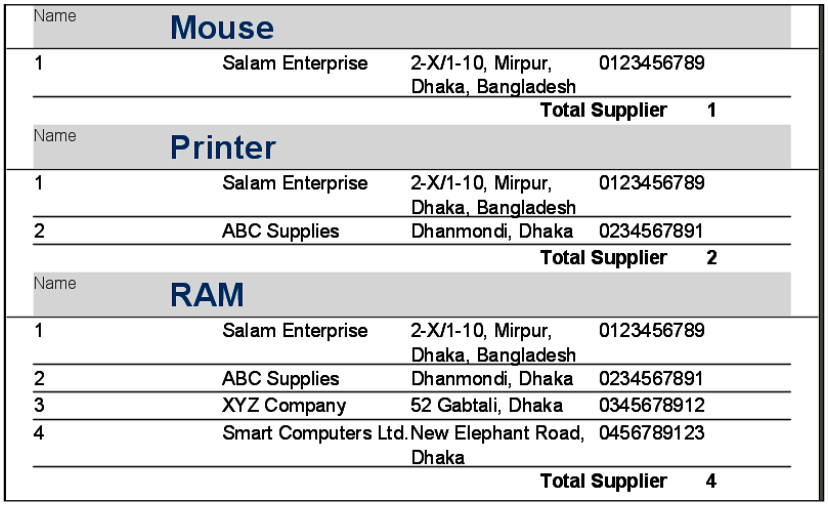
7. Double-click on the **SupplierNo** field as we will count the total number of suppliers.



8. Press **OK**. The expression editor will disappear, and the **Variable Expression** will be set.
9. Go to **Name Group Footer1 - Properties**, and set the **Band height** as **20**. We need this because we will display the variable in this band, as the variable's value is final after showing all details of the group data.
10. Drag a **Static Text** element to the **Name Group Footer1** band, set the text as **Total**, and then drag-and-drop the **totalNumberOfSupplier** variable just beside the static text. Now, see the **Name Group Footer1** band; it looks like the following screenshot. Remember to change the font name and size according to the other fields. You can make these fields **Bold**.



11. **Preview** the report, and see the output as shown in the following screenshot:



Name	Mouse		
1	Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
		Total Supplier	**1**
Name	**Printer**		
1	Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
2	ABC Supplies	Dhanmondi, Dhaka	0234567891
		Total Supplier	**2**
Name	**RAM**		
1	Salam Enterprise	2-X/1-10, Mirpur, Dhaka, Bangladesh	0123456789
2	ABC Supplies	Dhanmondi, Dhaka	0234567891
3	XYZ Company	52 Gabtali, Dhaka	0345678912
4	Smart Computers Ltd.	New Elephant Road, Dhaka	0456789123
		Total Supplier	**4**

Summary

In this chapter, we have discussed report groups.

Specifically, we have covered:

- How to add report groups, with and without the report wizard
- How to manage the report groups after creating them
- Adding variables, which is reset when a new group starts

In the next chapter, we will learn another important report type called **Subreport**.

7
Subreports

A subreport is a complete report that is placed in another report. If you've got one item that is linked to several items in another table, such as a department and its employees, then a subreport is what you need. In such cases, generally, the main report (where the subreport is placed) contains the data of the **master/parent** table, and the subreport contains the data of the **detail/child** table. For example, one department has many employees, where `Department` is the master/parent table and `Employee` is the detail/child table. In our database, a particular `Sale` has many `salesLine` entries. If we want to show the details of a particular `Sale`, then in the main report, we will show `SalesNo`, `SalesDate`, and `CustomerNo`; and in the subreport, we will show `ProductCode`, `SalesQuantity`, `UnitSalesPrice`, and so on. Note that subreports can be used for many other purposes also, and not just to print records of a child table. Using the subreport feature of iReport, we can display the output of a report in another report.

The process for creating a subreport is similar to creating a normal report. A subreport has most of the characteristics of a normal report. The difference is that a subreport is placed as an object inside a report.

Typically, a subreport is used to:

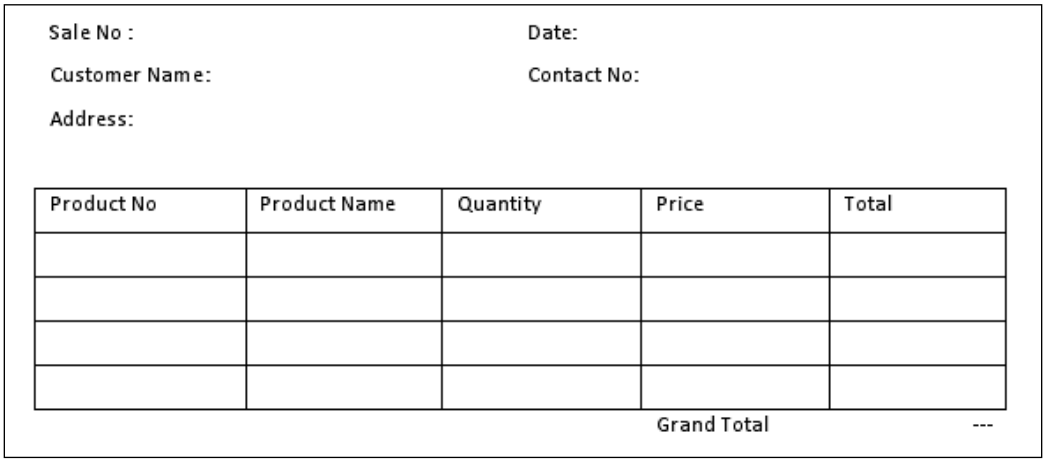
- Combine the master and detail data
- Combine the unrelated reports into a single report
- Link two reports
- Present different aspects of the same data

In this chapter, we will:

- Understand the basics of subreports
- Create subreports
- Compile subreports
- Link the main report to the subreport
- Pass data between the main report and the subreport

Creating a subreport

We will create a report, which shows the details of a particular sale, as shown in the following figure:



There are two parts in the report. The data of the first part consists of entries from the `customer` and `sales` tables, and that of the second part consists of entries from the `product` and `salesLine` tables.

Note that:

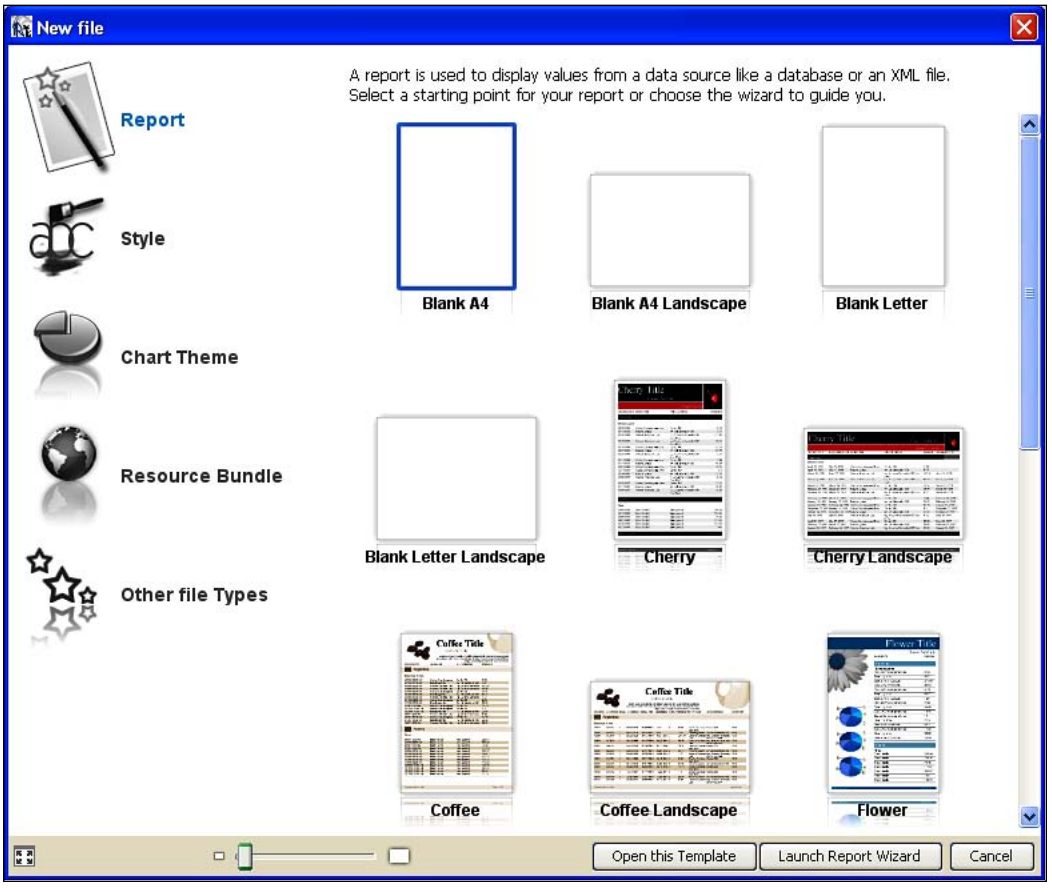
- One sale has only one customer (master)
- One sale has many products (detail)

So the first part of the report will be created as a normal report, and the second part will be created as the subreport.

Creating the master report

Let's create the report. Just follow the listed steps:

1. Go to **File** | **New...**, and select a **Blank A4** template.

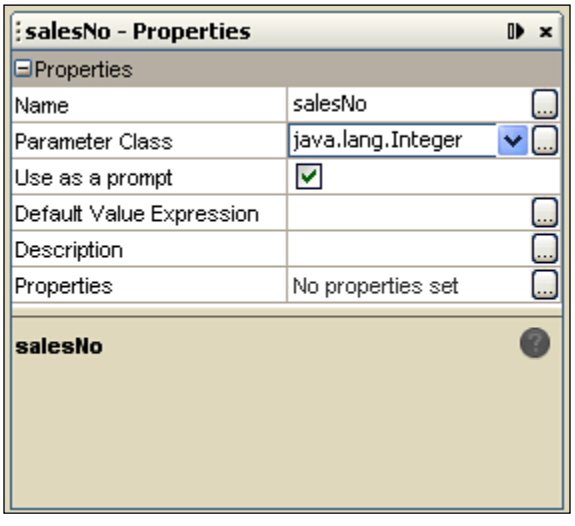


2. Create a report named `SalesSlip` with the following SQL query:

```
SELECT * FROM Sales,Customer
WHERE Sales.customerNo=Customer.customerNo
```

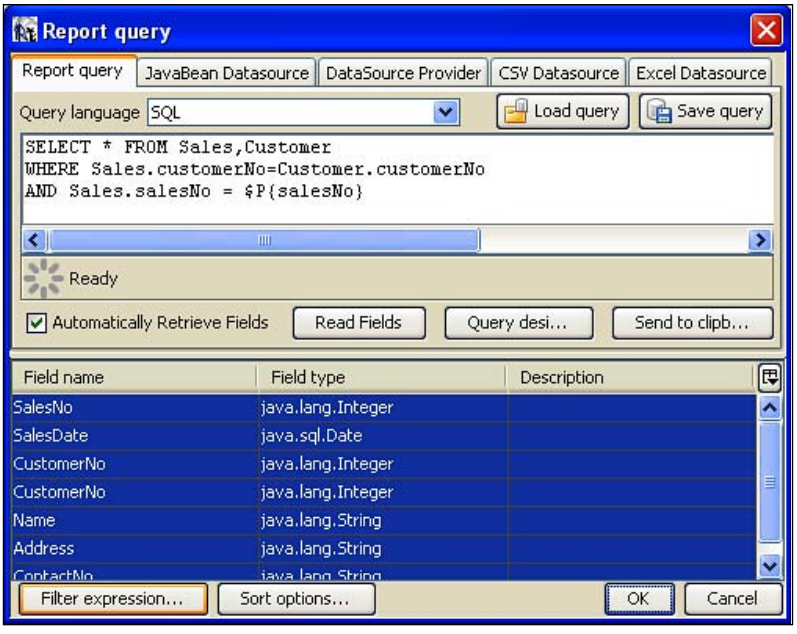
3. Select the **SalesNo**, **Name**, **Address**, **SalesDate**, and **ContactNo** fields, and press **Next >**. After completing all the steps of the Report Wizard, drag-and-drop the report fields from the **Report Inspector** into the **Detail** band. Add labels (**Static Text**) for the fields. Arrange all the elements, and a add a **Static Text** (**Sales Slip**) in the **Page Header** band, so that the report design looks like the following screenshot:

4. If we execute this report, data of all the sales will be shown, as we have not set the filter criteria. To show data from a particular sale, we need to add a parameter. Right-click on **Parameters** in the **Report Inspector** window, and select **Add Parameter**. Enter **salesNo** as the parameter **Name**, choose **java.lang.Integer** as the **Parameter Class** type, and check the **Use as a prompt** checkbox.

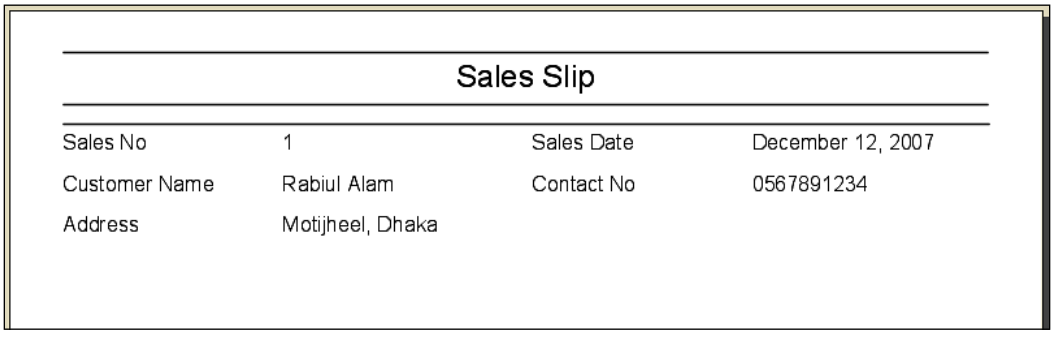


5. Go to the **Report query** editor, and replace the existing query with the following one:

```
SELECT * FROM Sales,Customer
WHERE Sales.customerNo=Customer.customerNo
AND Sales.salesNo = $P{salesNo}
```



6. We have finished creating the master part of the report. You will see the following output if you **Preview** the report with an active connection and input **1** as the **salesNo**.



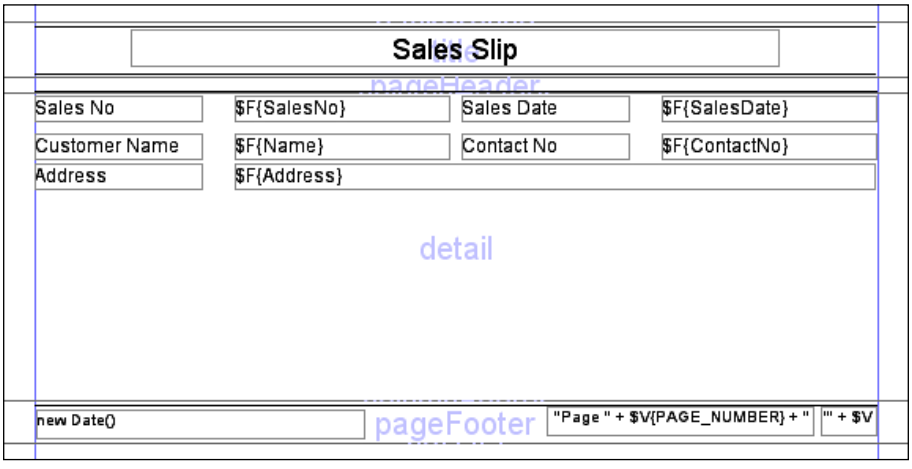
Sales Slip

Sales No	1	Sales Date	December 12, 2007
Customer Name	Rabiul Alam	Contact No	0567891234
Address	Motijheel, Dhaka		

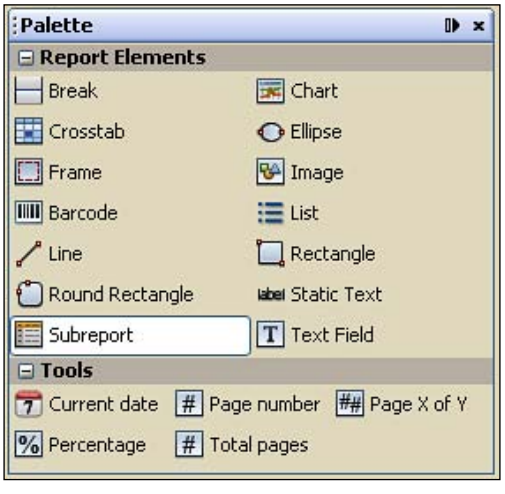
Creating the subreport

Now we will create the subreport in the following steps:

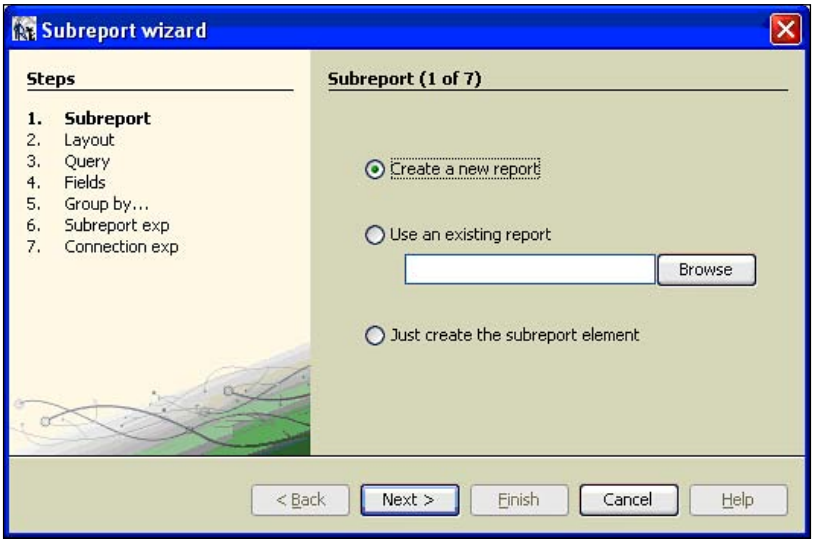
1. First, increase the **Band height** of the **Detail** band, as shown in the following screenshot:



2. Drag-and-drop a **Subreport** element from the **Palette** into the **Detail** band.

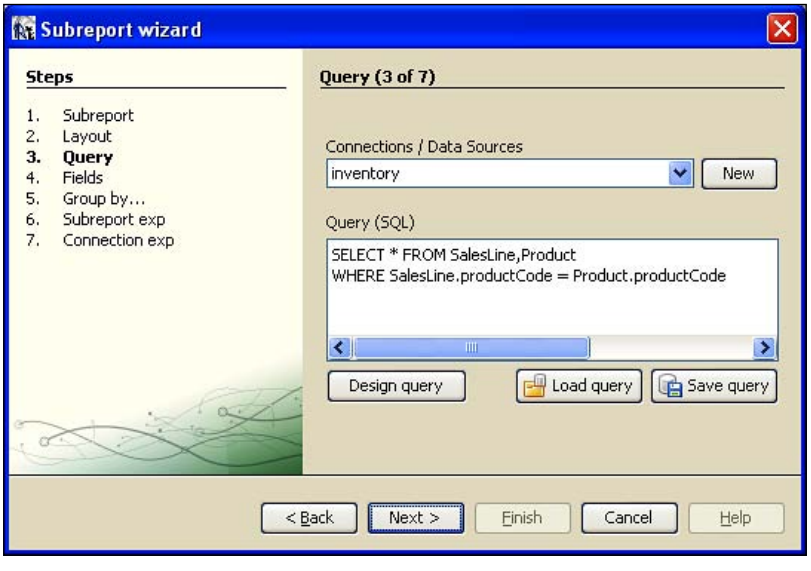


3. You will see a dialog box, as shown in the following screenshot:

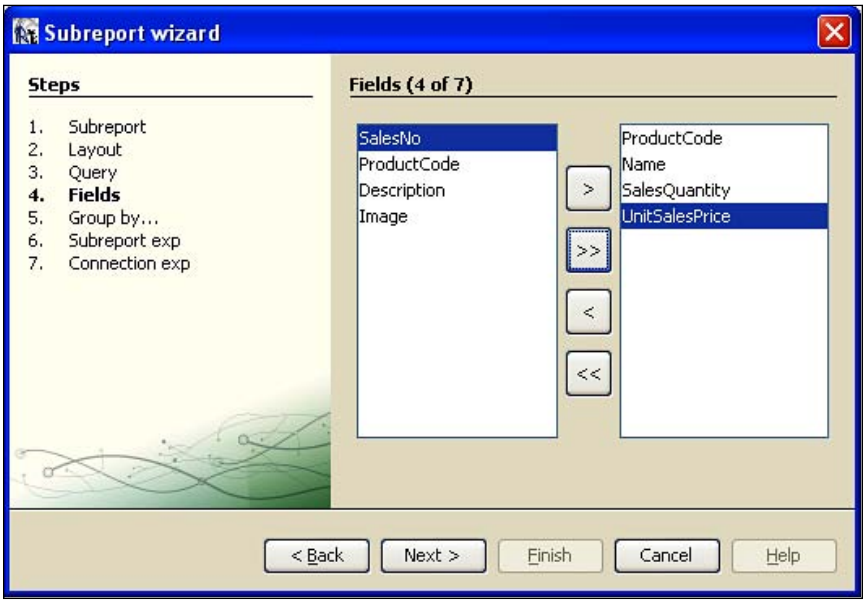


4. Select **Create a new report**, and press **Next >**.
5. From the **Layout** section, select **Blank A4**, and then press **Next >**.
6. Select **inventory** as the connection, and write the following SQL query:

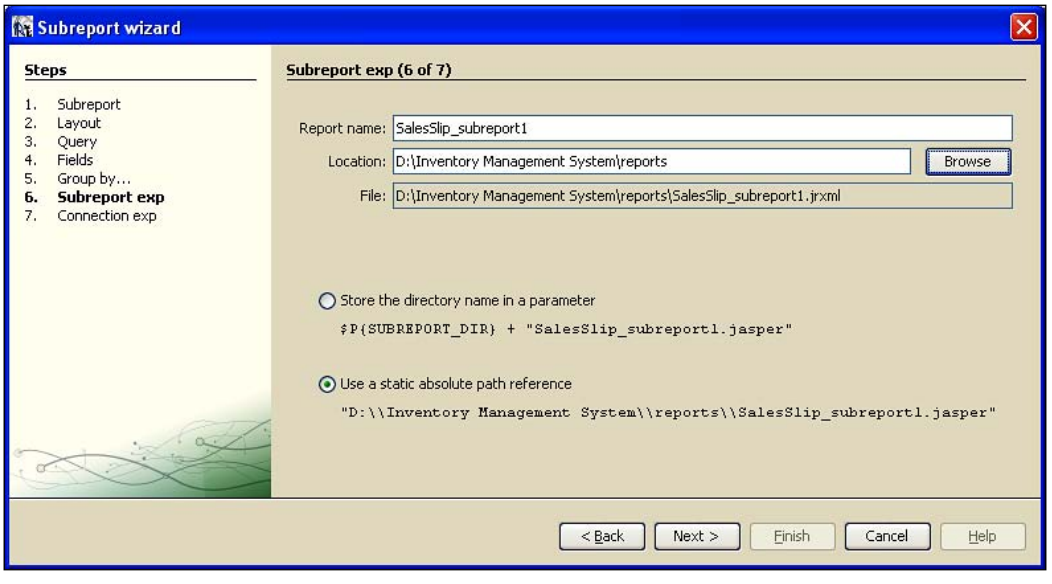
```
SELECT * FROM SalesLine,Product
WHERE SalesLine.productCode = Product.productCode
```



7. Press **Next >**. From the **Fields** section, choose **ProductCode**, **Name**, **SalesQuantity**, and **UnitSalesPrice**, click **>**, and then press **Next >**.



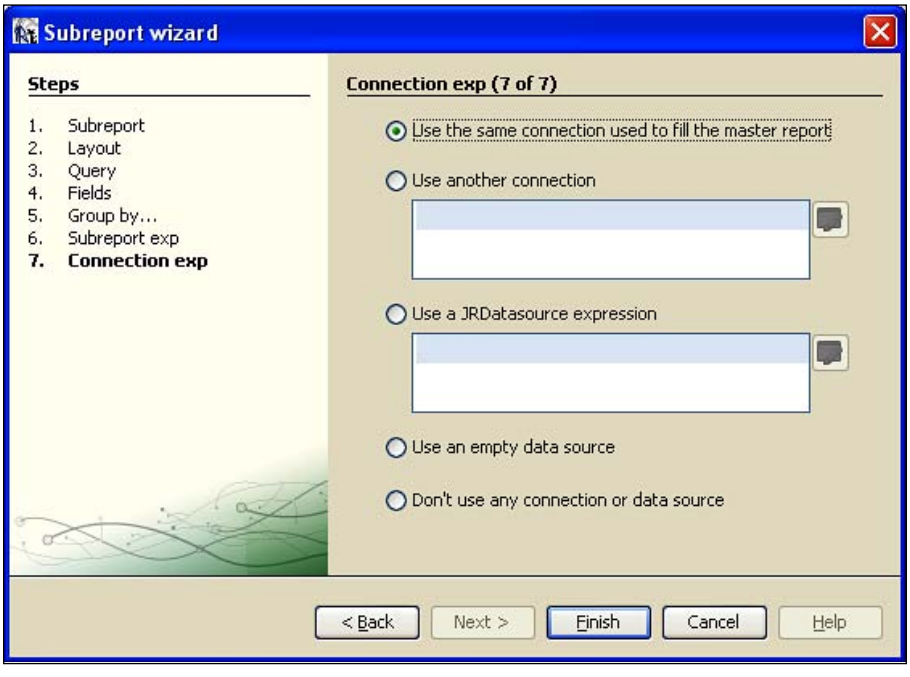
8. Press **Next >** without selecting any group. You will see the following screenshot:



9. Enter the report **Location**, choose the **Use a static absolute path reference** option, and then press **Next >**.

There are two options for the subreport expression: **Store the directory name in a parameter** and **Use a static absolute path reference**.

If the first option is chosen, a parameter named `SUBREPORT_DIR` is created in the location where the subreport is stored. You can modify it from **Report Inspector** | **Parameters**. You should change the **Default Value Expression** of this parameter when you change the report location. Alternatively, you can enter the location as input when the report executes. The **Use a static absolute path reference** option is suitable when the reports are stored in a fixed location.



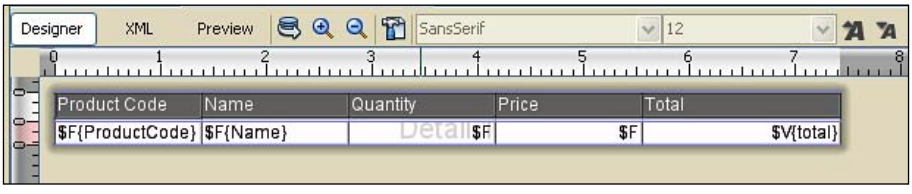
10. In the **Connection exp** section, select the **Use the same connection used to fill the master report** option, and press **Finish**.
11. You will now see the subreport **Designer** view.
12. Create a parameter named `salesNo` for the subreport.

If there is any common parameter in the master report and the subreport, then the subreport parameter should not be taken as input from the user, as the main parameter will be passed to it. JasperReports does not use the **Use as a prompt** information at all. iReport uses it only with master reports, not for subreports. However, it can be useful if it is checked to test the subreport.

13. Replace the report query with the following one:

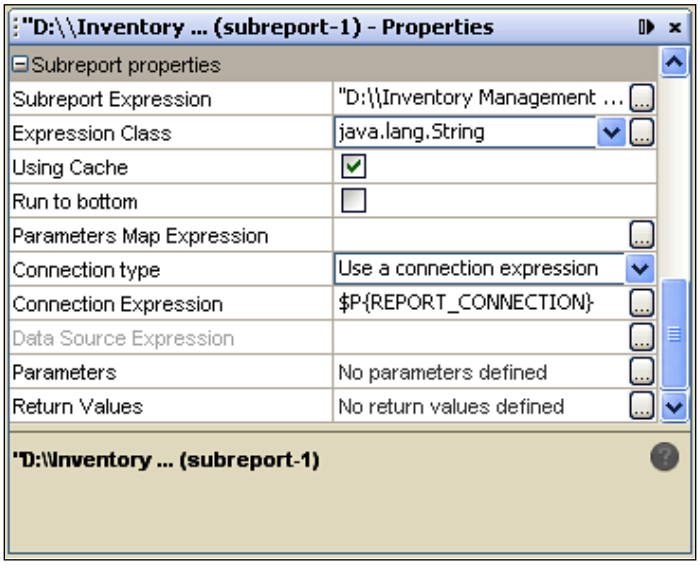
```
SELECT * FROM SalesLine,Product
WHERE SalesLine.productCode = Product.productCode
AND SalesLine.salesNo = $P{salesNo}
```

14. Create two variables: `total` and `grandTotal`, as you have learnt in Chapter 4, *Using Variables*.
15. Arrange all the report elements in the subreport so that the report design looks like the following screenshot:



16. Set borders for the fields, and set right horizontal alignment for **Quantity**, **Price**, and **Total**.
17. Go to the master report by clicking the **SalesSlip.jrxml** tab at the top. Now the report design of the master report looks like the following screenshot:

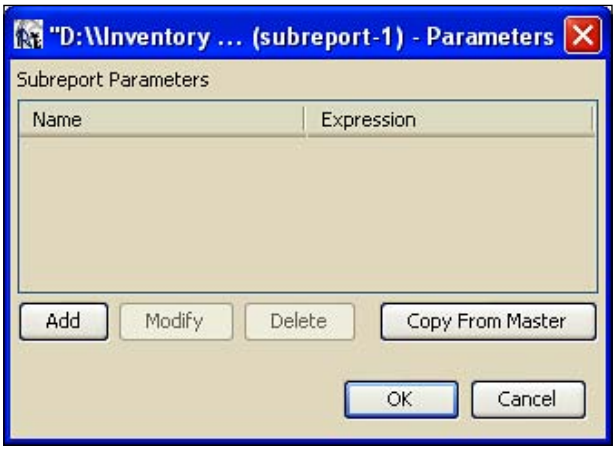
18. Select the subreport, and go to **Properties** | **Subreport properties**.



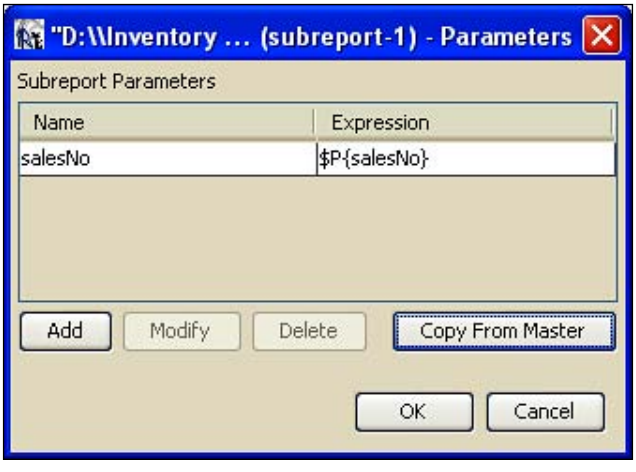
The **Subreport Expression** decides which subreport will be called. You can modify it if required.

Actually, the **Subreport Expression** is automatically generated when you choose the expression option (**Use the same connection used to fill the master report**) while creating the subreport. If the `.jrxml` file (the main report file, which we create) and the `.jasper` file (the compiled file, which is created when we preview the report) exist in the same directory, then we do not need to write the full `.jasper` file location. Instead, the `.jasper` filename is enough. In this case, note that it will work only in iReport, as the report directory is added to the classpath. If you call your report from other applications, then you have to enter the report directory in the classpath.

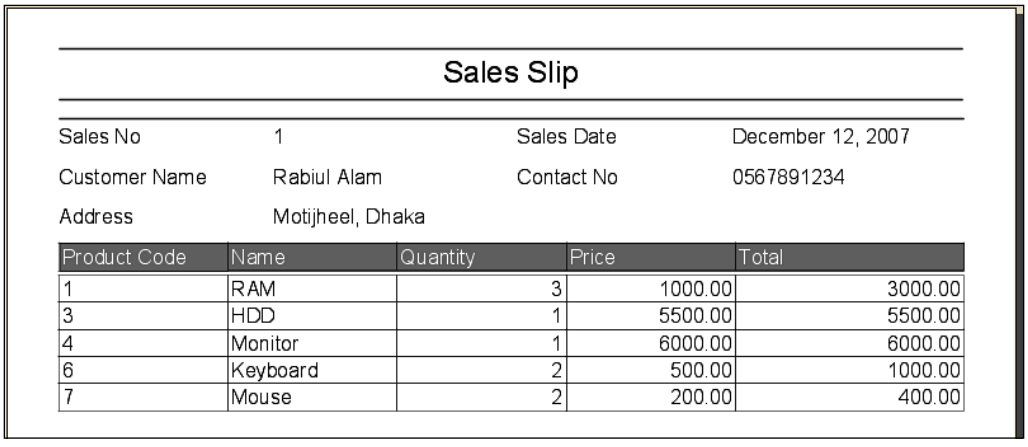
19. Press the button next to **Parameters** to open the **Subreport Parameters** dialog box, as shown in the following screenshot:



20. Press **Copy From Master**, and press **OK**.



21. **Preview** the report with an active connection. Input **1** as the **salesNo**. The report output will be shown in the following screenshot:



Sales Slip

Sales No	1	Sales Date	December 12, 2007
Customer Name	Rabiul Alam	Contact No	0567891234
Address	Motijheel, Dhaka		

Product Code	Name	Quantity	Price	Total
1	RAM	3	1000.00	3000.00
3	HDD	1	5500.00	5500.00
4	Monitor	1	6000.00	6000.00
6	Keyboard	2	500.00	1000.00
7	Mouse	2	200.00	400.00

We have just created a master-detail report. Creating the master section is similar to creating a normal report, but there are some additional steps in the detail section (subreport). Let's review the steps for creating a subreport:

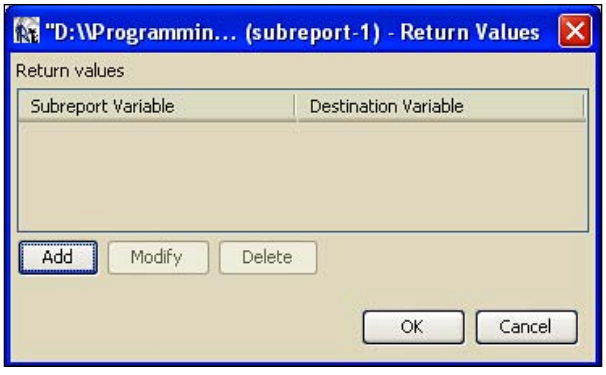
- Drag-and-drop the **Subreport** element into the **Detail** band of the report.
- Develop the subreport element: select subreport, choose layout, write query, select fields, define subreport expression, and select connection expression.
- Create parameters for the subreport.
- Pass values to the subreport parameters from the master report.

Returning values from the subreport

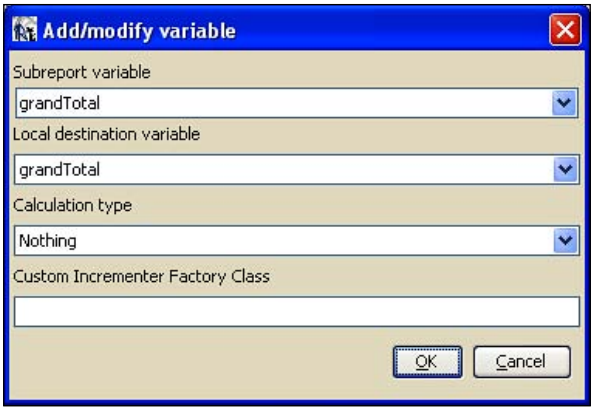
We have created a variable `grandTotal` in the subreport, but we haven't used it yet. Now, we will display the grand total in the master report just below the subreport. Note that the variable is created in the subreport, but the values will be shown in the master report. The concept behind this is that the subreport will return the value of the `grandTotal` variable, and the master report will accept it and copy it to its own local variable. Just follow the listed steps:

1. Create a variable `grandTotal` in the master report. Its data type should be same as the subreport variable, as it will store the returned value. In our case, it is `java.lang.Double`. Note that the variable name will not necessarily be the same. This local variable will be treated as the destination variable, as the subreport variable's value will be returned to it.

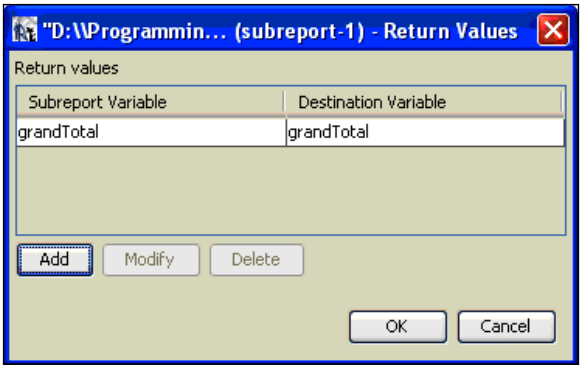
2. Select the **Subreport** element, go to **Properties** | **Subreport properties**, and press the button next to **Return Values**. You will see the following dialog box:



3. Press the **Add** button, and select **grandTotal** as both the **Subreport variable** and the **Local destination variable**.

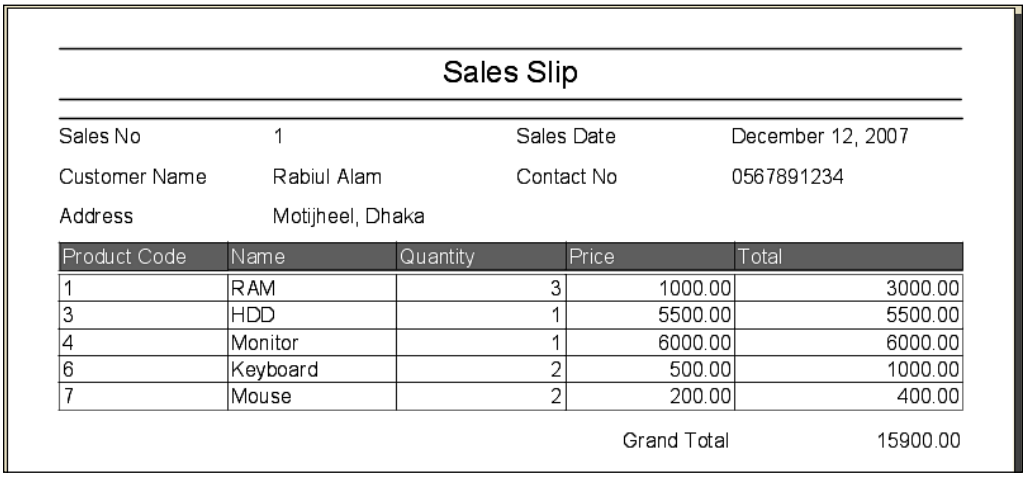


4. Press **OK**.



5. Press **OK**, and place the local `grandTotal` variable on the **Summary** band of the master report. Before that, you need to set the **Summary** band height. Now, the master report design looks like the following screenshot:

6. **Preview** the report with an active connection, and see the report output with the **Grand Total**, as shown in the following screenshot:



Sales Slip

Sales No	1	Sales Date	December 12, 2007
Customer Name	Rabiul Alam	Contact No	0567891234
Address	Motijheel, Dhaka		

Product Code	Name	Quantity	Price	Total
1	RAM	3	1000.00	3000.00
3	HDD	1	5500.00	5500.00
4	Monitor	1	6000.00	6000.00
6	Keyboard	2	500.00	1000.00
7	Mouse	2	200.00	400.00

Grand Total 15900.00

Using an existing report as a subreport

In the previous example, we created a new master report and a new subreport. We can also use an existing independent report as a subreport. To understand this, let's create another report. Suppose we want a list of purchases (just `purchaseNo` and `purchaseDate`) made by a particular supplier. In the master part of the report, the supplier details will be displayed. In the detail part, the list of purchases will be displayed. As we are using an independent report as a subreport, we will produce the list of purchases first, and then this report will be used as a subreport.

Create a report that shows the list of purchases. The following are some report-specific steps:

- SQL query for the report:

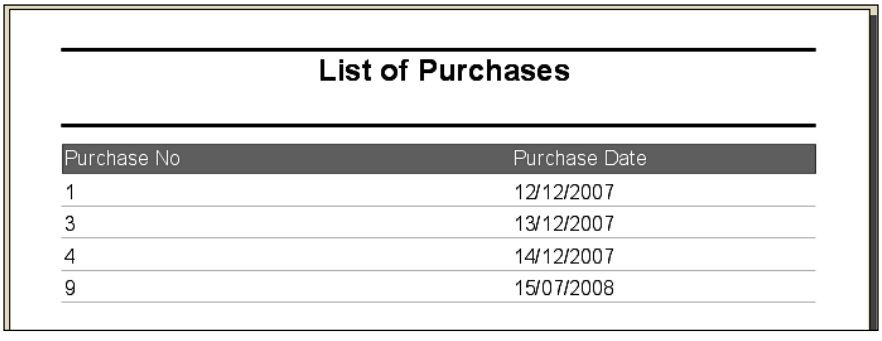
  ```
  SELECT * FROM Purchase
  ```

- Create a parameter `supplierNo`, and modify the query as follows:

  ```
  SELECT * FROM Purchase
  WHERE SupplierNo = $P{supplierNo}
  ```

- Save the report as `ListOfPurchase.jrxml`

If you execute the report, you will see the following output for **supplierNo 1**.



List of Purchases

Purchase No	Purchase Date
1	12/12/2007
3	13/12/2007
4	14/12/2007
9	15/07/2008

Now, create another report that shows the details of a particular supplier. Create a parameter `supplierNo`, and save the report as `ParticularSupplier.jrxml`. The SQL query for the report is as follows:

```
SELECT * FROM Supplier
WHERE SupplierNo = $P{supplierNo}
```

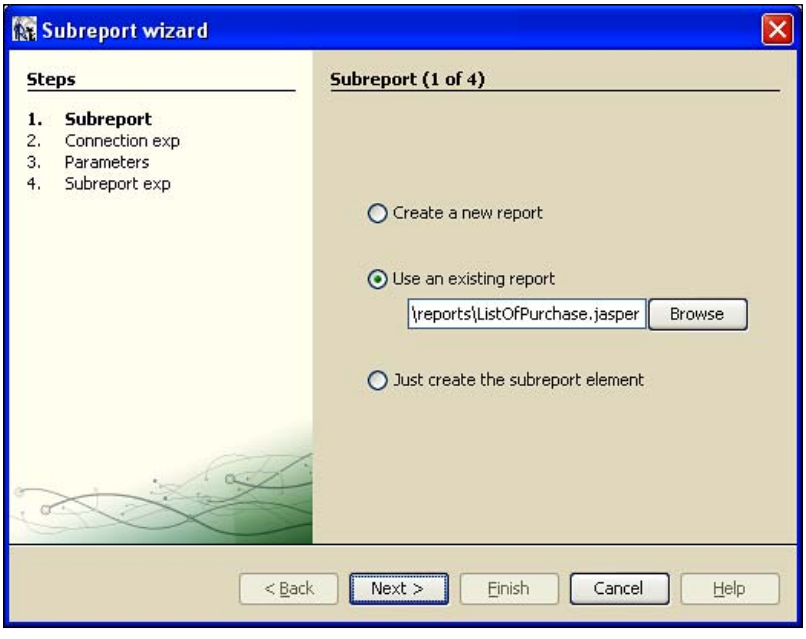
The report output will be as shown in the following screenshot:

Now we have two reports:

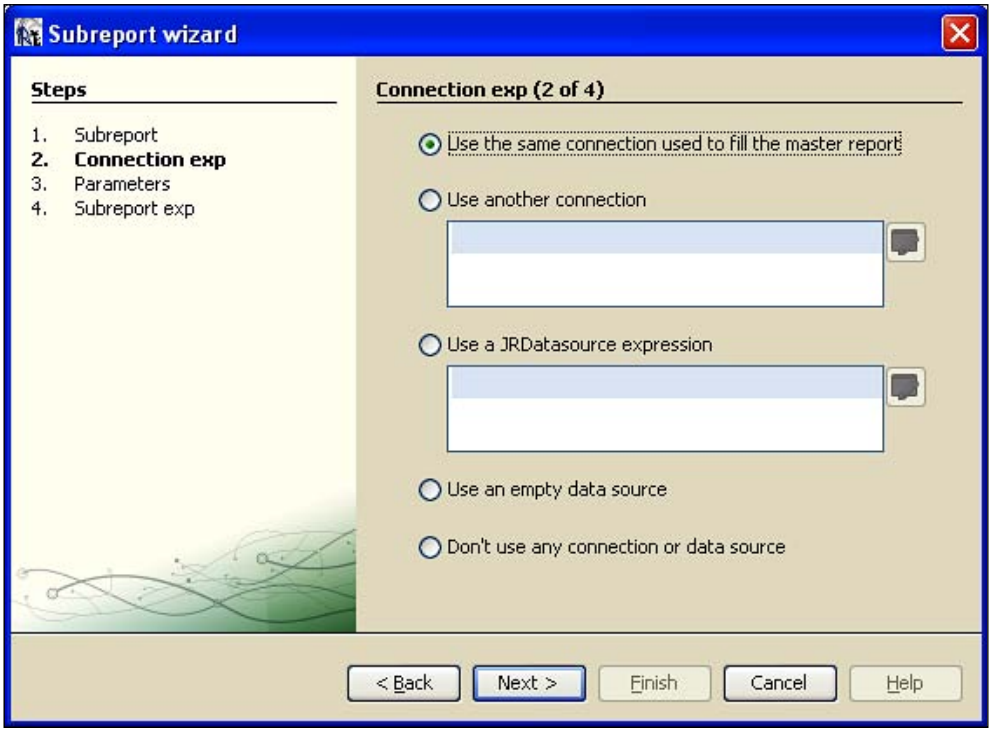
- List of Purchases
- Particular Supplier Details

We want to use **List of Purchases** as the subreport of **Particular Supplier Details**. Just follow the listed steps:

1. Open the `ParticularSupplier.jrxml` file.
2. Drag a **Subreport** element on the **Detail** band. You will see the following dialog box:

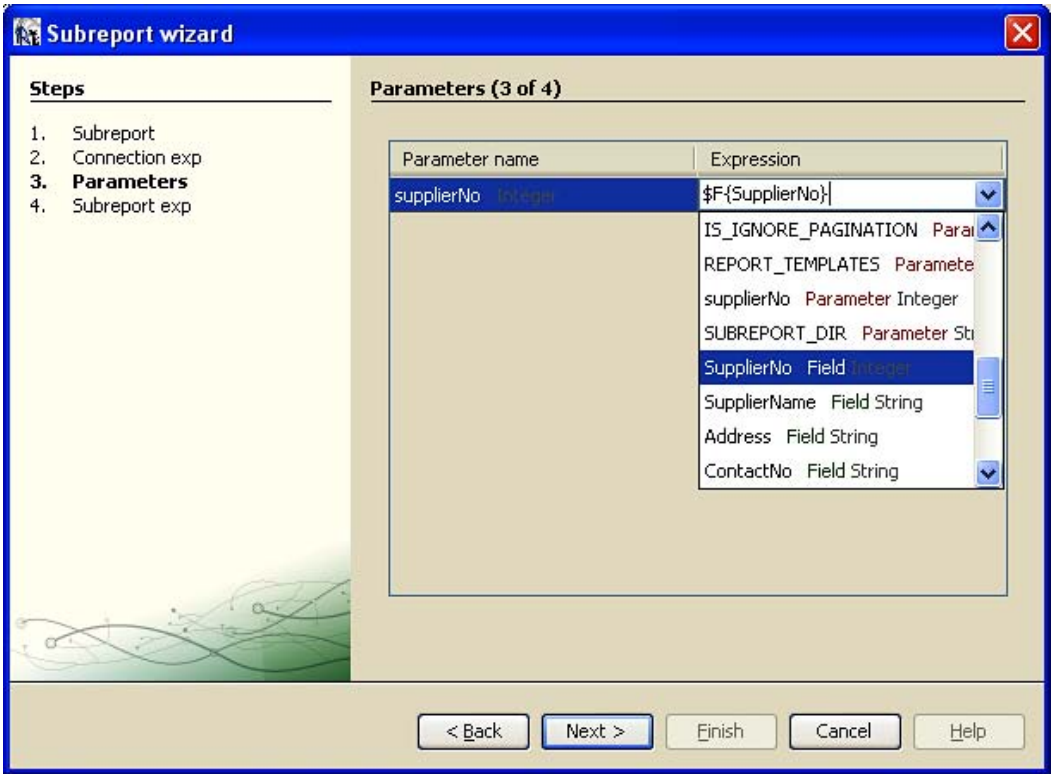


3. Select the **Use an existing report** option.
4. Browse to `ListOfPurchase.jasper`.
5. Press **Next >**.
6. You will see the following dialog box. Select the **Use the same connection used to fill the master report** option, and then press **Next >**.



7. Select **SupplierNo** from the **Expression** drop-down list, and press **Next >**.

Here, the **supplierNo** in the left column is the subreport parameter name, and the **SupplierNo** on the right is the master report field name.



8. Select the **Use a static absolute path reference** as the **Subreport exp** option.
9. Press **Finish**.

10. **Preview** the report with an active connection, and see the output as shown in the following screenshot:

Supplier Details

Supplier No	1
Name	Salam Enterprise
Address	2-H/1-10, Mirpur, Dhaka, Bangladesh
ContactNo	0123456789

List of Purchases

Purchase No	Purchase Date
1	12/12/2007
3	13/12/2007
4	14/12/2007
9	15/07/2008

If the subreport page width does not fit within the master report, then change the page setup of the subreport accordingly.

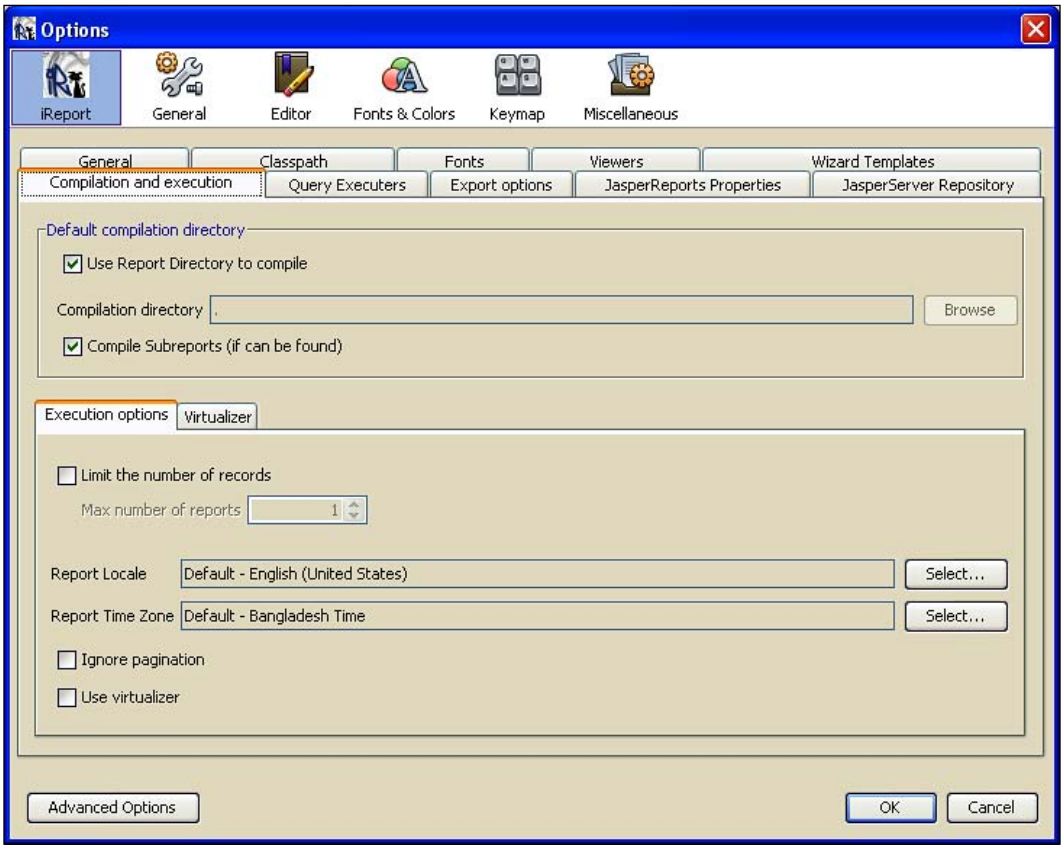
Compiling a report

When we develop a report, a `.jrxml` file is created, but when a report is executed, the compiled version of that `.jrxml` file is called.

When the master report is executed, the compiled version of the subreport, as well as the master report, is called. That's why you have to ensure that the jasper files exist.

JRXML is the default file extension of reports, and JASPER is the extension of the compiled file.

For modifying the compiler settings, go to **Tools | Options | Compilation and execution**.



For the ease of development, ensure that both the **Use Report Directory to compile** and **Compile Subreports (if can be found)** options are checked.

Summary

In this chapter, we have learnt how to use a report inside another report. If we want to see the output of at least two queries in a single report, or if we want data from different connections in a single place, then using subreport is the solution.

Specifically, we have covered:

- Understanding master-detail reports
- Creating subreports
- Passing values to subreports
- Understanding subreport expressions
- Returning values from subreports
- Using both new and existing reports as subreports

8
Crosstab Reports

Business data in the form of a list is not very suitable for comparison or trend analysis. Crosstab elements in a report provide the support to present data in a suitable format, through which business decisions can be made easily. Crosstab elements provide an easy way to create summaries of data in a tabular format. Like the list reports, crosstab elements show data in columns and rows, but the values at the intersection of rows and columns show summarized information rather than detailed information. The user can very easily analyze this data visually to compare values in one group against the values in another group.

After going through this chapter, you will have a good understanding of how to use crosstab reports. In this chapter, we will design a crosstab report by learning how to:

- Define the data set
- Define row and column groups
- Show group total
- Format crosstab elements

Understanding a crosstab report

The reports we have created so far were in the form of lists, that means we haven't created any summary reports. Before going to create a crosstab report, let's look at a typical crosstab report:

	Jan- 09	Feb- 09	Mar- 09	**Total**
RAM	89	75	90	**254**
Monitor	45	45	40	**130**
Printer	60	41	36	**137**
Total	**194**	**161**	**166**	**521**

This is a monthly sales report grouped by product. The outermost group field (the leftmost column) is the product name, represented on each row of the grid. The next three columns (**Jan-09**, **Feb-09**, and **Mar-09**) represent the innermost group field, sales date grouped by month. These columns span horizontally along the page based on the report data.

Creating a crosstab report

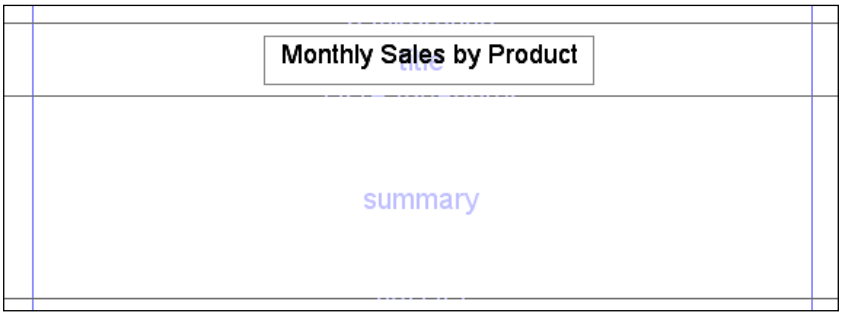
Let's create a crosstab report in some easy steps, as follows:

1. Create a **Blank A4** report. Enter **Monthly Sales by Product** as the **Report name**, and write the following SQL command as the report query:

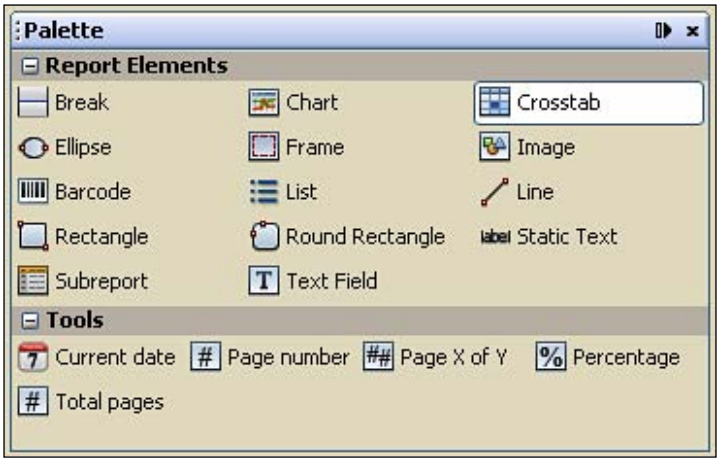
```
SELECT * FROM Sales, SalesLine, Product
WHERE Sales.salesNo = SalesLine.salesNo
AND SalesLine.productCode = Product.productCode
```

2. Select **SalesDate**, **Name**, and **SalesQuantity** fields, and create the report without selecting any groups.
3. Set the **Band height** of all bands to zero, except the **title** and the **summary** bands. Increase the **summary** band height. Now, the report layout will be as shown in the following screenshot:

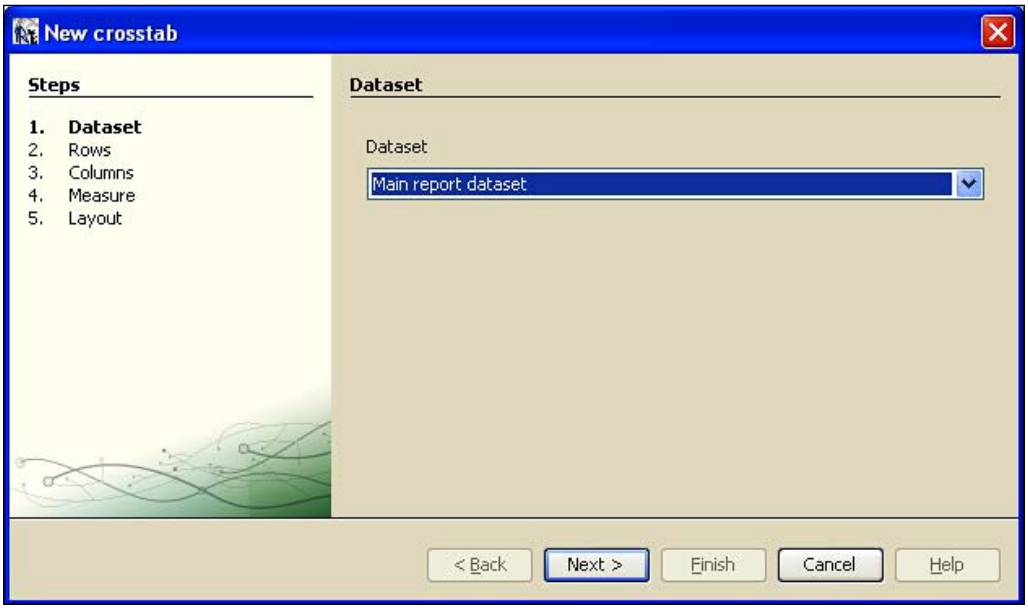
4. Place a **Static Text** element on the **title** band, and set the text as **Monthly Sales by Product**.



5. Drag a **Crosstab** element onto the **summary** band from the **Palette**.

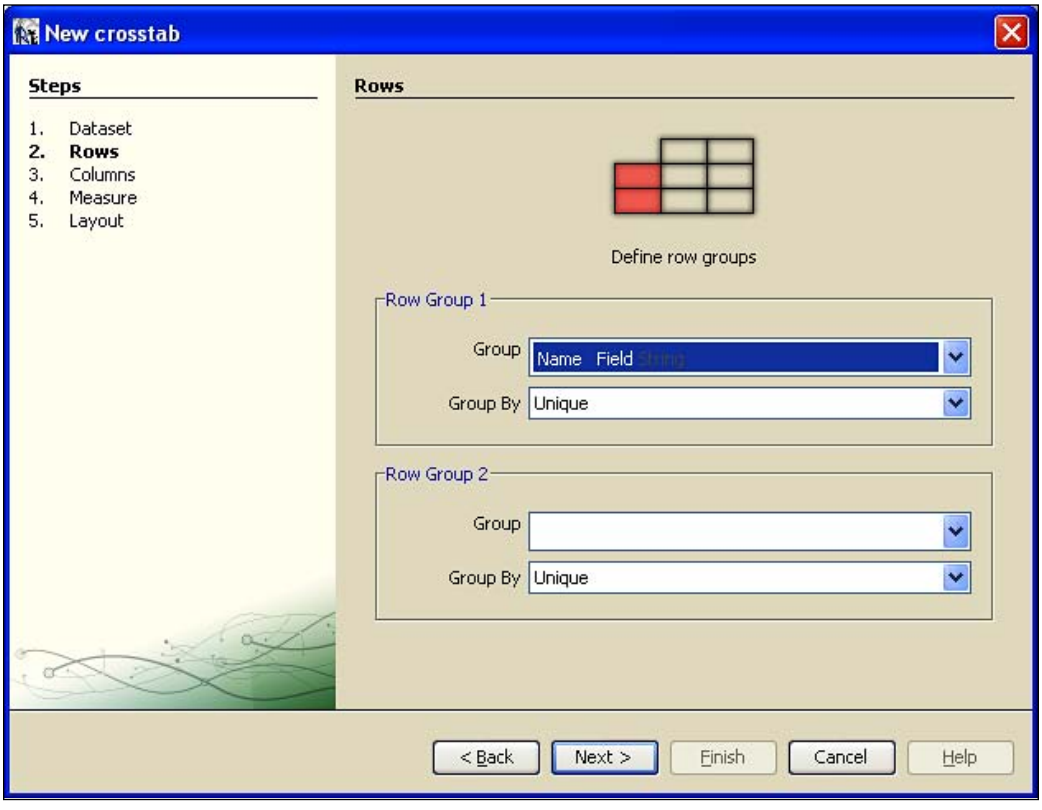


6. You will see a dialog box for **Dataset** selection. Choose **Main report dataset**, and press **Next >**.



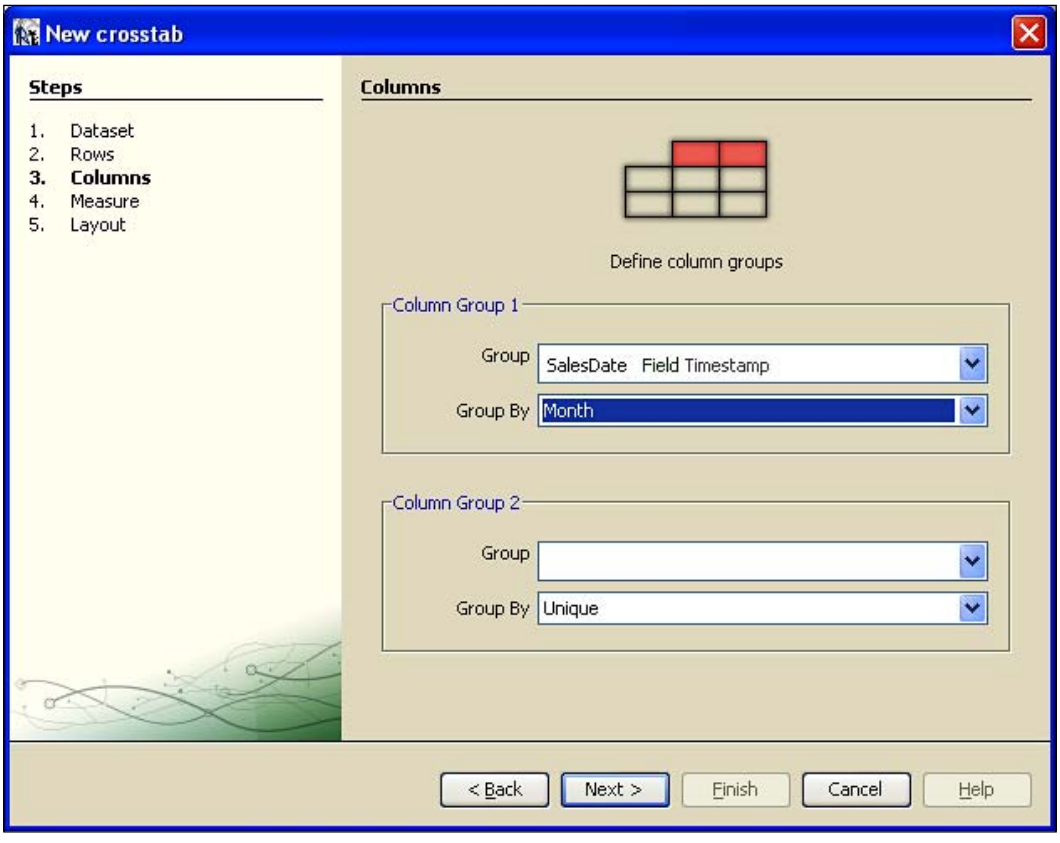
Grouped or summary data can be produced from the detailed data supplied in a report. The data, which is gathered from the database or any other data source, for creating the crosstab is called the **dataset**. The dataset which was produced from the main report query is called the **main report dataset**. Besides this main dataset, it is possible to create a sub dataset to be used only with the crosstab or charts. In that case, you have to select the appropriate dataset.

7. In step two of defining the crosstab element, you have to define the rows for grouping. Select **Name Field** as the **Group** option of **Row Group 1** and press **Next >**.



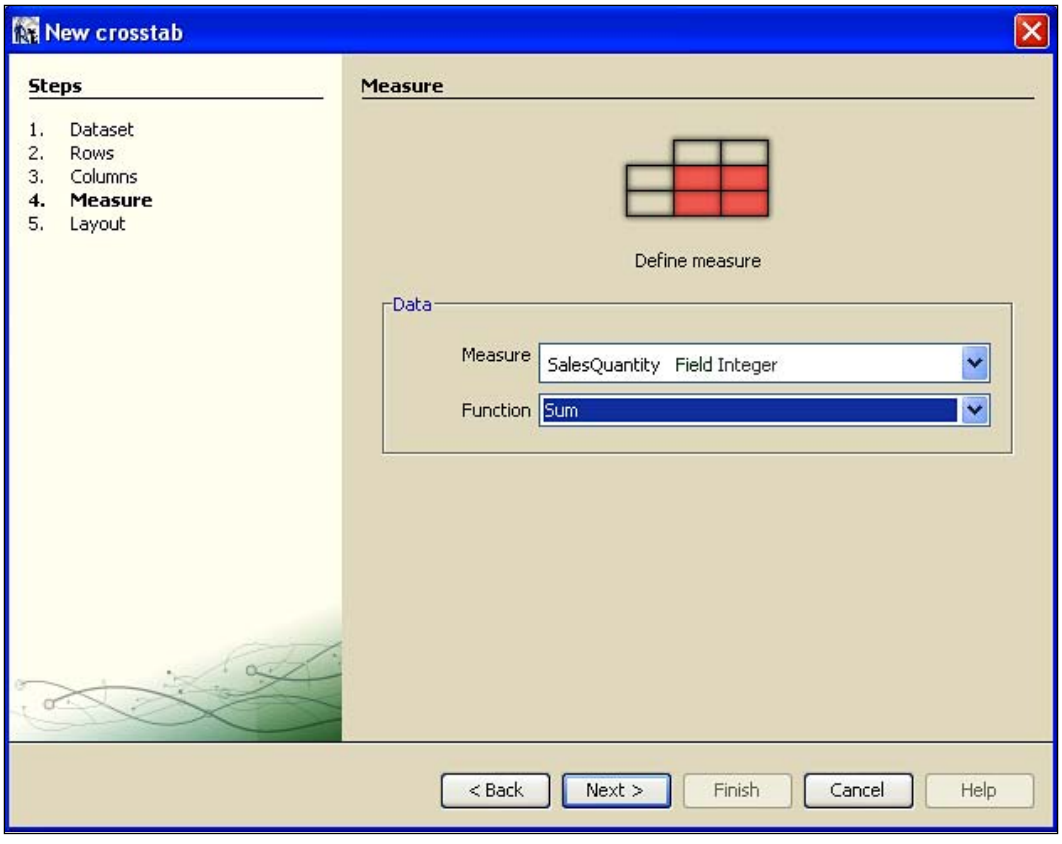
In this step, the outermost group field (the first column, which is colored in the previous screenshot) of the crosstab is selected. We will display the product name in the first column of each row, which is the reason for selecting the **Name Field** from the list of available fields.

8. In step three, from the **Column Group 1** options, select **SalesDate Field** as the **Group** option and **Month** as the **Group By** option, and then press **Next >**.

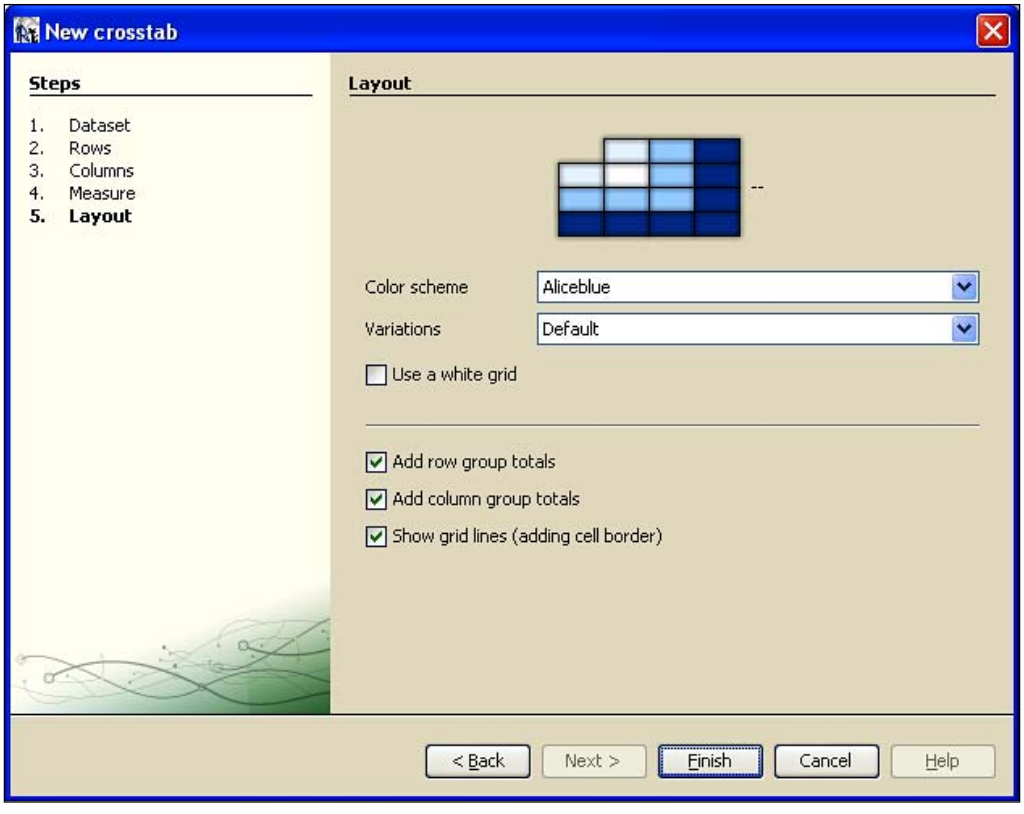


In this step, the innermost group fields are selected (the columns colored in the figure). The number of columns is not fixed, and it will span dynamically based on the data. In our case, if the data is available for three months, then the number of columns will be three. If data for five months is available, then the number of columns will be five. As our objective is to show monthly sales, we have chosen **SalesDate** as the **Group** field and grouped it by month.

9. In step four, select **SalesQuantity Field** from the **Measure** options and **Sum** from the **Function** options. Press **Next >**.



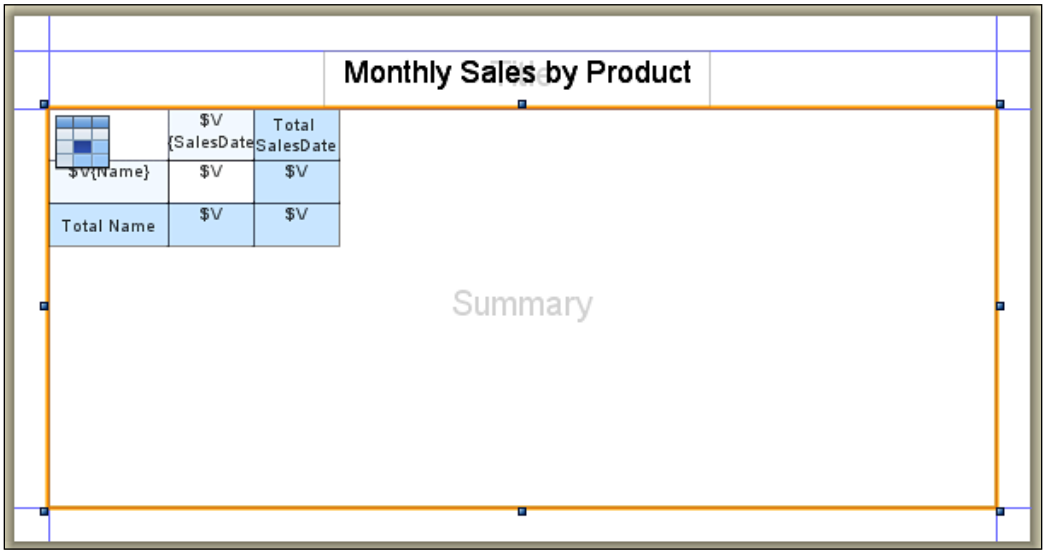
10. In the last step, select the desired **Color scheme** and **Variations**. Check the **Add row group totals**, **Add column group totals**, and **Show grid lines** checkboxes, and then press **Finish**.



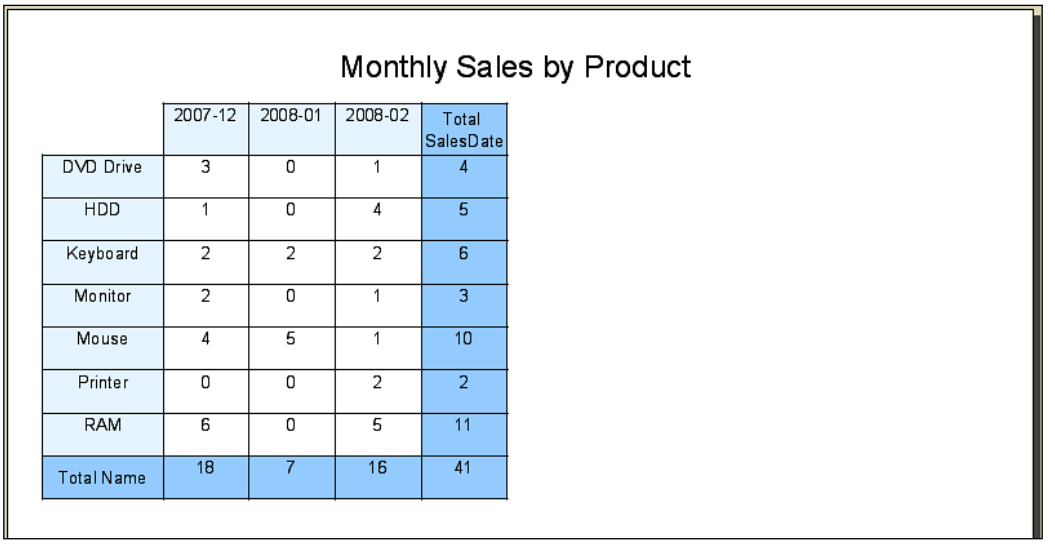
In the preceding steps, we have defined just the columns. Now, the **Measure** field is selected. We have shown the total quantity of sales by selecting the **SalesQuantity** field and the **Function** as **Sum**. The **Function** options available are **Count**, **Average**, **Lowest**, **Highest**, and so on. We have selected **Sum** as we need the total sales of a particular product.

For showing the group total in the last row and the last column, we need to check two checkboxes: **Add row group totals** and **Add column group totals**. In our report, you will see the total sales of a particular month and total sales of a particular product because these options were checked.

11. Now the report layout looks like the following screenshot:



12. **Preview** the report, and see the output as shown next:



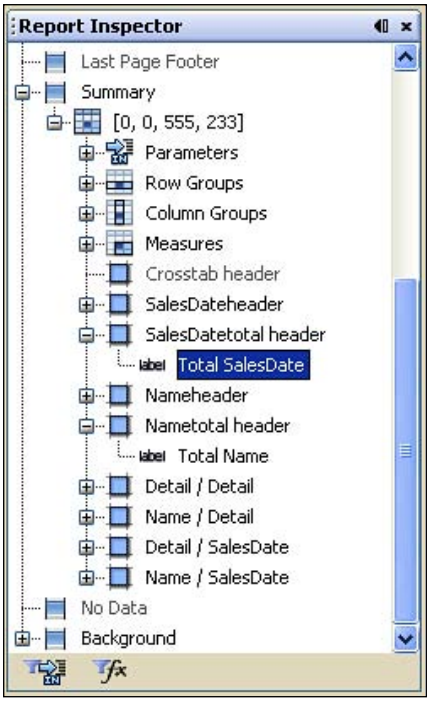
Monthly Sales by Product

	2007-12	2008-01	2008-02	Total SalesDate
DVD Drive	3	0	1	4
HDD	1	0	4	5
Keyboard	2	2	2	6
Monitor	2	0	1	3
Mouse	4	5	1	10
Printer	0	0	2	2
RAM	6	0	5	11
Total Name	18	7	16	41

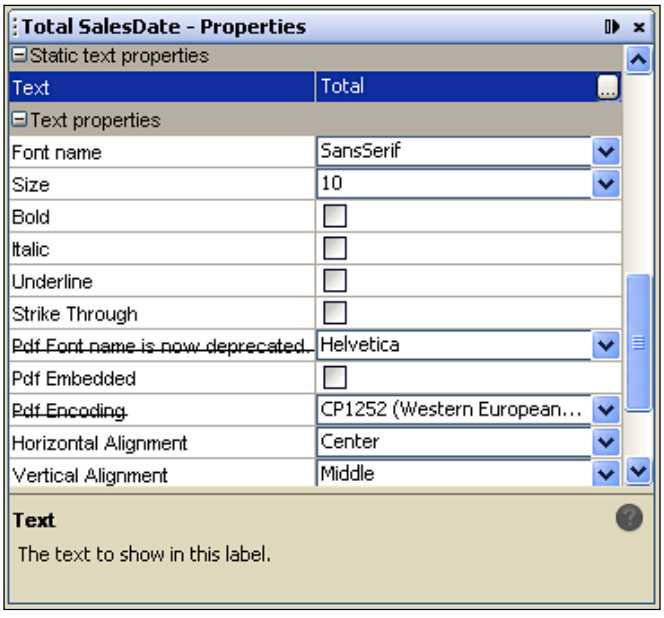
Formatting crosstab elements

Now, we will change some default settings of the crosstab elements. We can modify the text field expression, change alignment, format date, and so on.

1. We want to change the total headers from **Total SalesDate** and **Total Name** to only **Total**. To change the text, expand the crosstab under the **Summary** band in the **Report Inspector**, and select **Total SalesDate** by expanding the **SalesDatetotal header**.



2. Go to **Total SalesDate - Properties**, and scroll to **Static text properties**. Type **Total** in the **Text** field, as shown in the next screenshot:



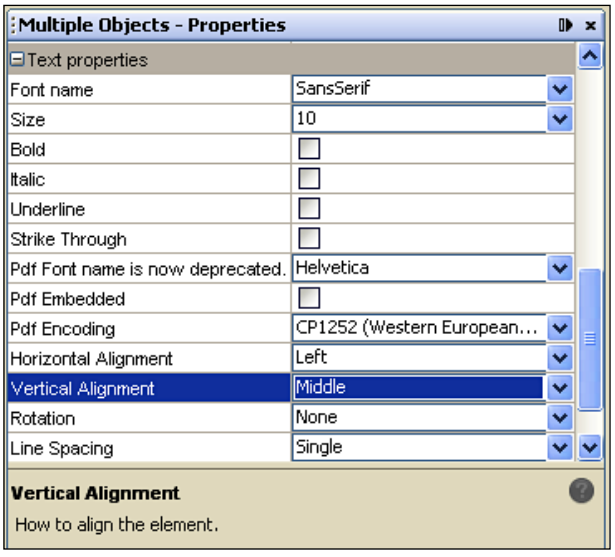
3. In the same way, change the header **Total Name** to **Total**.

Selecting elements in the crosstab

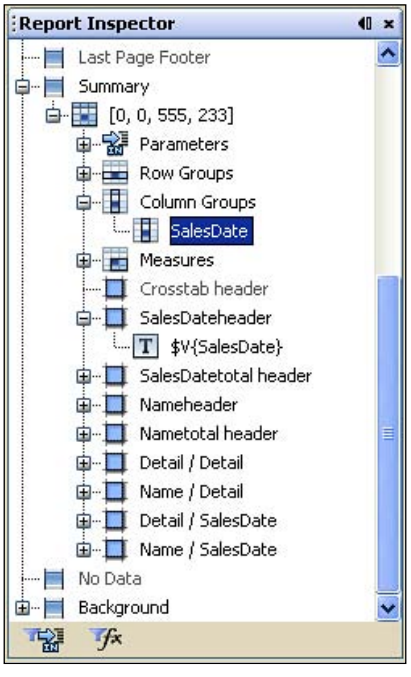
The crosstab tree in the **Report Inspector** is not the only option to select an element. You can select the desired element from the crosstab **Designer** also.

4. From the designer, select the **$V{Name}**(**Nameheader**) and **Total**(**Nametotal header**).

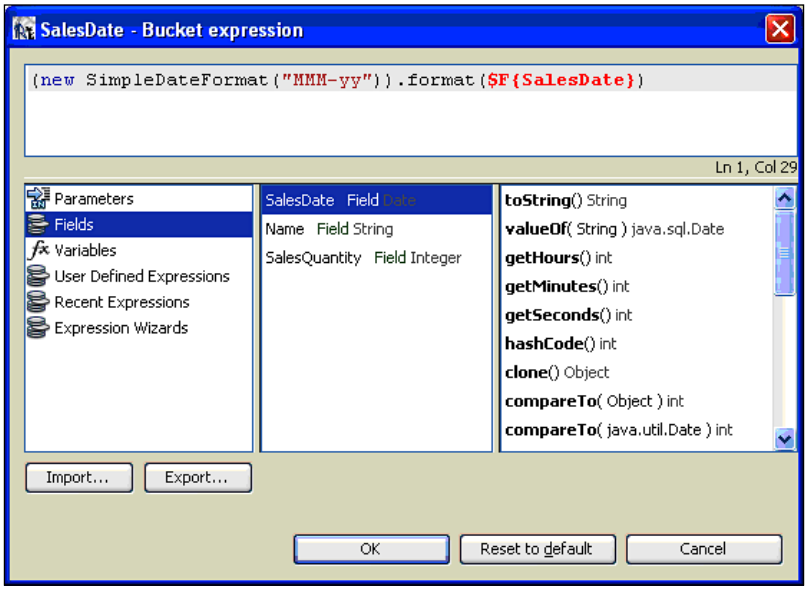
5. Now, change their **Horizontal Alignment** to **Left** and **Vertical Alignment** to **Middle** from the **Properties | Text properties** window.



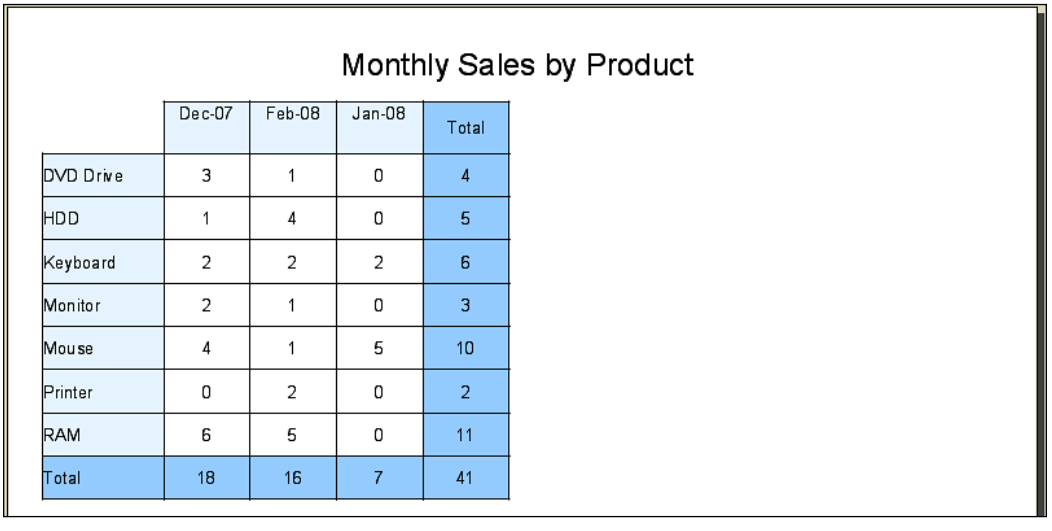
6. For changing the date format, go to **Report Inspector | Summary**, expand the crosstab, and select **SalesDate** from **Column Groups**.



7. Go to the **Properties** window, modify the **Bucket Expression** to `(new SimpleDateFormat("MMM-yy")).format($F{SalesDate})`, and then press **OK**.



8. Now, **Preview** the report with an active connection, and see the output, as shown in the following screenshot:



Monthly Sales by Product

	Dec-07	Feb-08	Jan-08	Total
DVD Drive	3	1	0	4
HDD	1	4	0	5
Keyboard	2	2	2	6
Monitor	2	1	0	3
Mouse	4	1	5	10
Printer	0	2	0	2
RAM	6	5	0	11
Total	18	16	7	41

Summary

We learned a lot in this chapter about crosstab reporting.

Specifically, we have covered:

- Understanding crosstab reports
- Creating crosstab reports
- Formatting crosstab elements

Now that we've learned about crosstab reports, we're ready to develop reports with charts, and that is the topic of the next chapter.

9
Charting

Charting is a very important feature for modern day reporting. From a chart, one can get an easy-to-understand, visual view of business data. Modern decision makers largely depend on reports with charts for crucial business decisions. You can create various types of charts in iReport, such as a **Pie**, **Pie 3D**, **Bar**, **Bar 3D**, **YX Bar**, **Stacked Bar**, **Stacked Bar 3D**, **Line**, **XY Line**, **Area**, **YX Area**, **Stacked Area**, **Scatter**, **Bubble**, **Time Series**, **High Low**, **Candlestick**, **Gantt**, **Meter**, **Thermometer**, and **Multi Axis**.

In this chapter, we will:

- Develop a report with a pie chart
- Develop a report with a 3D pie chart
- Develop a report with a bar chart

So let's get on with it.

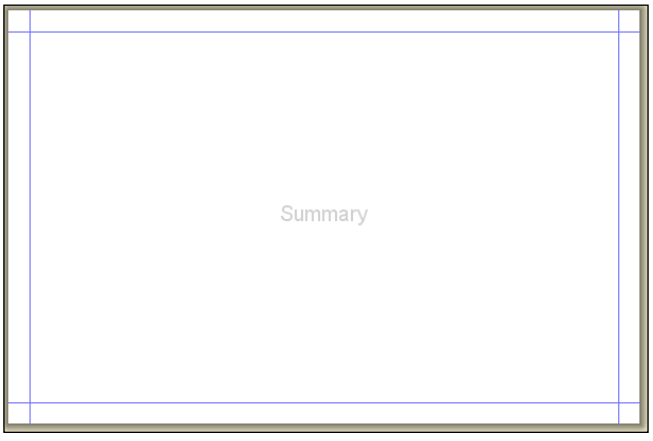
Developing a pie chart report

A pie chart is a circular chart, which is divided into several pie shaped sections. The circle represents the whole and each section represents a portion or percentage of the whole. For example, a company may have several revenue options, and after a financial year, the decision makers need to know the percentage of each revenue option. In this case, the pie chart will be an important tool for representing the data.

In our case, total sales volume in terms of the sales quantities is the whole and sales quantities of a particular product is a section. In the following example, **RAM**, **Mouse**, **Printer**, and so on are different sections that represent the percentage of the sales quantity of **RAM**, **Mouse**, **Printer**, and so on.

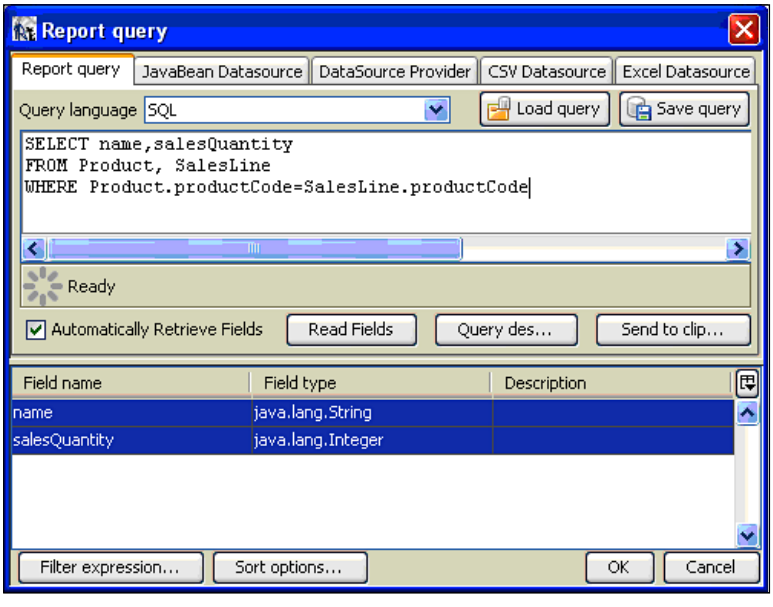
We want a report with a chart of the all-time sales volume percentage. Just follow the listed steps:

1. Create a blank report having just the **Summary** band visible, with a **Band height** that is big enough to place the chart element in.

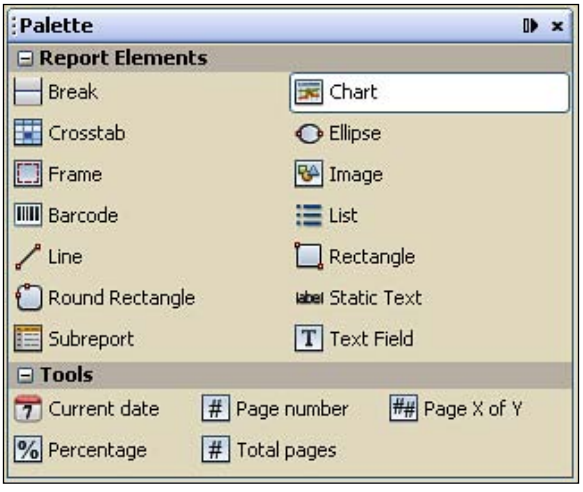


2. Open the **Report query** editor, and write the following query:

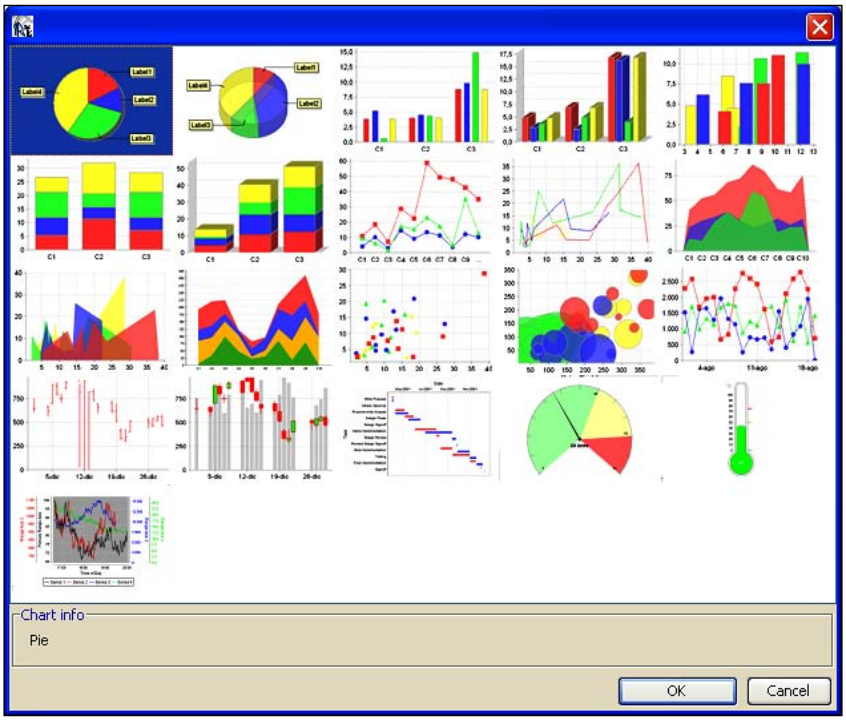
```
SELECT name,salesQuantity
FROM Product, SalesLine
WHERE Product.productCode=SalesLine.productCode
```



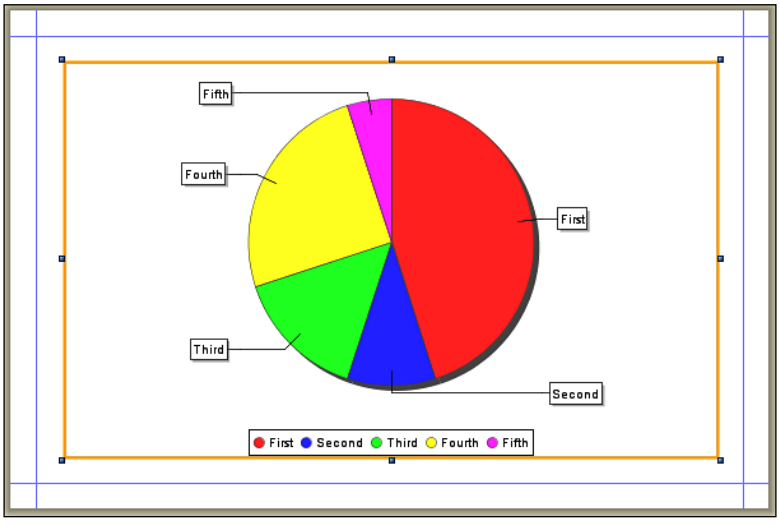
3. Press **OK**.
4. From the **Palette | Report Elements**, drag-and-drop the **Chart** tool on the **Summary** band.



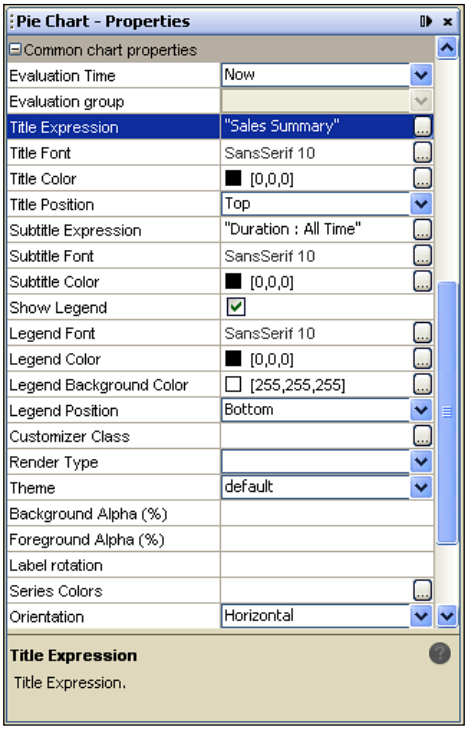
5. From the list of chart options, select **Pie** chart.



6. Now the report design with the chart looks like the following screenshot:



7. Select the chart, and set some of the properties from the **Pie Chart - Properties** pane. Scroll down to the **Common chart properties**.



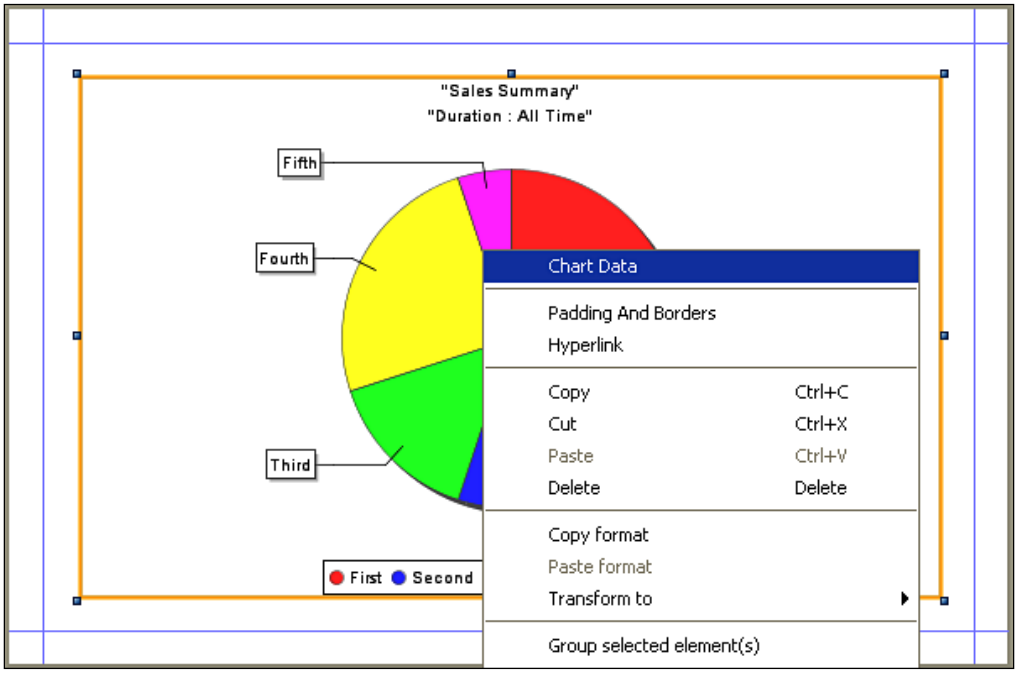
8. Enter **"Sales Summary"** as the **Title Expression** and **"Duration : All Time"** as the chart **Subtitle Expression**, as shown in the previous screenshot.

Before going to the next step, let's understand some common chart properties:

- **Title Expression**: Is a string expression, which is shown at the top, bottom, left, or right of the chart as a title. We have set **Title Expression** as **Sales Summary**. You can change the font and color of the title using the **Title Font** and **Title Color** options.
- **Title Position**: There are four positions, which are: **Top**, **Bottom**, **Left**, and **Right**. We have chosen **Top** as the **Title Position**.
- **Subtitle Expression**: Is a string expression, which is shown below the **Title Expression**. We have shown the time period of the report data as a subtitle.
- **Legend**: Legends are just captions of the different sections of the chart, which are shown with the chart as an indication of the values. To hide the legend, uncheck the **Show Legend** checkbox.

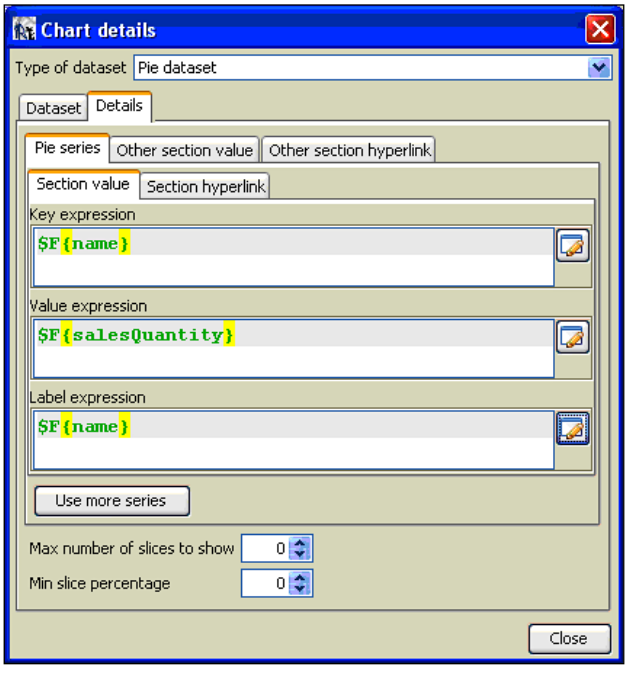
Let's get back to creating the pie chart report:

9. Right-click on the chart, and select **Chart Data**.



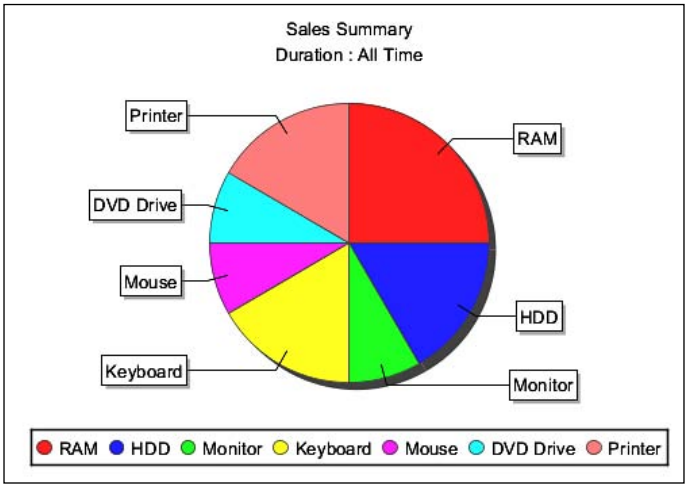
10. Go to the **Details** tab, and set **$F{name}**, **$F{salesQuantity}**, and **$F{name}** as the **Key expression**, **Value expression**, and **Label expression** respectively. Note that the name field should not be null in this case.

Alternatively, you can use the expression editor to select **name**, **salesQuantity**, and **name** fields for the expressions.



The chart is drawn based on the data extracted from the database after the SQL query is executed. We defined the SQL query first, and then drew the chart. In the **Details** section of **Chart Data**, we defined three things: **Key Expression**, **Value Expression**, and **Label Expression**. Generally, the **Key Expression** is the unique key field for which the percentage is calculated, and the size of the portion/section depends on the value of that key. If your label is other than the key field, then you can choose the relevant field that can describe the key or value. If you don't specify the **Value expression**, then the **Key expression** is used by default.

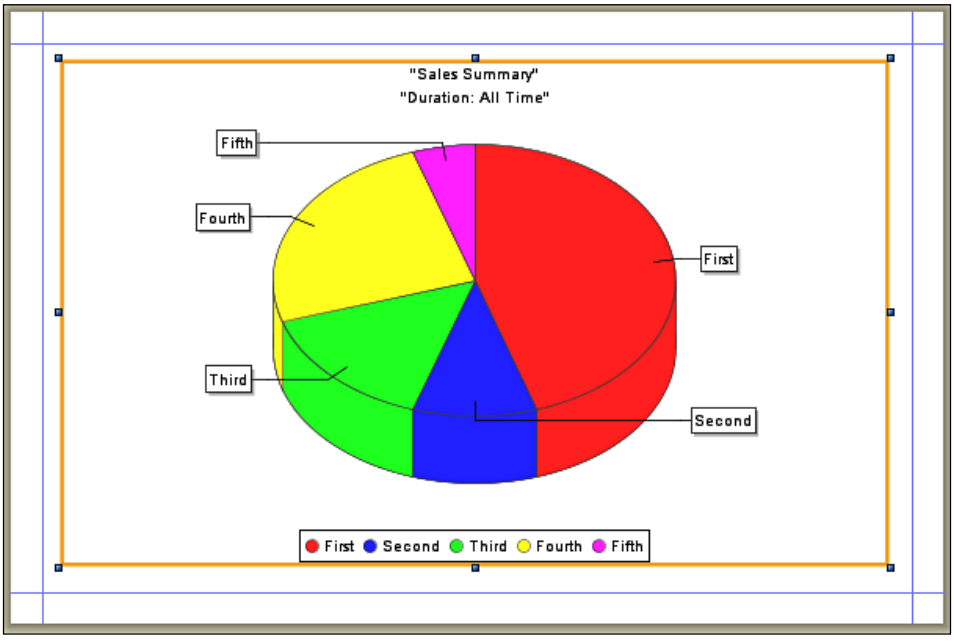
11. Press **Close**.
12. The report is created. Now, **Preview** the report with an active connection. You will see the output, as shown in the following screenshot:



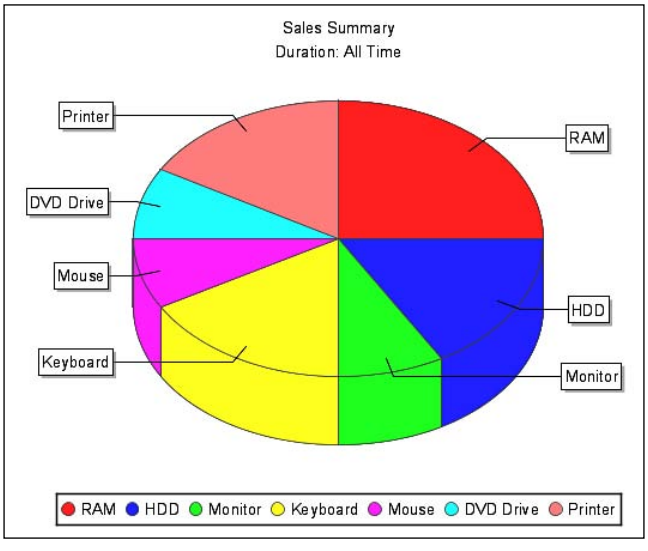
Developing a 3D pie chart report

To create a 3D pie chart, follow all the steps listed for creating a pie chart, except Step 5, where you have to choose **Pie 3D** from the list of chart options.

The initial report design will be as shown in the following screenshot:



Then if you go through all the steps, your report output will be as shown in the following screenshot:



Developing a bar chart report

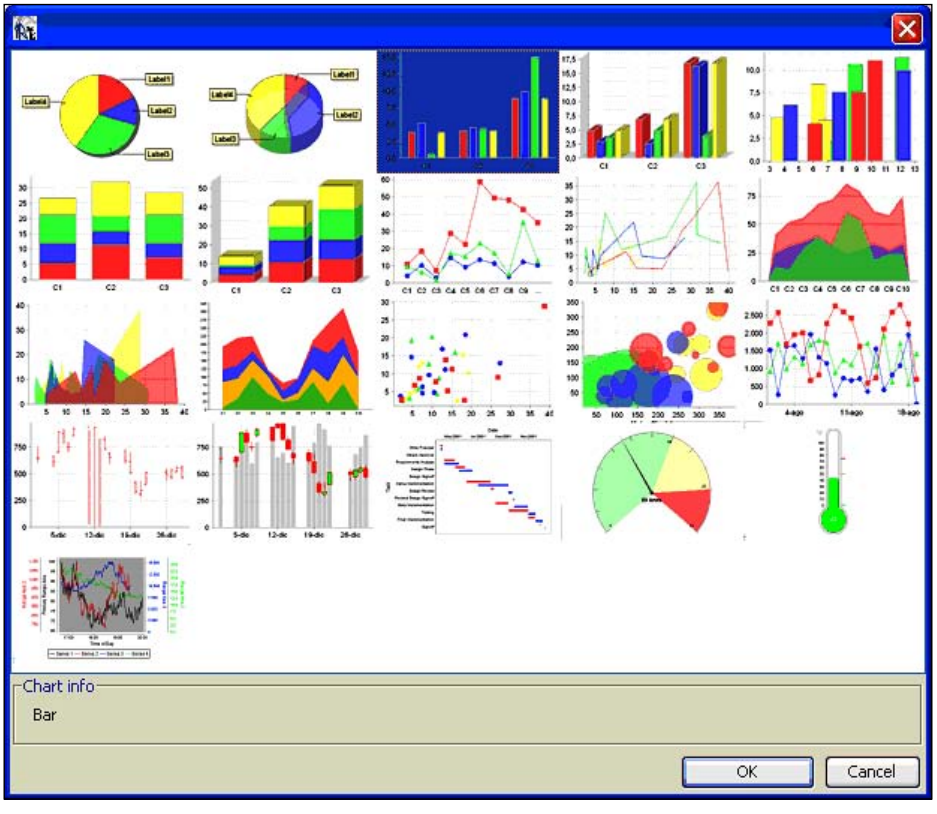
Bar chart is another type of a chart where the options are shown as standing bars.

Suppose we want to see the sales volume of individual products, then we will use the bar chart. Follow the listed steps to create a bar chart report:

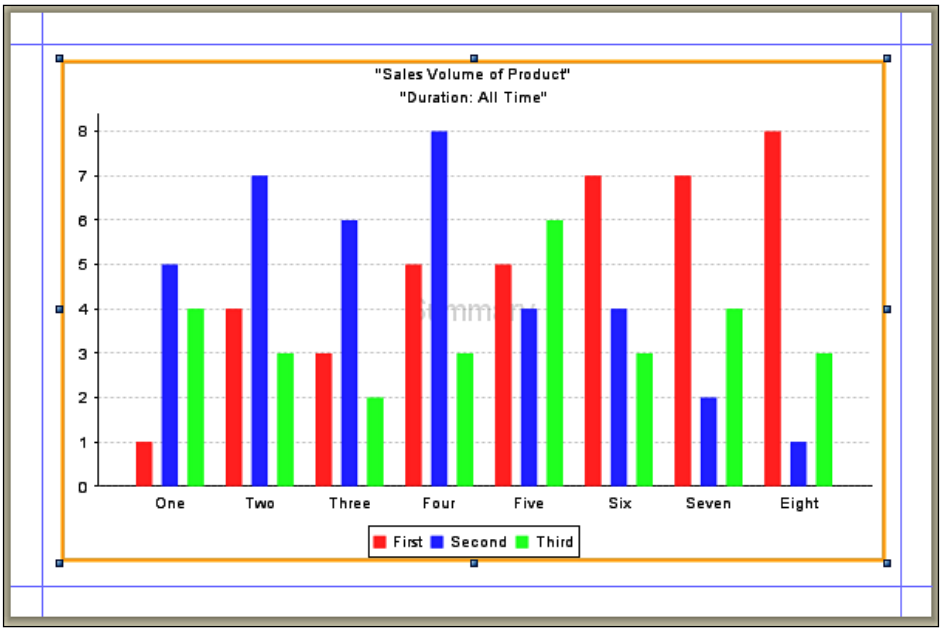
1. Create another blank report, having just the **Summary** band visible and **Band height** big enough to place the chart element in.
2. Open the **Report query** editor, and write the following SQL query:

```
SELECT name,sum(salesQuantity)
FROM Product, SalesLine
WHERE Product.productCode=SalesLine.productCode
GROUP By name
```

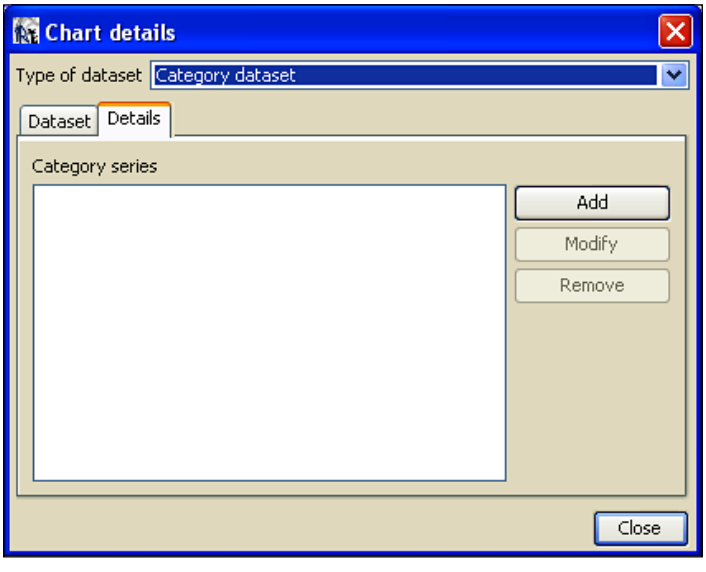
3. Place a **Bar** chart on the **Summary** band.



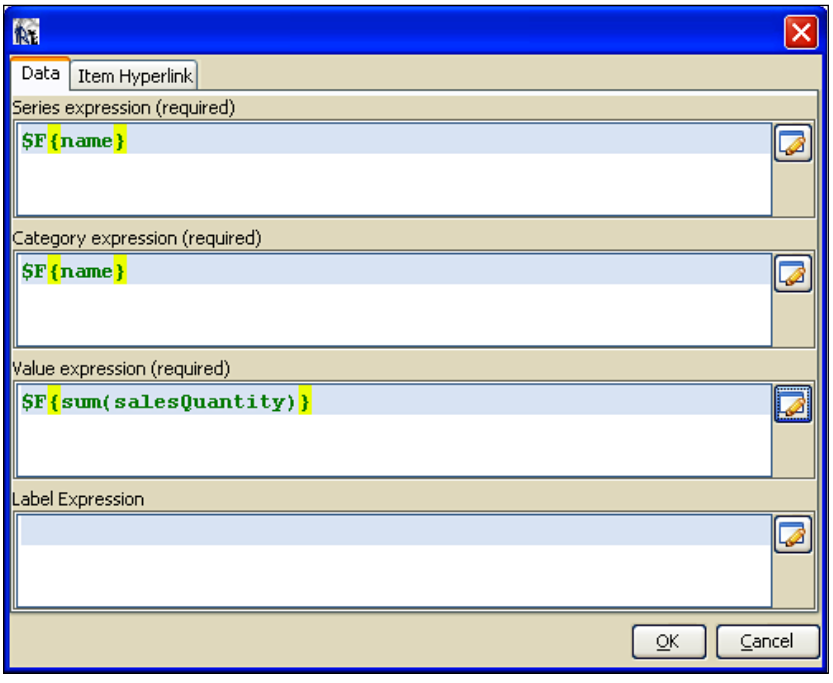
4. Go to the **Bar Chart - Properties**, and enter **"Sales Volume of Product"** as the **Title Expression** and **"Duration: All Time"** as the **Subtitle Expression**.



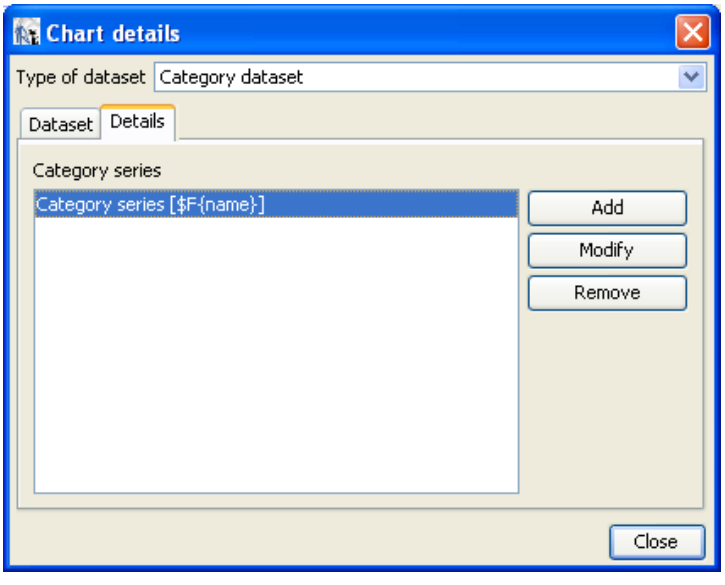
5. Go to **Chart Data** | **Details**, and press **Add**.



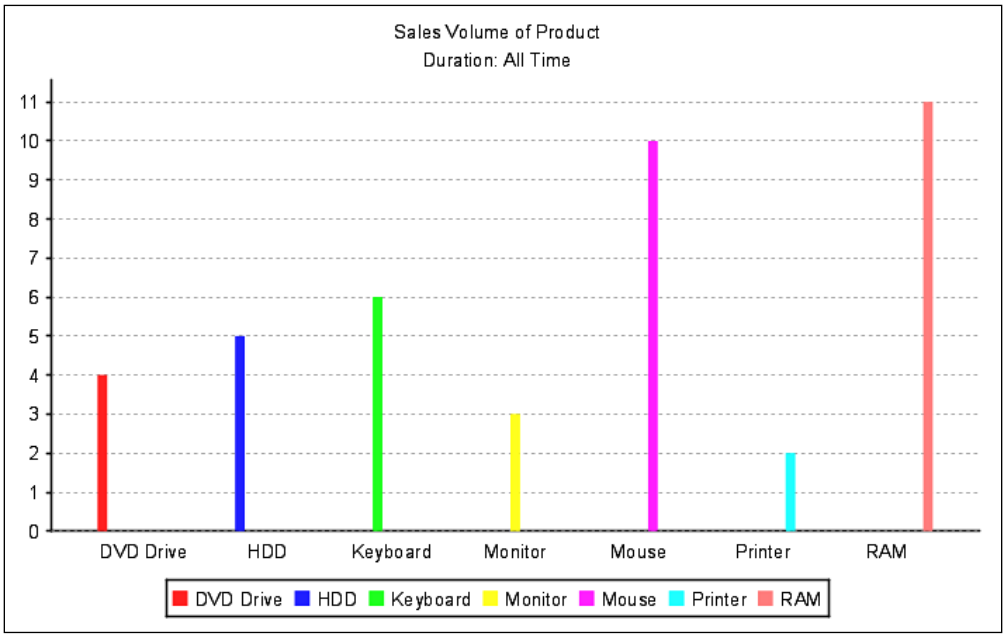
6. Set **$F{name}**, **$F{name}**, and **$F{sum(salesQuantity)}** as the **Series expression**, **Category expression**, and **Value expression** respectively.



7. Press **OK**.



8. **Preview** the report with an active connection, and see the report output as shown in the following screenshot:



Summary

In this chapter, we learned about creating reports with charts in iReport. We also learned about the different types of charts supported by iReport, the chart properties, and defining chart data expressions. We created reports with a pie chart, 3D pie chart, and bar chart. Based on these ideas, you can develop other types of charts very easily.

In the next chapter, we will learn about loading images from the database and from a static location on the hard drive for the reports.

10
Working with Images

In a software application, images have various uses. Some images are used for viewing as a logo and some are stored as binary data in the database. For example, a client's photograph can be stored in the database as binary data.

In this chapter, we shall:

- Learn how to display images from the database
- Learn how to display a static image from the hard drive
- Learn how to set a background image in a report

So let's get on with it.

Displaying an image from the database

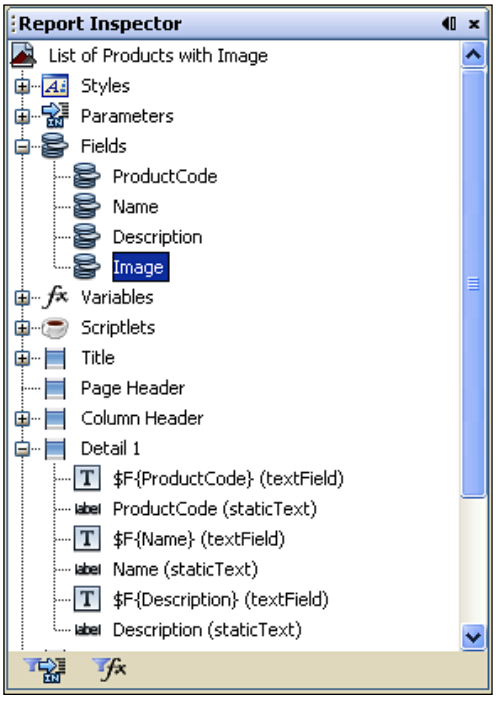
To store an image as data in the database, the required data type is `BLOB` or `LONGBLOB`. **BLOB** stands for **Binary Large Object**, which can store large images or binary data.

In our database, the `Product` table has an attribute `Image` of type `LONGBLOB`. We will display the image of all products in the report. Follow the listed steps:

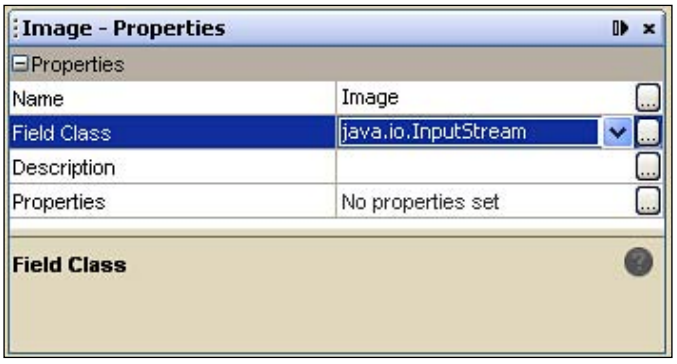
1. Create a blank report with the SQL query `SELECT * FROM Product`.
2. Drag-and-drop the **ProductCode**, **Name**, and **Description** fields from the **Report Inspector** into the **Detail** band.

3. Now the report design may look like the following screenshot:

4. Select the **Image** field from **Report Inspector | Fields**.

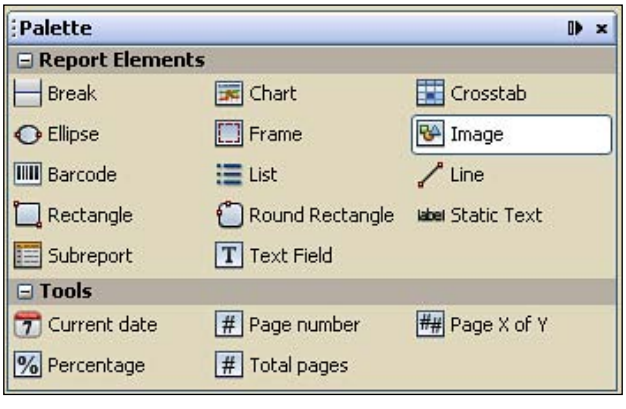


5. From the **Image - Properties**, change the **Field Class** type to **java.io.InputStream**.



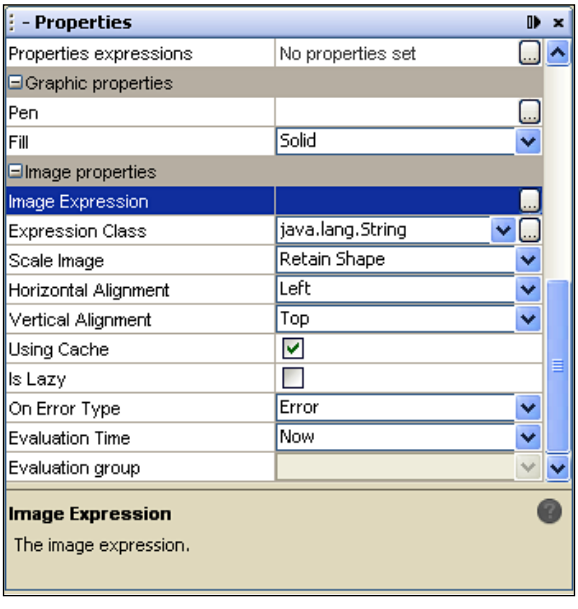
We have already mentioned that `BLOB` is the data type to store images or large textual data in a database. However, you have seen that we didn't use the word `BLOB` as the data type anywhere in iReport to build the report. By default, the data type of the **Image** field was `Object`, but the image will not be shown in this data type. Actually, images are written to, and read from, the database as `Stream`. That's why, to show images in iReport from the database, the image **Field Class** type must be `java.io.InputStream`, as it reads data from the database.

6. From **Palette** | **Report Elements**, select the **Image** tool, and drag-and-drop it on the detail band beside the fields, as shown in the design. **Cancel** the appeared file dialog box, as we are not showing static images here.



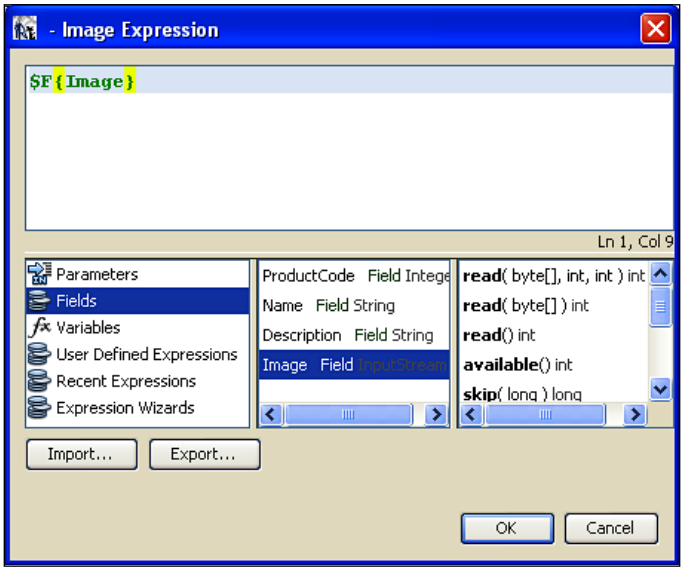
The report design is as shown in the following screenshot:

7. Select the **Image** field to see the **Properties** pane.
8. Scroll to the **Image properties**.

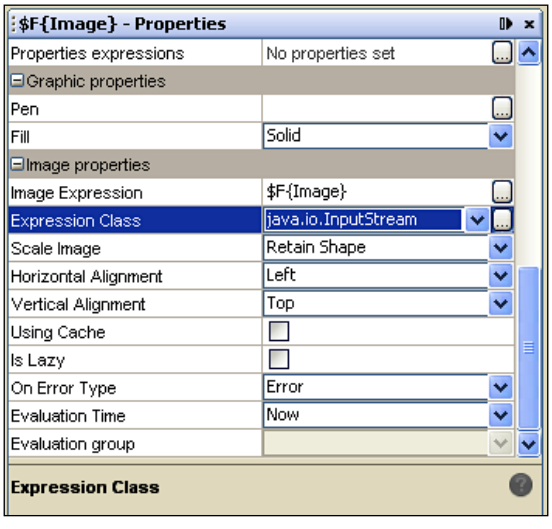


9. Open the **Image Expression** editor.

10. Double-click on the **Image** field.



11. Press **OK**.
12. From the **Image properties**, change the image **Expression Class** to **java.io.InputStream**.



13. **Preview** the report with an active connection. You will see the following output:

Scaling images

To display the images properly, the images may need to be scaled to fit the display area. The available options for scaling are: **Clip**, **Fill Frame**, **Retain Shape**, **Real size**, and **Real height**. To retain the actual shape of the image, setting **Retain Shape**, as the **Scale Image** option is recommended for this report.

- If **Clip** is selected as the **Scale Image** option, then only a part of the full image is shown (if the original image size is large), as shown in the following screenshot:

- If **Fill Frame** scaling is chosen, then the image is scaled to fit the image field drawn in the report. The following screenshot shows **Fill Frame** scaling output:

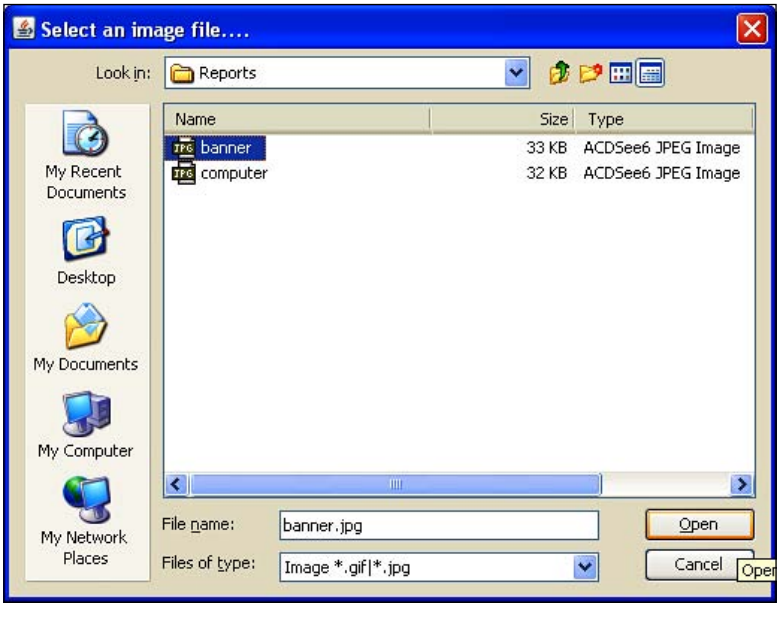
- The actual size of the image is drawn on report, if **Real size** is chosen as the **Image Scaling** option, as shown in the following screenshot:

Displaying images from the hard drive

This concept helps you to display the company logo or any static image in your report. Suppose we want to show a banner image in the **Page Header** band, then just follow the listed steps:

1. Set the **Band Height** of the **Page Header** band to **100**. Now, the report design is as shown in the following screenshot:

2. Place an **Image** tool from the **Palette** on the **Page Header** band. A file open dialog box will appear to choose the image file, as shown in the following screenshot:



3. Resize it, so that it covers the entire area of the **Page Header** band, and set **Fill Frame** as the **Scale Image** option in **Image Properties**.

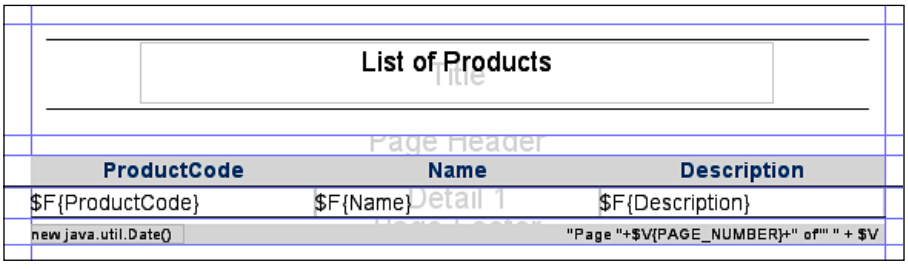
4. **Preview** the report with an active connection, and see the output, as shown in the following screenshot:

Displaying an image from the hard drive is easier than displaying an image from the database. We have to set the image class type for `BLOB` images, but when we choose an image from the hard drive, we need not do this because the image class is set in iReport, which is a `String` type expression.

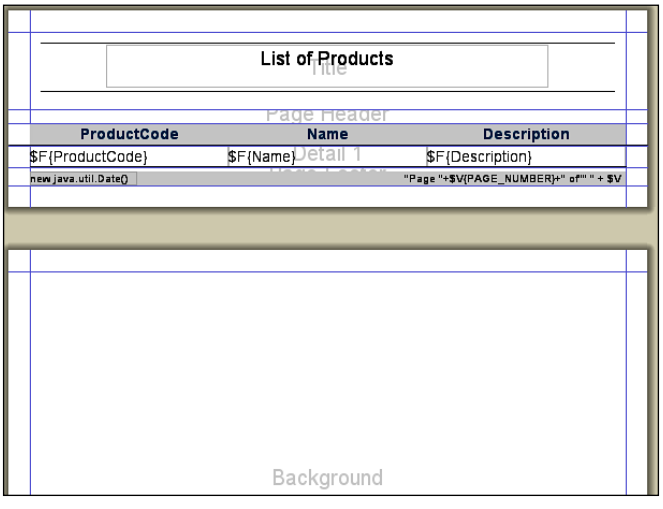
Setting a background image

Sometimes we may need to set a watermark in our report. To learn how to do this, we will create another report—`List of products without image`—and display a watermark image in the background.

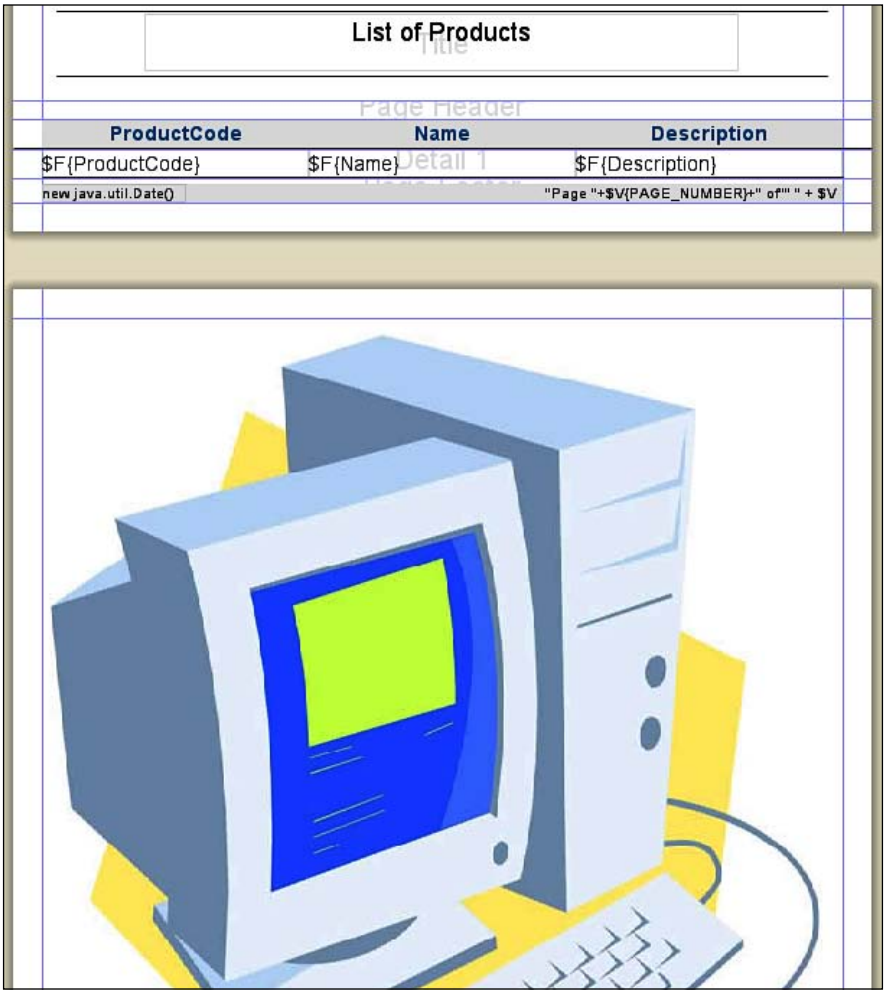
1. Create a report as we did previously, but don't include the **Image** field. The report design may look like the following screenshot:



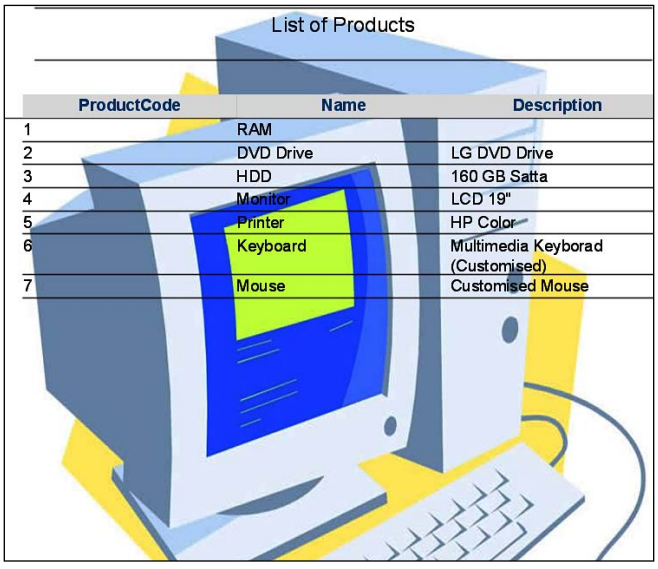
2. Set the height of the **Background** band in the **Report Inspector** to **802**. Now, a blank **Background** band has been created at the bottom of the report, as shown in the following screenshot:



3. Place the **Image** tool from the **Palette** on the **Background** band. Resize the image, so that it covers the entire area of the **Background** band, and select **Fill Frame** as the **Scale Image** option.



4. **Preview** the report with an active connection, and see the output, as shown in the following screenshot:



List of Products

ProductCode	Name	Description
1	RAM	
2	DVD Drive	LG DVD Drive
3	HDD	160 GB Satta
4	Monitor	LCD 19"
5	Printer	HP Color
6	Keyboard	Multimedia Keyborad (Customised)
7	Mouse	Customised Mouse

You might have found similarities between creating reports with `BLOB` images, creating reports with a page header image, and creating reports with a background image. The steps are almost the same, but a significant difference is in the placement of the image. In the first case, the image was in the **Detail** band, as the images are displayed as table rows. In the second case, the image was placed in the **Page Header** band. And in the last case, it was placed in the **Background** band.

Summary

We learned a lot in this chapter about working with images.

Specifically, we have covered:

- Creating a report with `BLOB` images
- Creating a report with a page header image
- Setting a background image in a report

Now that we've learned about creating different types of report, we're ready to call these reports from our Java application, which is covered in the next chapter.

11
Calling Reports from Java Applications

We have developed various types of reports so far, and all of the reports have been viewed from the iReport application. In this chapter, we will discuss how we can view report outputs from our own Java application. Before calling the report, we will create a Java application using NetBeans IDE to call iReport from it.

In this chapter, we shall learn about:

- Creating a Java application in NetBeans
- Using the JasperReports library
- Calling iReport from Java programs

So let's get on with it.

Downloading and installing NetBeans

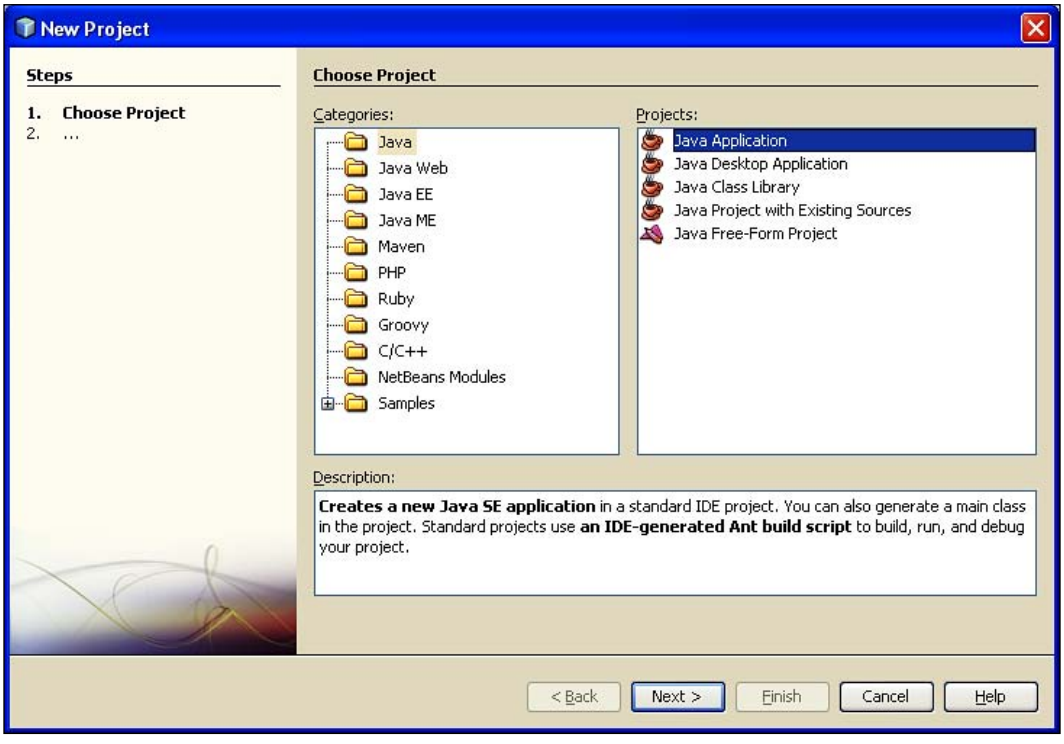
For this chapter, you need to have the NetBeans IDE installed. You can download the latest NetBeans IDE from `http://www.netbeans.org/downloads/`. The Java SE download bundle is enough for this chapter.

Creating a project in NetBeans

Creating a project is the first step of creating a Java application. When a project is created, the required directories and other files are created. After this, new classes or other necessary parts are created within the project.

We are going to create a project named **Inventory Management System**. Follow the listed steps after starting NetBeans IDE:

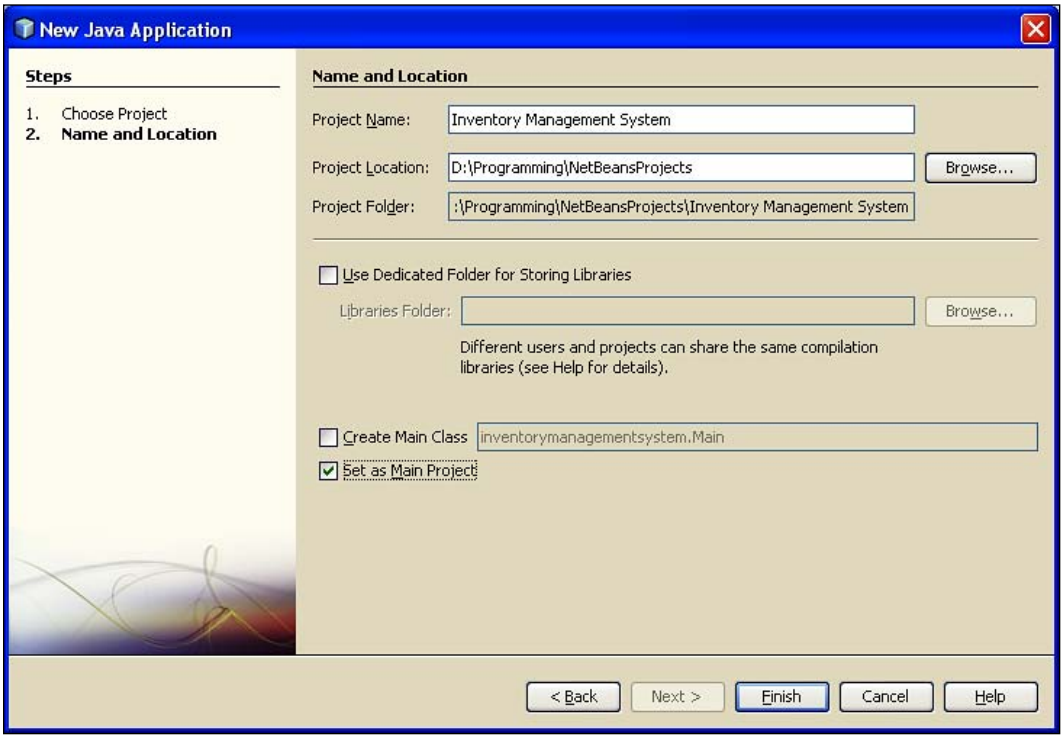
1. Go to **File | New Project...**.
2. Select **Java** from the **Categories:** section and **Java Application** from the **Projects:** section.



Using NetBeans IDE, you can create **JavaSE**, **JavaEE**, **JavaME**, and other types (**PHP**, **C++**, **Ruby**, and so on) of applications—these are all project categories. By choosing Java, you are able to create a new JavaSE desktop application from which the reports will be viewed.

3. Press **Next >**.

4. Enter **Inventory Management System** as the **Project Name:**. Click **Browse…** to enter the **Project Location:**. Uncheck the **Create Main Class** checkbox and check the **Set as Main Project** checkbox.

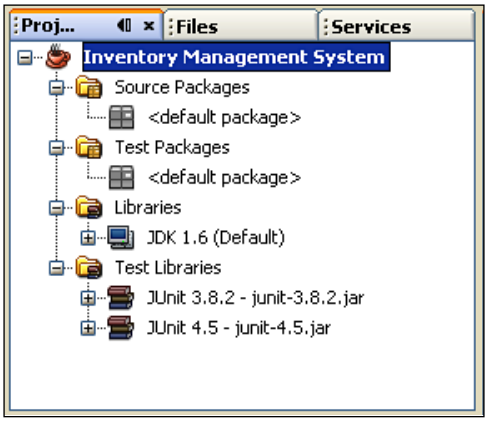


There are several classes in an application, but there is only one main class that contains the main method. This main class is the launching point of the application. When you create a project, the IDE allows you to decide whether you need the main class to be generated automatically. If you check the checkbox **Create Main Class**, a class with the main method will be generated. In our case, this is not required because our main method will be generated in another class, which we will create later on.

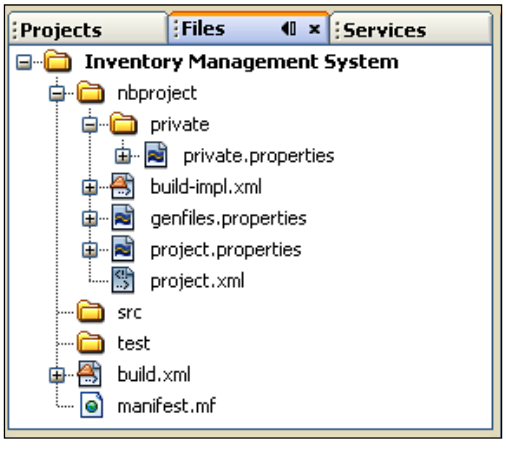
The **Set as Main Project** option allows you to choose whether the newly created application will be the main project among all of the projects of this IDE (you can manage more than one project with this IDE). The shortcut commands of the IDE work on the main project. For such advantages, we have chosen our project to be the main project.

5. Press **Finish**. Now, your project is created. On the left side of the IDE, you will find the **Projects** and **Files** tabs. Look at these to see which directories and files have been created.

   The directories and files under the **Projects** tab are as follows:



   The directories and files under the **Files** tab are as follows:



Creating the iReport viewer class

We will now create a class that will be used as the viewer for reports produced by iReport. We will create the viewer as an internal frame so that this can be added on a `JDesktopPane` within a Java swing frame (`javax.swing.JFrame`).

The viewer program will be created based on the JasperReport API. Special handling is required to use JasperReport API in your NetBeans project.

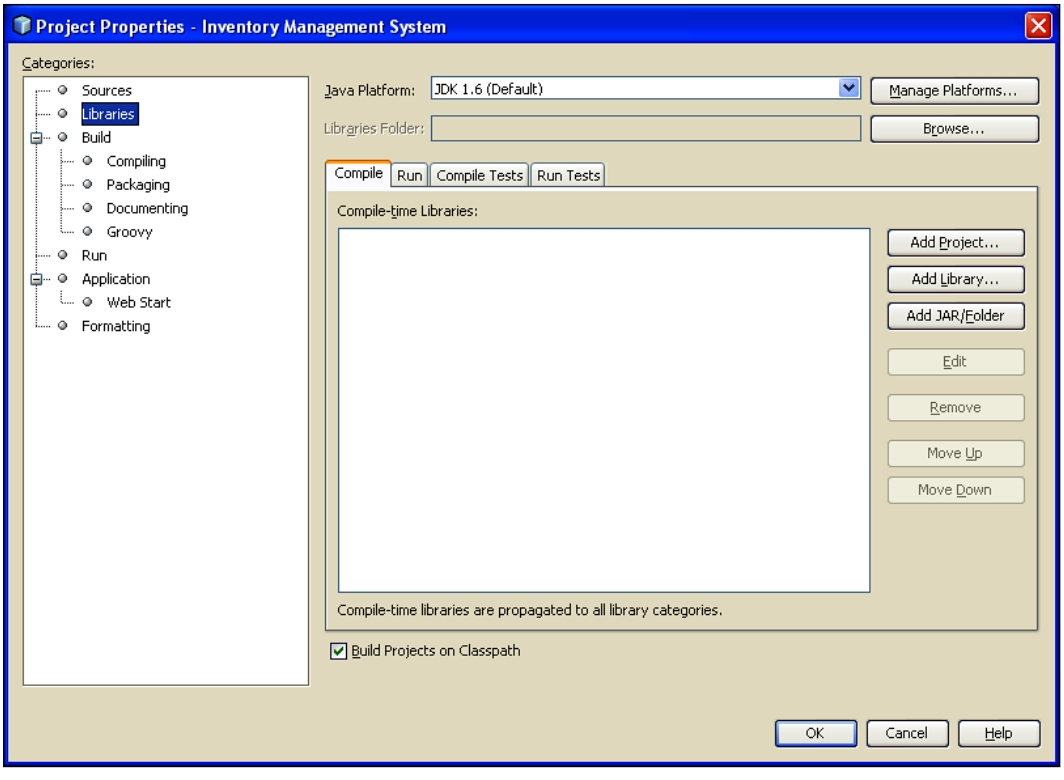
Adding JasperReports API in the NetBeans project

The API for calling report is located generally at the location `C:\Program Files\Jaspersoft\iReport-nb-3.7.0\ireport\modules` and `C:\Program Files\Jaspersoft\iReport-nb-3.7.0\ireport\modules\ext`, or the location where you installed the iReport. Before following the steps listed, create a folder named `lib` in your NetBeans project directory (the location you gave when you created the NetBeans project) and copy the JAR files from `C:\Program Files\Jaspersoft\iReport-nb-3.7.0\ireport\modules\ext` to this `lib` folder.

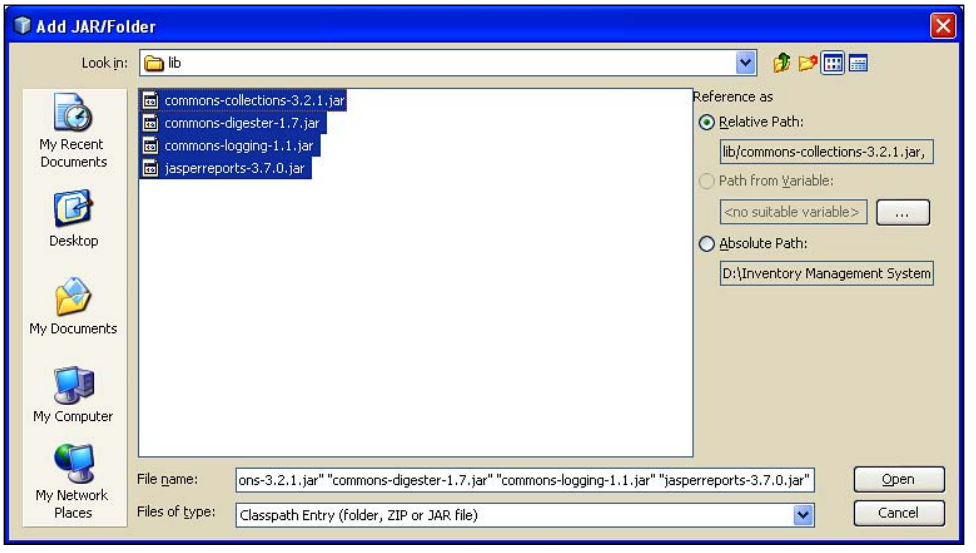
The following JAR files are required to be placed in the lib directory: `commons-collections-3.2.1.jar`, `commons-digester-1.7.jar`, `commons-logging-1.1.jar`, `jasperreports-3.7.0.jar`.

In future you may need to include other JAR files based on the reports you call. Follow the listed steps to add the library files:

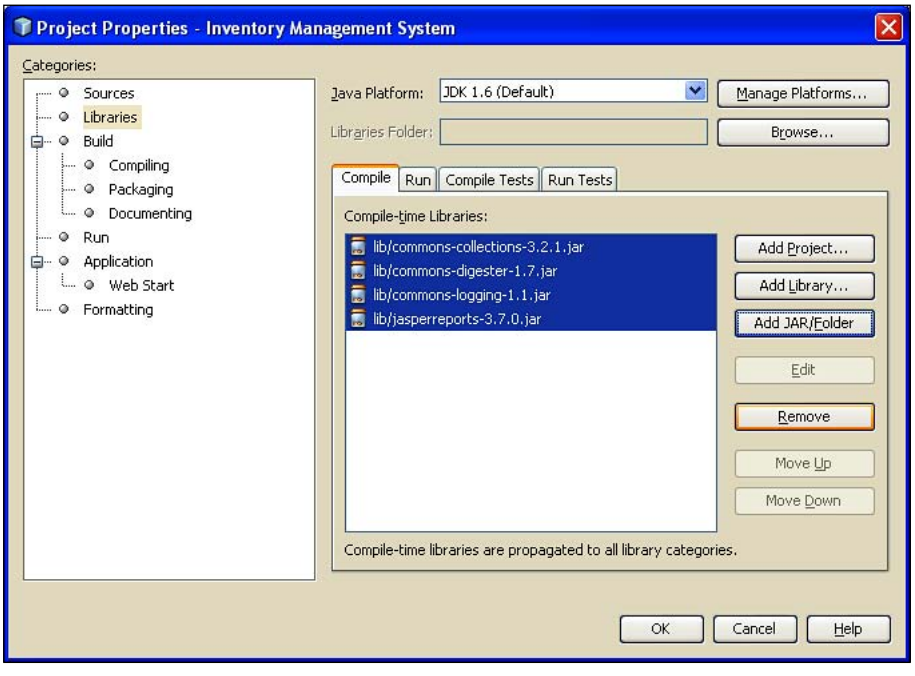
1. Right-click on **Libraries** in the **Projects** tab, and click on **Properties**.



2. Click on **Add JAR/Folder**.
3. Open the path to the `lib` directory of your project, and select the JAR files.

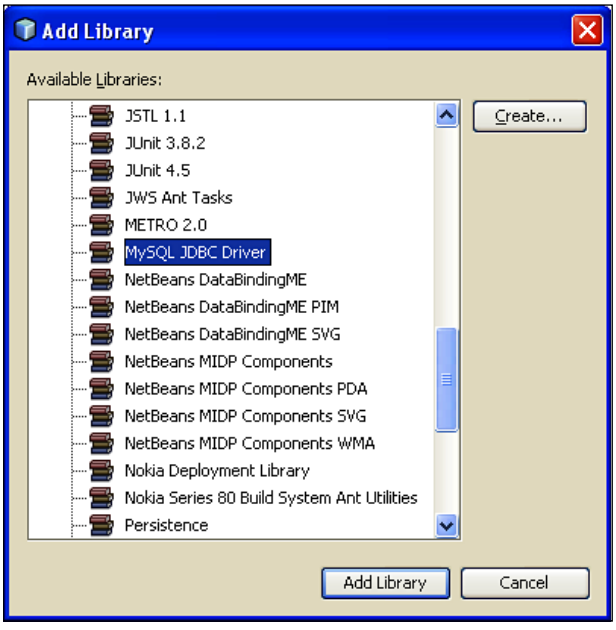


4. Select **Relative Path:** as the **Reference as** option.
5. Press **Open**.



6. Press **OK**.

7. We also need to add the MySQL JDBC driver to our library. Go to **Projects | Libraries**, right-click on it and select **MySQL JDBC Driver**.

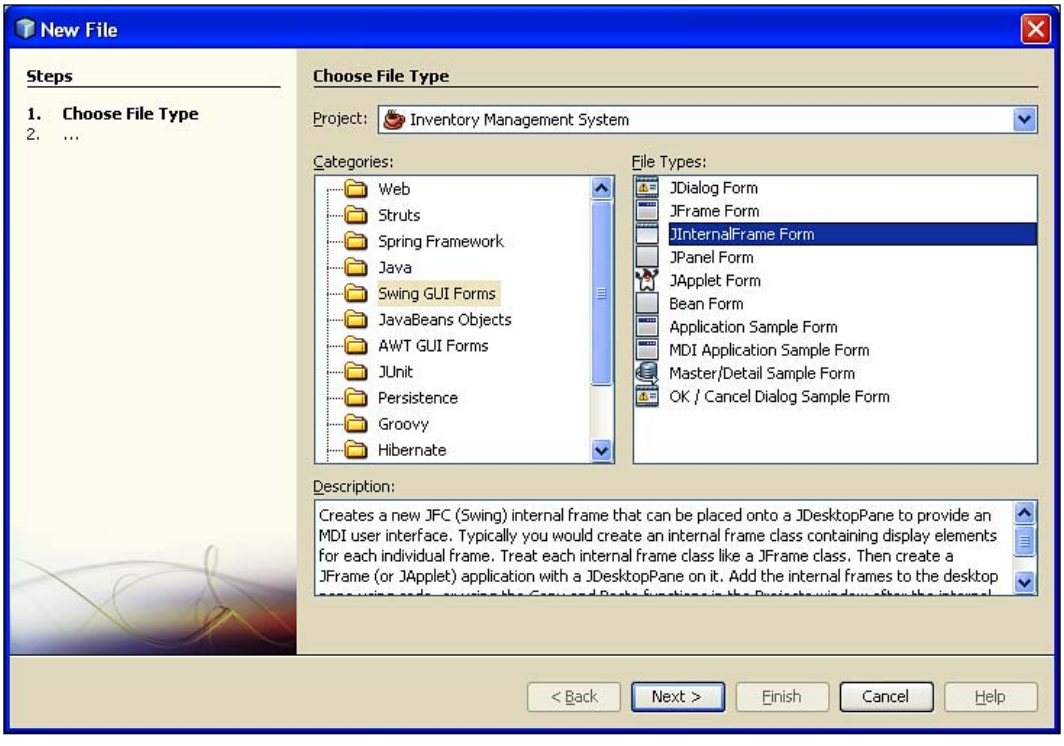


8. Press **Add Library**.

Creating the viewer class

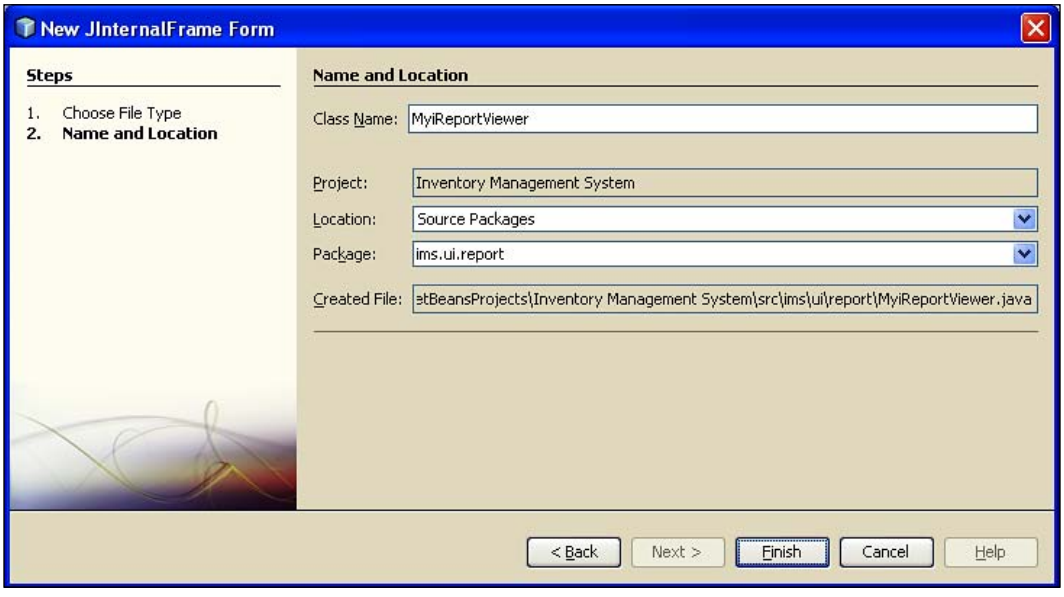
Now, we are ready to create the viewer class.

1. Go to **File** | **New File…**.
2. Select **Swing GUI Forms** from the **Categories:** section, and **JInternalFrame Form** from the **File Types:** section.



3. Press **Next >**.
4. Enter **MyiReportViewer** as the **Class Name:**.
5. Enter **ims.ui.report** as the **Package** (ims stands for Inventory Management System and ui stands for User Interface).

6. Press **Finish**.



7. The **Design** view is opened now. Press **Source** to view the source code as follows:

```
package ims.ui.report;
/**
 *
 * @author Shamsuddin Ahammad
 */
public class MyiReportViewer extends javax.swing.JInternalFrame {
/** Creates new form MyiReportViewer */
public MyiReportViewer() {
       initComponents();
    }
/** This method is called from within the constructor to
  * initialize the form.
  * WARNING: Do NOT modify this code. The content of this method
is
  * always regenerated by the Form Editor.
  */
  @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed"
                    desc="Generated Code">
```

```
            private void initComponents() {
            javax.swing.GroupLayout layout = new
            javax.swing.GroupLayout(getContentPane());
            getContentPane().setLayout(layout);
            layout.setHorizontalGroup(
            layout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 394, Short.MAX_VALUE)
            );
            layout.setVerticalGroup(
            layout.createParallelGroup(
                            javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 274, Short.MAX_VALUE)
            );
            pack();
            } // </editor-fold>
             // Variables declaration - do not modify
            // End of variables declaration
}
```

8. Now, replace the constructor (`public MyiReportViewer`) with the following one:

```
  private MyiReportViewer()
    {
      super("Report Viewer",true,true,true,true);
      initComponents();
      setBounds(10,10,600,500);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }
```

9. Add a parameterized constructor, as follows:

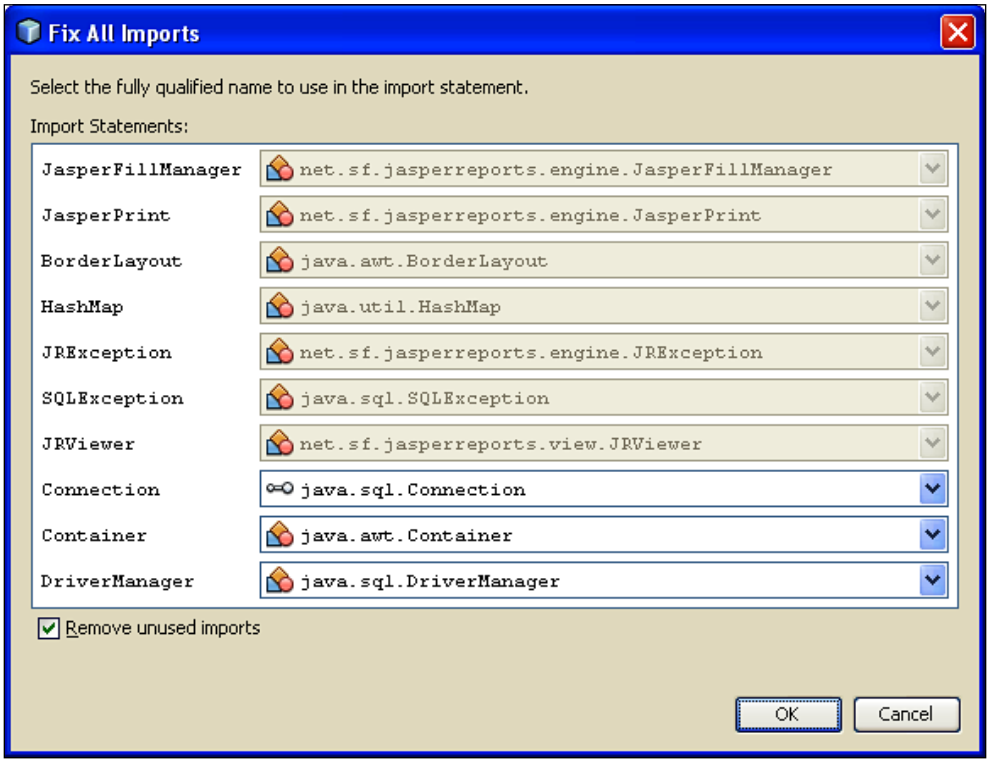
```
  public MyiReportViewer(String fileName,HashMap parameter)
    {
      this();
      try
    {
        /* load the required JDBC driver and create the connection
           here JDBC Type Four Driver for MySQL is used*/
    Class.forName("com.mysql.jdbc.Driver");
    Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/inventory
","root","packt");
```

```
        /*(Here the parameter file should be in .jasper extension
           i.e., the compiled report)*/
      JasperPrint print = JasperFillManager.fillReport(
                                        fileName, parameter, con);
      JRViewer viewer=new JRViewer(print);
      Container c=getContentPane();
      c.setLayout(new BorderLayout());
      c.add(viewer);
    }
      catch(ClassNotFoundException cnfe)
        {
          cnfe.printStackTrace();
        }
      catch(SQLException sqle)
        {
          sqle.printStackTrace();
        }
      catch(JRException jre)
        {
          jre.printStackTrace();
        }
    }
```

10. After writing the code, you will see a lot of errors. Don't worry! This is because the packages of the used classes or interfaces (`HashMap`, `Connection`, `DriverManager`, `JasperPrint`, `JasperFillManager`, `JRViewer`, `Container`, `BorderLayout`, `ClassNotFoundException`, `SQLException`, `JRException`) are not imported. To remove the errors, go to **Source | Fix Imports**, and select as follows:

Class name	Package and class
`JasperFillManager`	`net.sf.jasperreports.engine.JasperFillManager`
`JasperPrint`	`net.sf.jasperreports.engine.JasperPrint`
`BorderLayout`	`java.awt.BorderLayout`
`HashMap`	`java.util.HashMap`
`JRException`	`net.sf.jasperreports.engine.JRException`
`SQLException`	`java.sql.SQLException`
`JRViewer`	`net.sf.jasperreports.view.JRViewer`
`Connection`	`java.sql.Connection`
`Container`	`java.awt.Container`
`DriverManager`	`java.sql.DriverManager`

The **Fix All Imports** window is as shown in the following screenshot:



11. Press **OK** and see that the following code is generated in your source code (top of your program code):

```
import java.awt.BorderLayout;
import java.awt.Container;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.HashMap;
import net.sf.jasperreports.engine.JRException;
import net.sf.jasperreports.engine.JasperFillManager;
import net.sf.jasperreports.engine.JasperPrint;
import net.sf.jasperreports.view.JRViewer;
```

12. Add another parameterized constructor, as follows:

```
public MyiReportViewer(String fileName)
  {
    this(fileName,null);
  }
```

13. Now the code is complete. The full code is as follows:

```
  package ims.ui.report;
  import java.awt.BorderLayout;
  import java.awt.Container;
  import java.sql.Connection;
  import java.sql.DriverManager;
  import java.sql.SQLException;
  import java.util.HashMap;
  import net.sf.jasperreports.engine.JRException;
  import net.sf.jasperreports.engine.JasperFillManager;
  import net.sf.jasperreports.engine.JasperPrint;
  import net.sf.jasperreports.view.JRViewer;
/**
 *
 * @author Shamsuddin Ahammad
 */
public class MyiReportViewer extends javax.swing.JInternalFrame {
    /** Creates new form MyiReportViewer */
    private MyiReportViewer()
    {
      super("Report Viewer",true,true,true,true);
      initComponents();
      setBounds(10,10,600,500);
      setDefaultCloseOperation(DISPOSE_ON_CLOSE);
    }
public MyiReportViewer(String fileName)
    {
      this(fileName,null);
    }
public MyiReportViewer(String fileName,HashMap parameter)
    {
      this();
      try
    {
/* load the required JDBC driver and create the connection
   here JDBC Type Four Driver for MySQL is used*/
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection(
          "jdbc:mysql://localhost:3306/inventory","root","packt");
/*(Here the parameter file should be in .jasper extension i.e.,
  the compiled report)*/
      JasperPrint print = JasperFillManager.fillReport(
                                        fileName, parameter, con);
      JRViewer viewer=new JRViewer(print);
```

```
            Container c=getContentPane();
            c.setLayout(new BorderLayout());
            c.add(viewer);
          }
            catch(ClassNotFoundException cnfe)
              {
                cnfe.printStackTrace();
              }
            catch(SQLException sqle)
              {
                sqle.printStackTrace();
              }
            catch(JRException jre)
              {
                jre.printStackTrace();
              }
          }
        /** This method is called from within the constructor to
          * initialize the form.
          * WARNING: Do NOT modify this code. The content of this method is
          * always regenerated by the Form Editor.
          */
            @SuppressWarnings("unchecked")
            // <editor-fold defaultstate="collapsed" desc="Generated Code">
            private void initComponents() {
            javax.swing.GroupLayout layout =
                            new javax.swing.GroupLayout(getContentPane());
            getContentPane().setLayout(layout);
            layout.setHorizontalGroup(
            layout.createParallelGroup(
                                javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 394, Short.MAX_VALUE)
            );
            layout.setVerticalGroup(
            layout.createParallelGroup(
                                javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 274, Short.MAX_VALUE)
            );
            pack();
            }  // </editor-fold>
              // Variables declaration - do not modify
             // End of variables declaration
          }
```

Accessing the database

To show data, reports need to access a database. **Java Database Connectivity** (**JDBC**) technology is used to access a database from a Java application. Sun Microsystems defines JDBC as follows:

> *The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases – SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.*
>
> *JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.*

To access a database using JDBC, the following steps are required:

1. Loading the driver: The JDBC driver is loaded first. We use the `JDBC Type four` driver for MySQL (`com.mysql.jdbc.Driver`). The static method `forName` of class `java.lang.Class` is used for loading the driver. That's why we have written `Class.forName("com.mysql.jdbc.Driver").`
2. Connecting to a database: An instance of the interface `java.sql.Connection` is created to connect to a database. Static method `getConnection` of class `java.sql.DriverManager` is used to establish a connection to a database. This method has three parameters:
    - URL: A database URL of the form `jdbc:subprotocol://serverhost:port/databaseName`
    - User: The database user on whose behalf the connection is being made
    - Password: The user's password

In our case, `jdbc:mysql://localhost:3306/inventory` is the URL, `root` is the user, and `packt` is the password.

Filling the report with data

The static method `fillReport` of the class `net.sf.jasperreports.engine.JasperFillManager` fills the compiled report design, loaded from the specified file, and returns the generated report object. It has three parameters:

- `sourceFileName`: Source file containing the compiled report design
- `parameters`: Report parameters map
- `connection`: JDBC connection object to be used for executing the internal report SQL query

Considering this, we have written `JasperPrint print = JasperFillManager.fillReport(fileName, parameter, con)` for filling the report with data from the database.

Viewing the report

Now that we have filled the report with data, we will view it. The JasperReport class, `net.sf.jasperreports.view.JRViewer`, creates a JPanel form to view the report. We have added this form on the `Container` of the JInternalFrame.

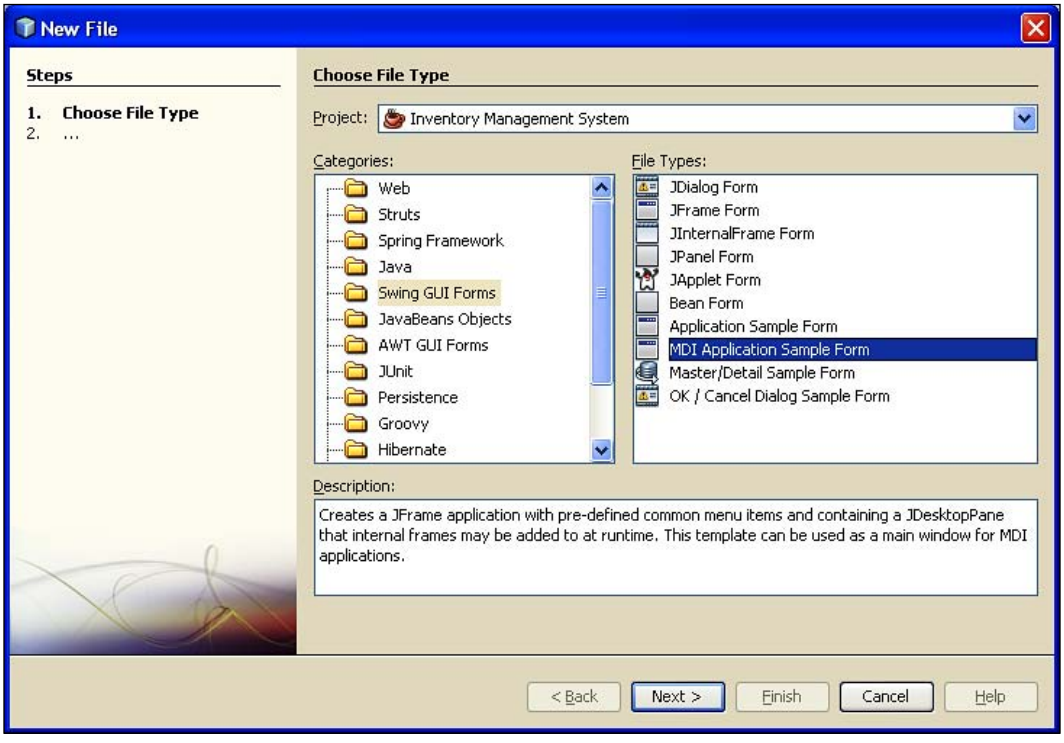
Calling the viewer class

We will now create another GUI that calls `MyiReportViewer` and shows the report output. Before following the steps listed, create a folder, `reports`, at the location where you have created the NetBeans project, place all of your reports in this folder, and also place all the report static images in the root folder of the project. The reports must have the compiled version (file with jasper extension) to be called from `MyiReportViewer`. It is assumed that you have done this and follow the steps listed.

Creating GUI with menus

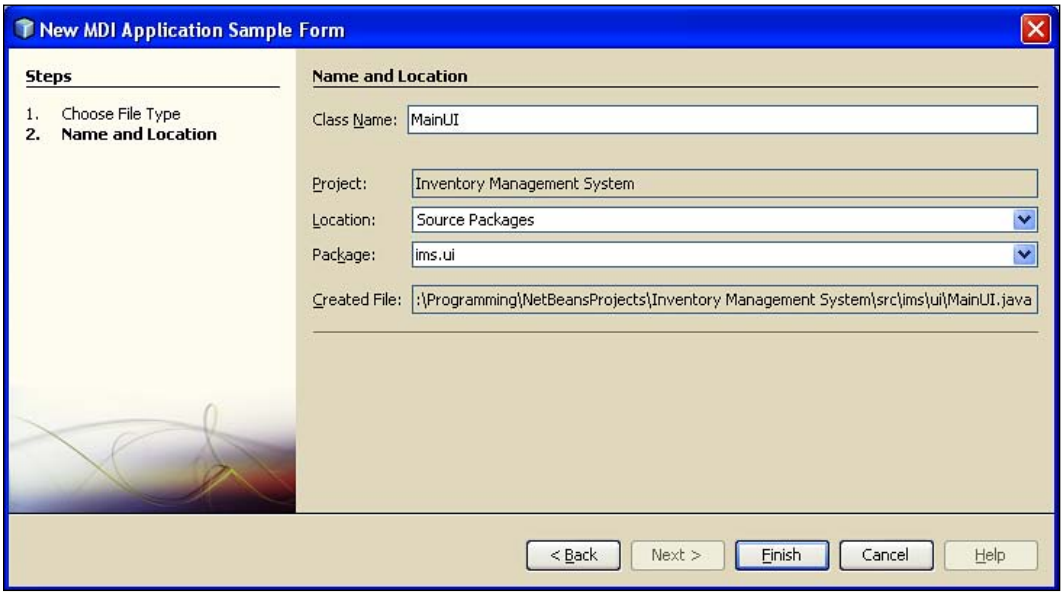
We are going to create a GUI (JFrame form with menus) from which the reports will be called when the menu item is clicked by the user.

1. Go to **File** | **New File...**.
2. Select **Swing GUI Forms** from the **Categories:** section and **MDI Application Sample Form** from the **File Types:** section.

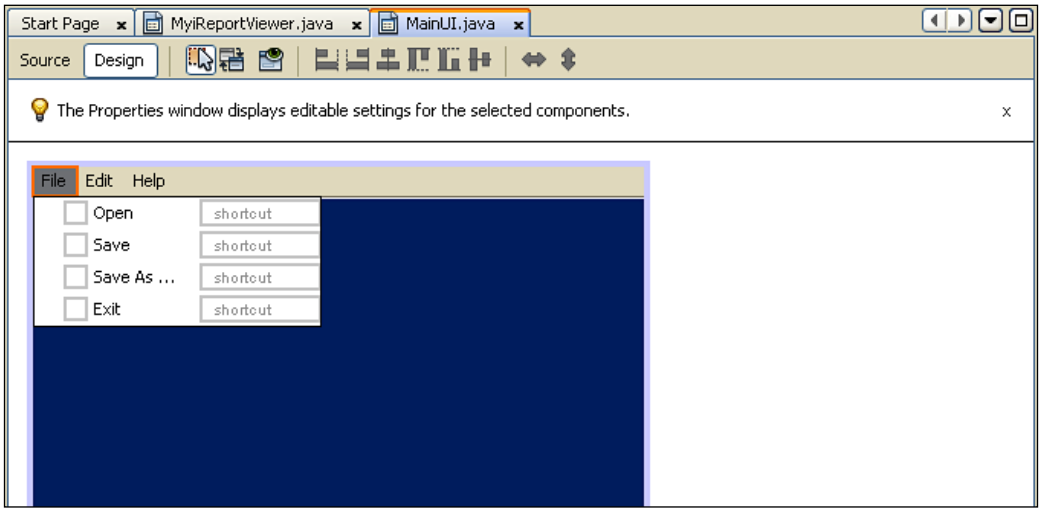


MDI Sample Application Form is a JFrame application with pre-defined common menu items, containing a JDesktopPane to which internal frames may be added, at runtime. This template can be used as the main window for MDI applications. As our report viewer is a JInternalFrame, we have chosen MDI form.

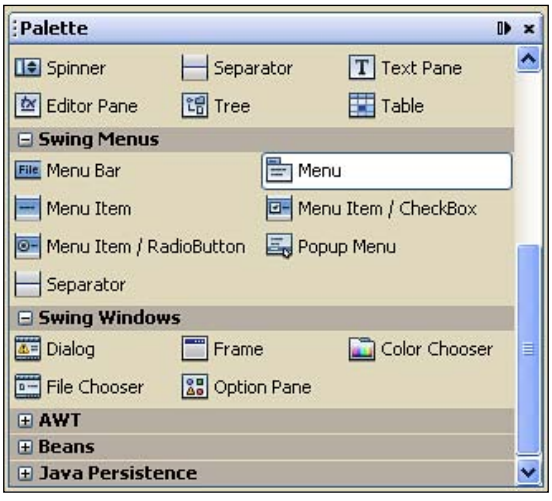
3. Press **Next >**.
4. Enter **MainUI** as the **Class Name:**.
5. Select **ims.ui** as the **Package**.
6. Press **Finish**.



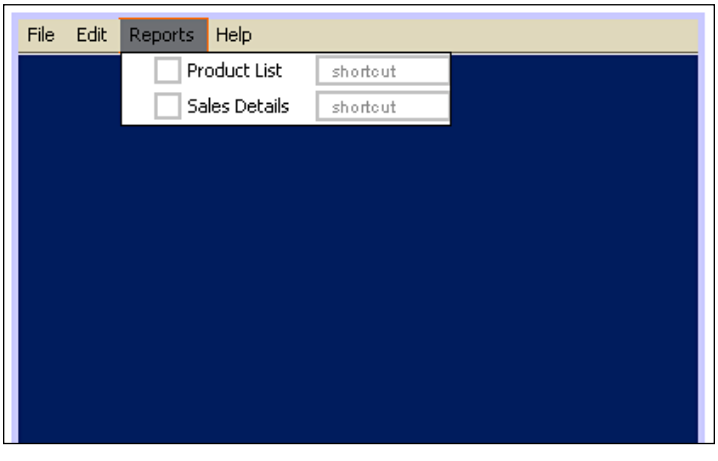
7. An initial UI is created, and its **Design** view is as shown in the following screenshot:



8. From the **Palette** on the right side, find **Swing Menus**. Select **Menu** and drag-and-drop it in between the **Edit** and **Help** menu.



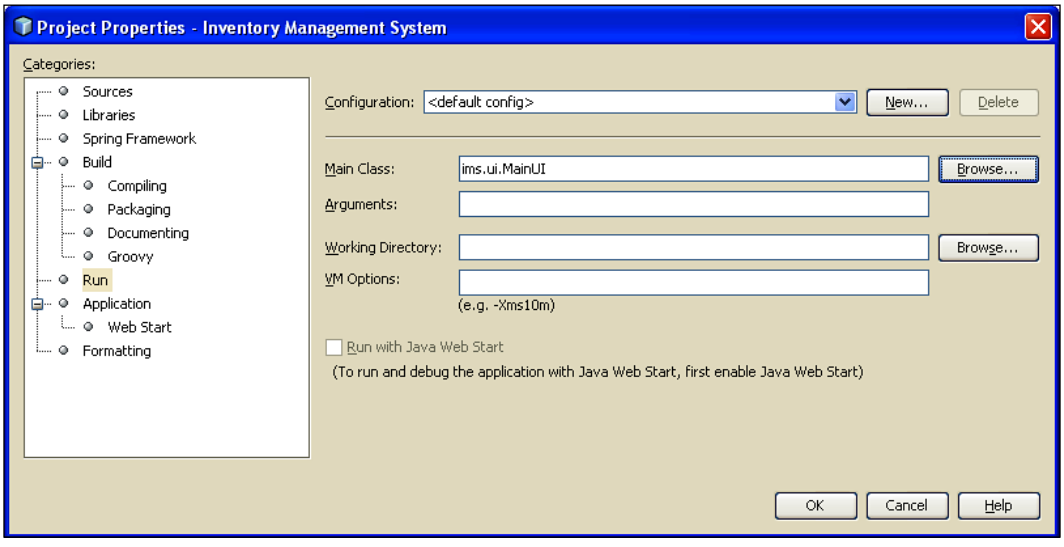
9. Select the newly dragged menu (may be named as **jMenu1**, right-click on it, press **Edit Text,** and write **Reports**.
10. Again, select the menu, right-click on it, press **Change Variable Name**, enter **reportsMenu**, and press **OK**.
11. In the same way, drag a **Menu Item** and drop it on the **Reports** menu. Set **Product List** as **Text** and enter **productListMenuItem** as the **Variable Name**.
12. Add another **Menu Item** (**Text**: **Sales Details** and **Variable Name**: **salesDetailsMenu**). Now the design looks as shown in the following screenshot:



13. Now go to **Source**, and add the following code just below the `initComponents()` method call, within the constructor, `MainUI`, to make the frame full screen:

```
setSize(java.awt.Toolkit.getDefaultToolkit().getScreenSize());
```

14. Go to **File** | **Project Properties** | **Run** and press the **Browse...** button of the **Main Class** to choose **ims.ui.MainUI**, press **Select Main Class,** and then press **OK**.



We have two classes in our project—one is `MyiReportViewer` and another is `MainUI`. We have to set the class, from where the application will start. `MyiReportViewer` has no main method; that's why it cannot be executed directly. However, if you have several classes in your project, you have to set the main class, which we have seen previously.

15. Go to **Run** | **Run Main Project**, and see the UI output.

Calling a report without a parameter

We have a report, `List of Products with Image.jasper` (created in Chapter 10, *Working with Images*), which has no parameter. We are going to call this report from `MainUI`. This report will be called when the user clicks **Product List**.

1. Select the **Product List** menu item from the **Design** view. Right-click on it, and choose **Events | Action | actionPerformed**.
2. You will see the source code within a method as follows:

```
private void productListMenuItemActionPerformed(java.awt.event.
ActionEvent evt)
{
       // TODO add your handling code here:
}
```

3. Remove the comment, `// TODO add your handling code here:`, and add the following code:

```
try
{
MyiReportViewer myiReportViewer = new
MyiReportViewer("reports/List of Products with Image.jasper");
myiReportViewer.setBounds(0, 0, desktopPane.getWidth(),
desktopPane.getHeight());
myiReportViewer.setVisible(true);
desktopPane.add(myiReportViewer);
myiReportViewer.setSelected(true);
}
catch (PropertyVetoException pve)
{
    pve.printStackTrace();
}
```

4. Run the project, and click on the **Product List** menu. You will see the following output:

We created two parameterized constructors in the class, `MyiReportViewer`. One has a single parameter (`String filename`) and the other has two parameters (`String fileName`, `HashMap parameter`). The first constructor is used when there is no parameter in the report. What we need to do is:

1. Create the instance of `MyiReportViewer` by writing the following code:

```
MyiReportViewer myiReportViewer = new MyiReportViewer("reports/
List of Products with Image.jasper");
```

2. Set the size and location of `myiReportViewer` by writing the following code. Here we have set the width and height equal to those of `desktopPane`; that's why the report will have a full screen size.

```
myiReportViewer.setBounds(0, 0, desktopPane.getWidth(),
desktopPane.getHeight());
```

3. Make the internal frame visible.
4. Add the instance on the `desktopPane`.

If you want the instance to be on top of other internal frames, call the `setSelected` method by entering `Boolean true` as the argument.

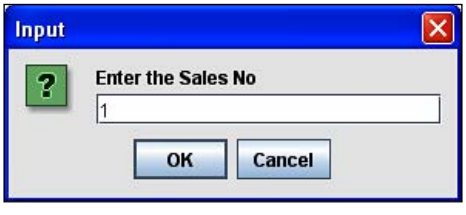
Calling a report with a parameter

We had a report, `SalesDetails.jasper` (created in Chapter 5, *Using Parameters*), which has a parameter named `SalesNo`. Let's call this parameterized report.

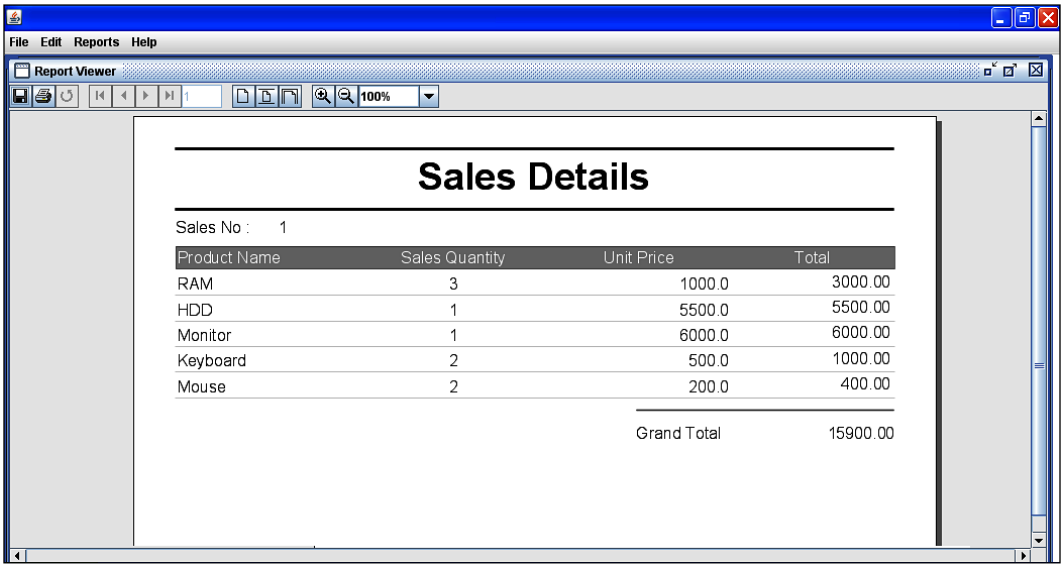
1. Follow all the steps covered in the previous section (*Calling a report without a parameter*), but write the following code for event handling:

```
String input=JOptionPane.showInputDialog("Enter the Sales No");
if(input!=null)
{
  try
    {
        int salesNo=Integer.parseInt(input);
        HashMap parameters=new HashMap();
        parameters.put("salesNo",salesNo);
        MyiReportViewer myiReportViewer = new MyiReportViewer(
                          "reports/SalesDetails.
jasper",parameters);
        myiReportViewer.setBounds(
              0, 0, desktopPane.getWidth(), desktopPane.
getHeight());
        myiReportViewer.setVisible(true);
        desktopPane.add(myiReportViewer);
        myiReportViewer.setSelected(true);
    }
      catch (PropertyVetoException pve)
        {
          pve.printStackTrace();
        }
      catch(NumberFormatException nfe)
        {
          JOptionPane.showMessageDialog(
                                  this,"Please input numbers
only");
        }
}
```

2. Now, run the project, and click on the **SalesDetails** menu item. It will ask for an input. Enter **1** as the **Sales No**:



You will see that the report output is as shown in the following screenshot:



The complete code of the program is as follows:

```
package ims.ui;
import ims.ui.report.MyiReportViewer;
import java.beans.PropertyVetoException;
import java.util.HashMap;
import javax.swing.JOptionPane;
/**
 *
 * @author Shamsuddin Ahammad
 */
public class MainUI extends javax.swing.JFrame {
    /** Creates new form MainUI */
  public MainUI() {
      initComponents();
      setSize(java.awt.Toolkit.getDefaultToolkit().getScreenSize());
      }
    /** This method is called from within the constructor to
      * initialize the form.
      * WARNING: Do NOT modify this code.
        The content of this method is
      * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
  private void initComponents() {
```

```
desktopPane = new javax.swing.JDesktopPane();
menuBar = new javax.swing.JMenuBar();
fileMenu = new javax.swing.JMenu();
openMenuItem = new javax.swing.JMenuItem();
saveMenuItem = new javax.swing.JMenuItem();
saveAsMenuItem = new javax.swing.JMenuItem();
exitMenuItem = new javax.swing.JMenuItem();
editMenu = new javax.swing.JMenu();
cutMenuItem = new javax.swing.JMenuItem();
copyMenuItem = new javax.swing.JMenuItem();
pasteMenuItem = new javax.swing.JMenuItem();
deleteMenuItem = new javax.swing.JMenuItem();
reportsMenu = new javax.swing.JMenu();
productListMenuItem = new javax.swing.JMenuItem();
salesDetailsMenuItem = new javax.swing.JMenuItem();
helpMenu = new javax.swing.JMenu();
contentMenuItem = new javax.swing.JMenuItem();
aboutMenuItem = new javax.swing.JMenuItem();
setDefaultCloseOperation(
                      javax.swing.WindowConstants.EXIT_ON_CLOSE);
  fileMenu.setText("File");
  openMenuItem.setText("Open");
  fileMenu.add(openMenuItem);
  saveMenuItem.setText("Save");
  fileMenu.add(saveMenuItem);
  saveAsMenuItem.setText("Save As ...");
  fileMenu.add(saveAsMenuItem);
  exitMenuItem.setText("Exit");
  exitMenuItem.addActionListener(
                            new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEvent evt) {
  exitMenuItemActionPerformed(evt);
  }
});
  fileMenu.add(exitMenuItem);
  menuBar.add(fileMenu);
  editMenu.setText("Edit");
  cutMenuItem.setText("Cut");
  editMenu.add(cutMenuItem);
  copyMenuItem.setText("Copy");
  editMenu.add(copyMenuItem);
  pasteMenuItem.setText("Paste");
  editMenu.add(pasteMenuItem);
  deleteMenuItem.setText("Delete");
```

```
    editMenu.add(deleteMenuItem);
    menuBar.add(editMenu);
    reportsMenu.setText("Reports");
    productListMenuItem.setText("Product List");
    productListMenuItem.addActionListener(
                              new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      productListMenuItemActionPerformed(evt);
      }
    });
    reportsMenu.add(productListMenuItem);
    salesDetailsMenuItem.setText("Sales Details");
    salesDetailsMenuItem.addActionListener(
                              new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
      salesDetailsMenuItemActionPerformed(evt);
      }
    });
    reportsMenu.add(salesDetailsMenuItem);
    menuBar.add(reportsMenu);
    helpMenu.setText("Help");
    contentMenuItem.setText("Contents");
    helpMenu.add(contentMenuItem);
    aboutMenuItem.setText("About");
    helpMenu.add(aboutMenuItem);
    menuBar.add(helpMenu);
    setJMenuBar(menuBar);
    javax.swing.GroupLayout layout =
                    new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
    layout.createParallelGroup(
                         javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(desktopPane,
         javax.swing.GroupLayout.DEFAULT_SIZE, 400, Short.MAX_VALUE)
    );
    layout.setVerticalGroup(
    layout.createParallelGroup(
                         javax.swing.GroupLayout.Alignment.LEADING)
    .addComponent(desktopPane,
         javax.swing.GroupLayout.DEFAULT_SIZE, 279, Short.MAX_VALUE)
    );
      pack();
  }// </editor-fold>
```

```
    private void exitMenuItemActionPerformed(
                                    java.awt.event.ActionEvent evt) {
      System.exit(0);
      }
    private void productListMenuItemActionPerformed(
                                     java.awt.event.ActionEvent evt)
    {
      try
      {
        MyiReportViewer myiReportViewer = new MyiReportViewer(
                    "reports/List of Products with Image.jasper");
        myiReportViewer.setBounds(
            0, 0, desktopPane.getWidth(), desktopPane.getHeight());
        myiReportViewer.setVisible(true);
        desktopPane.add(myiReportViewer);
        myiReportViewer.setSelected(true);
      }
      catch (PropertyVetoException pve)
      {
        pve.printStackTrace();
      }
    }
    private void salesDetailsMenuItemActionPerformed(
                                     java.awt.event.ActionEvent evt)
    {
      String input=JOptionPane.showInputDialog(
                                              "Enter the Sales No");
    if(input!=null)
      {
        try
          {
            int salesNo=Integer.parseInt(input);
            HashMap parameters=new HashMap();
            parameters.put("salesNo",salesNo);
            MyiReportViewer myiReportViewer = new
MyiReportViewer("reports/SalesDetails.jasper",parameters);
            myiReportViewer.setBounds(0, 0, desktopPane.getWidth(),
desktopPane.getHeight());
            myiReportViewer.setVisible(true);
            desktopPane.add(myiReportViewer);
            myiReportViewer.setSelected(true);
           }
        catch (PropertyVetoException pve)
          {
            pve.printStackTrace();
```

```
            }
          catch(NumberFormatException nfe)
            {
              JOptionPane.showMessageDialog(
                                this,"Please input numbers only");
            }
        }
      }
    /**
    * @param args the command line arguments
    */
    public static void main(String args[]) {
      java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
          new MainUI().setVisible(true);
          }
        });
      }
    // Variables declaration - do not modify
    private javax.swing.JMenuItem aboutMenuItem;
    private javax.swing.JMenuItem contentMenuItem;
    private javax.swing.JMenuItem copyMenuItem;
    private javax.swing.JMenuItem cutMenuItem;
    private javax.swing.JMenuItem deleteMenuItem;
    private javax.swing.JDesktopPane desktopPane;
    private javax.swing.JMenu editMenu;
    private javax.swing.JMenuItem exitMenuItem;
    private javax.swing.JMenu fileMenu;
    private javax.swing.JMenu helpMenu;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JMenuItem openMenuItem;
    private javax.swing.JMenuItem pasteMenuItem;
    private javax.swing.JMenuItem productListMenuItem;
    private javax.swing.JMenu reportsMenu;
    private javax.swing.JMenuItem salesDetailsMenuItem;
    private javax.swing.JMenuItem saveAsMenuItem;
    private javax.swing.JMenuItem saveMenuItem;
    // End of variables declaration
}
```

Calling a report with a parameter involves more steps. At first you have to take the required number of inputs with appropriate data type from the user. Then you have to call the `put` method of class, `HashMap`, for mapping the iReport parameter with the user input. See the following code:

```
int salesNo=Integer.parseInt(input);
HashMap parameters=new HashMap();
parameters.put("SalesNo",salesNo);
```

The first argument of the `put` method, `SalesNo` is the name of the parameter which was created in iReport, and the second argument, `salesNo`, declared here, contains the user input. If you have more than one parameter in your report, then you have to call the `put` method for each parameter.

After that, the constructor with two parameters—`String fileName`, `HashMap parameter`—is called to create the instance of `MyiReportViewer` as follows:

```
MyiReportViewer myiReportViewer = new MyiReportViewer("reports/
SalesDetails.jasper",parameters);
```

Calling reports from a web application

The concept of calling a report from a web application is similar. Instead of using the `net.sf.jasperreports.view.JRViewer`, you should generate the report on server and export the report to HTML/PDF, or some other suitable format. Finally, call the exported HTML/PDF file from the client to display the report in the browser window.

There are some overloaded methods in the class `net.sf.jasperreports.engine.JasperExportManager` to convert a Jasper file (actually `JasperPrint` object) to HTML/PDF or other format.

The following line of code is used to export the report to HTML:

```
JasperExportManager.exportReportToHtmlFile(jasperPrintObject,
destinationFileName);
```

Similarly, to export `JasperPrint` to PDF call the method:

```
JasperExportManager.exportReportToPdfFile(jasperPrintObject,
destinationFileName);
```

Recall that we created a JasperPrint[code in text] object in our viewer class by writing the following line of code:

```
JasperPrint print = JasperFillManager.fillReport( fileName, parameter,
con);
```

This `JasperPrint` object should be passed, to export the report to another format. Call the generated HTML or PDF file from the client side of your web application.

Summary

We have learned a lot in this chapter about calling a report created using iReport from a Java program.

Specifically, we have covered:

- Creating projects in NetBeans
- Creating Java classes in NetBeans
- Creating GUI in NetBeans
- Calling reports without a parameter
- Calling reports with parameters

We also discussed JasperReport API and Java classes.

Now that we've learned about creating applications in NetBeans, we're ready to create reports with iReport using NetBeans, which is the topic of the next chapter.

12
iReport in NetBeans

NetBeans is a free, opensource **Integrated Development Environment**(**IDE**) for software developers. This IDE provides many good tools that are required to create professional, desktop, enterprise, web, and mobile applications using the Java language. We can also create reports in NetBeans, if the iReport plugin is installed. So far, we have created different types of reports with the iReport standalone version In this chapter, we will learn about:

- Installing the iReport plugins in NetBeans
- Creating different types of reports inside the NetBeans IDE

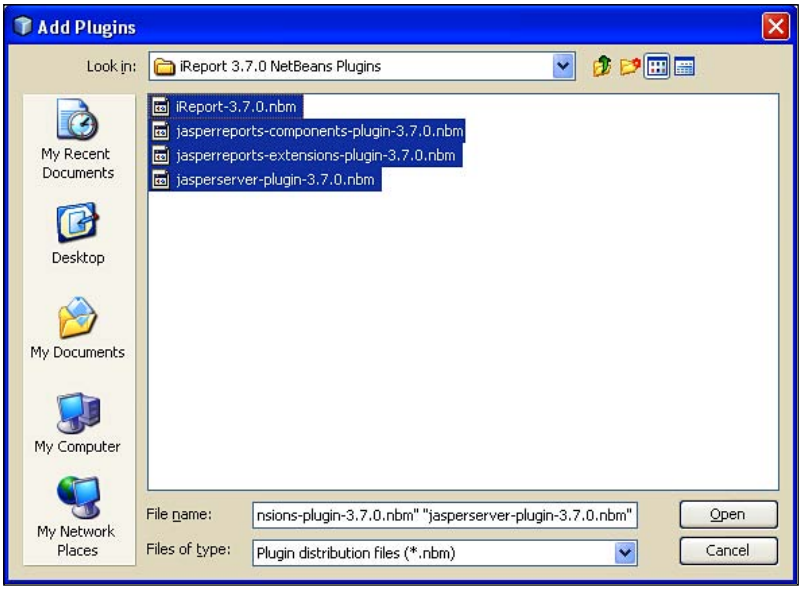
Installing iReport plugins in NetBeans

The first step is to download the NetBeans IDE and the iReport plugin for this. The iReport plugin for NetBeans is available for free download at the following locations: `https://sourceforge.net/projects/ireport/files` or `http://plugins.netbeans.org/PluginPortal/faces/PluginDetailPage.jsp?pluginid=4425`

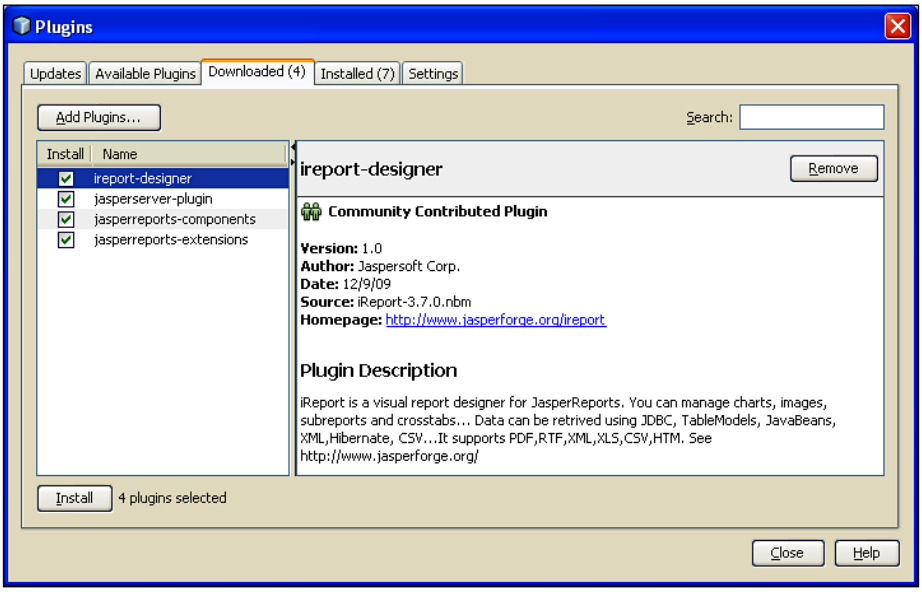
After downloading the plugin, follow the listed steps to install the plugin in NetBeans:

1. Start the NetBeans IDE.
2. Go to **Tools** | **Plugins**.
3. Select the **Downloaded** tab.

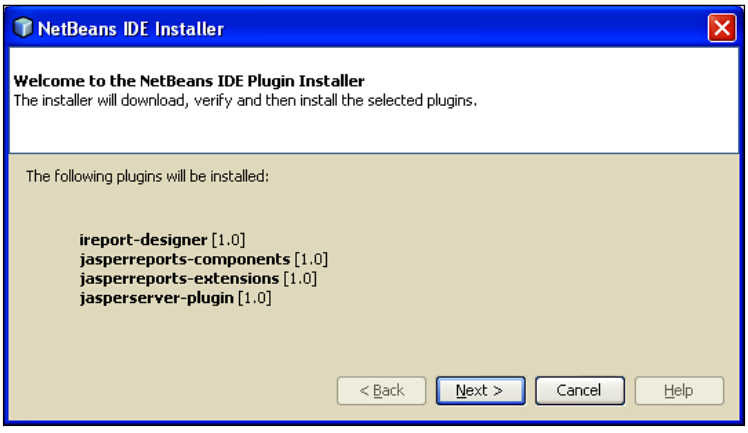
4. Press **Add Plugins...**.



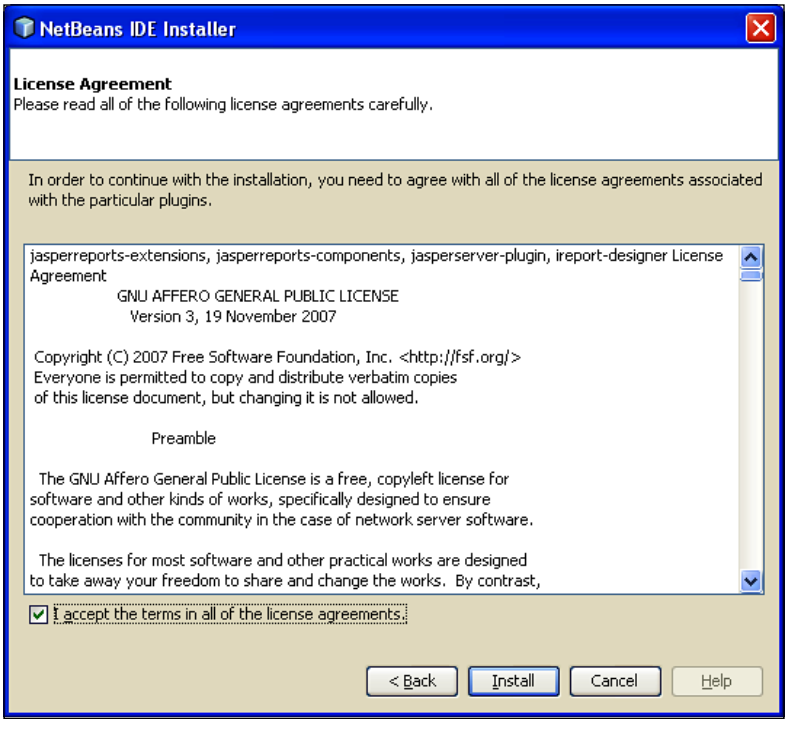
5. Select the plugin files. For iReport 3.7.0 the plugins are: `iReport-3.7.0.nbm`, `jasperreports-components-plugin-3.7.0.nbm`, `jasperreports-extensions-plugin-3.7.0.nbm`, and `jasperserver-plugin-3.7.0.nbm`. After opening the plugin files you will see the following screen:



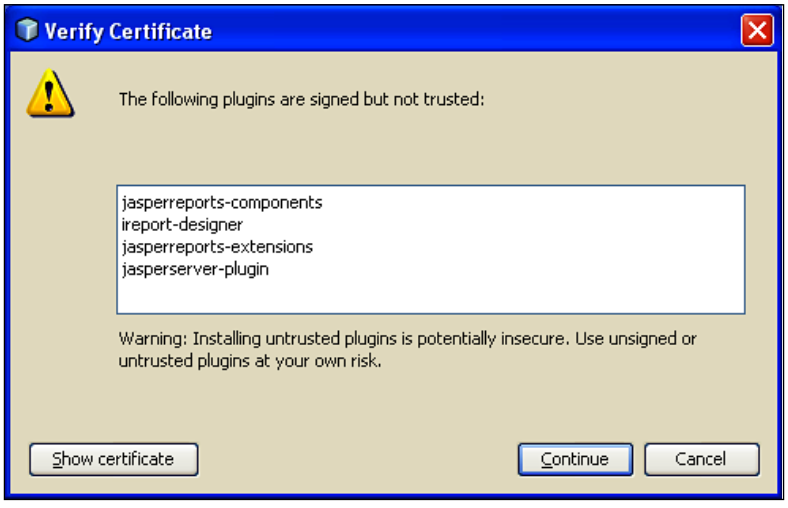
6. Check the **Install** checkbox of **ireport-designer**, and press the **Install** button at the bottom of the window. The following screen will appear:



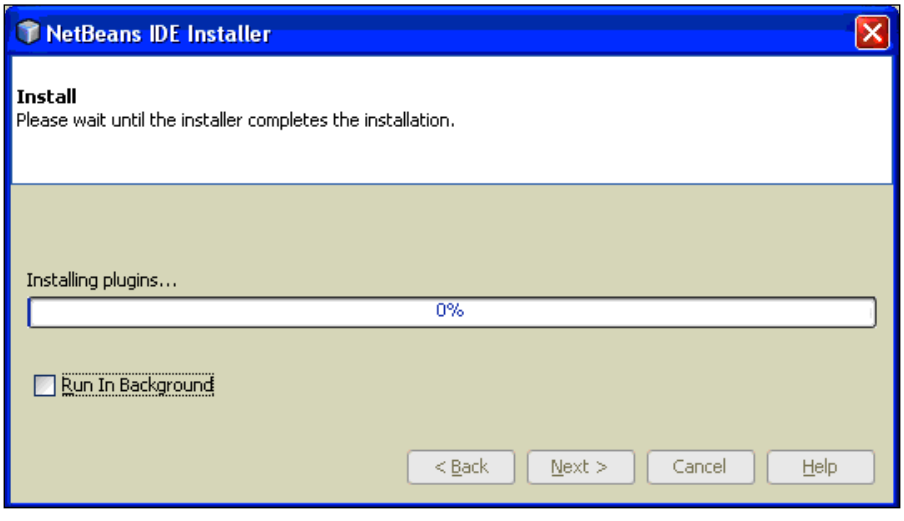
7. Press **Next >**, and accept the terms of the **License Agreement**.



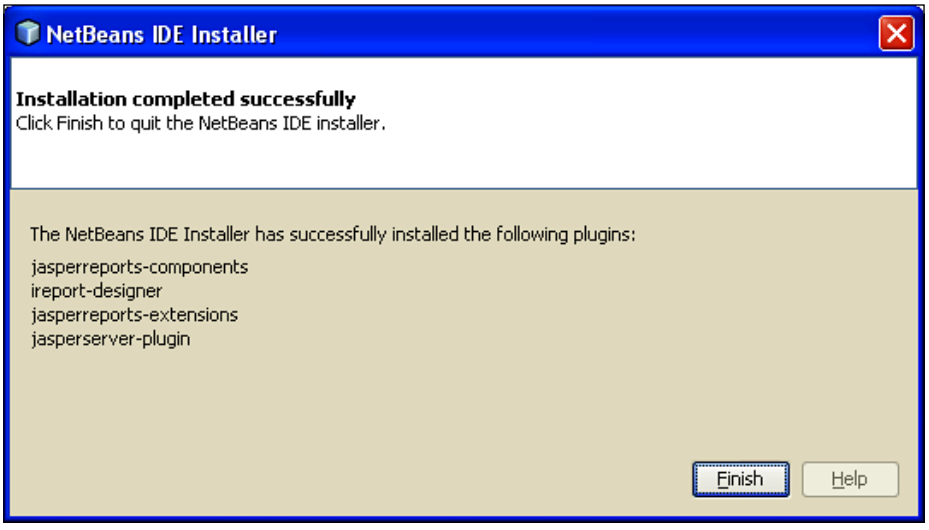
8. If the **Verify Certificate** dialog box appears, click **Continue**.



9. Press **Install**, and wait for the installer to complete the installation.



10. After the installation is done, press **Finish** and close the **Plugins** dialog. If the IDE requests for a restart, then do it. Now the IDE is ready for creating reports.



Creating reports

We have already learnt about creating various types of reports, such as reports without parameters, reports with parameters, reports with variables, subreports, crosstab reports, reports with charts and images, and so on. We have also attained knowledge associated with these types of reports. Now, we will learn quickly how to create these reports using NetBeans with the help of the installed iReport plugins.

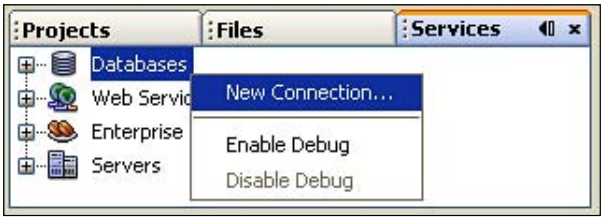
Open the NetBeans project that we created in the previous chapter, *Calling Reports from Java Applications*, and follow the listed instructions.

Creating a NetBeans database JDBC connection

The first step is to create a database connection, which will be used by the report data sources. Follow the listed steps:

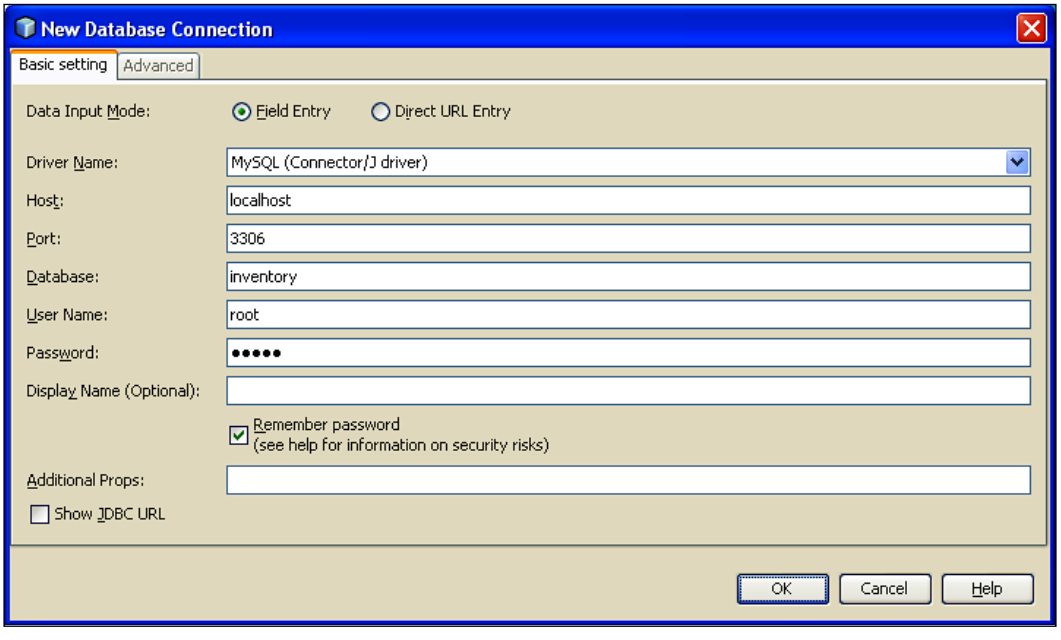
1. Select the **Services** tab from the left side of the project window.
2. Select **Databases**.

3. Right-click on **Databases**, and press **New Connection...**.

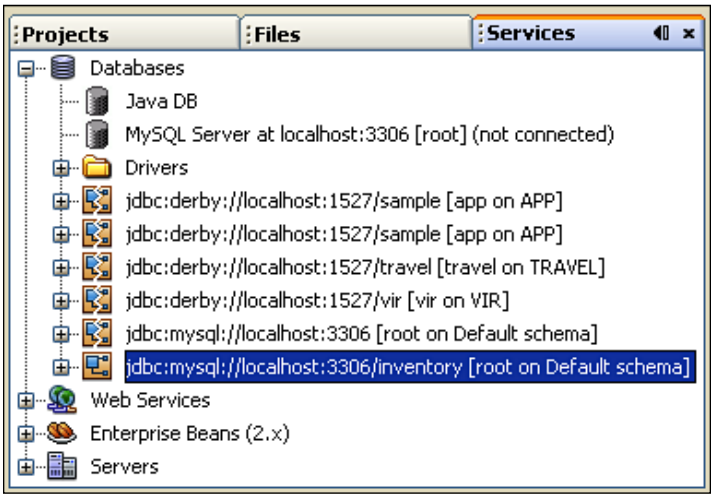


4. In the **New Database Connection** dialog, set the following under **Basic setting**, and check the **Remember password** checkbox:

Option	Value
Driver Name	**MySQL (Connector/J Driver)**
Host	**localhost**
Port	**3306**
Database	**inventory**
User Name	**root**
Password	**packt**



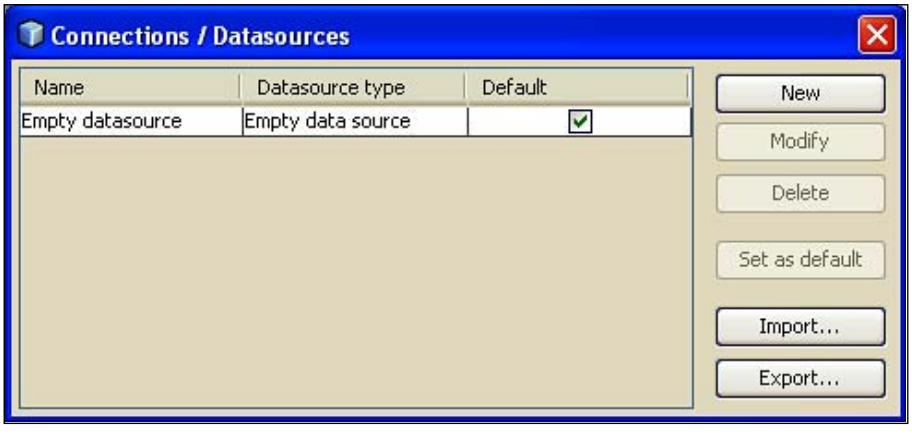
5. Press **OK**. Now the connection is created, and you can see this under the **Services** | **Databases** section, as shown in the following screenshot:



Creating a report data source

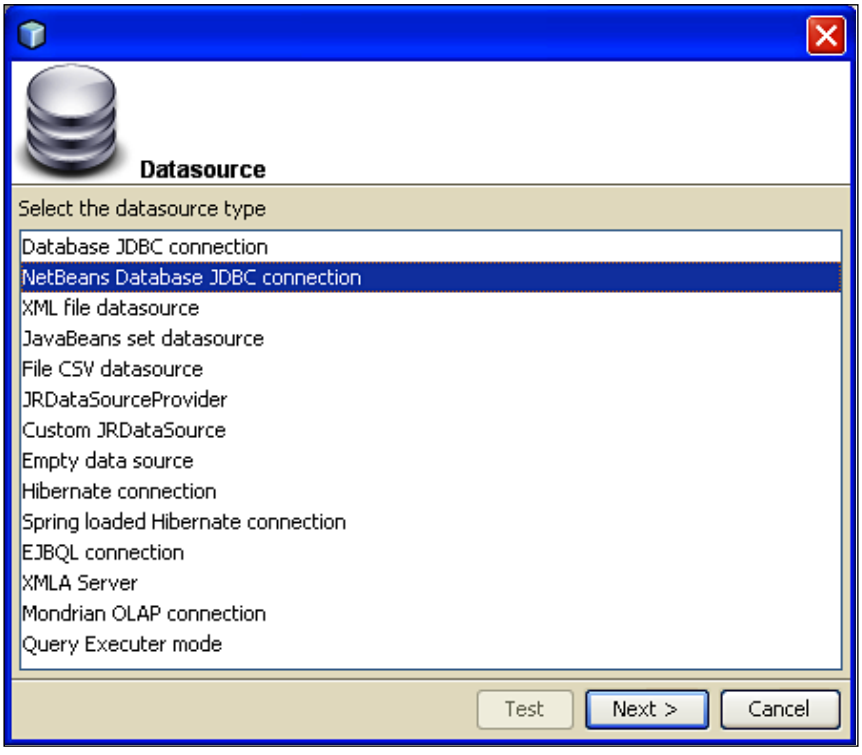
The NetBeans database JDBC connection created previously will be used by a report data source that will be used by the report. Follow the listed steps to create the data source:

1. From the NetBeans toolbar, press the **Report Datasources** button. You will see the following dialog box:

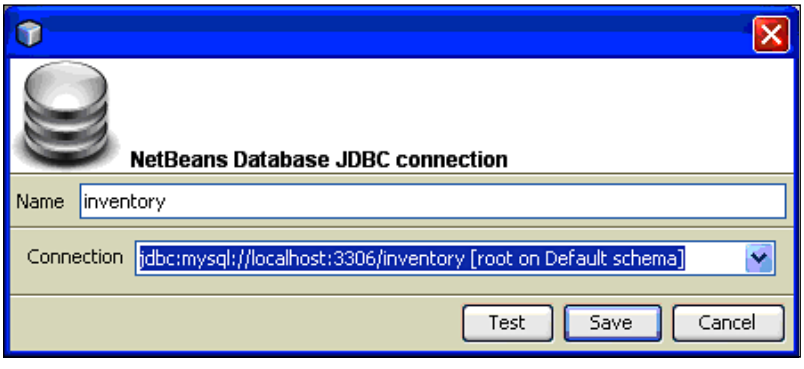


2. Press **New**.

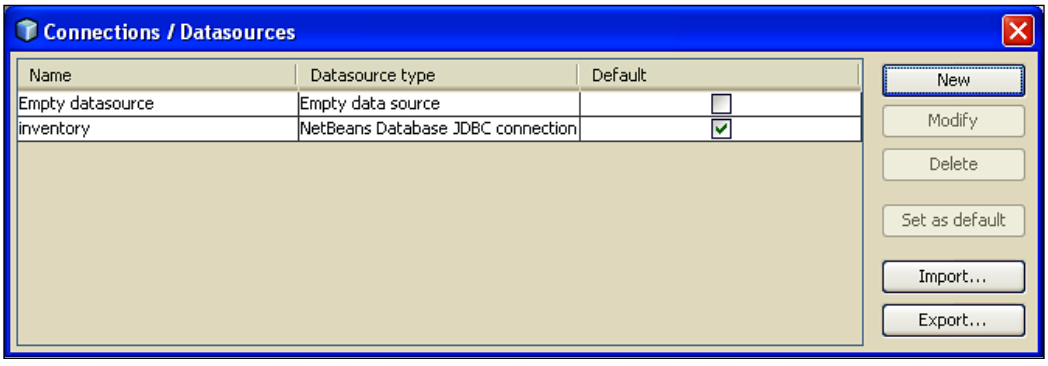
3. Select **NetBeans Database JDBC connection**, and press **Next >**.



4. Enter **inventory** in the **Name** field, and from the **Connection** drop-down list, select **jdbc:mysql://localhost:3306/inventory [root on Default schema]**.



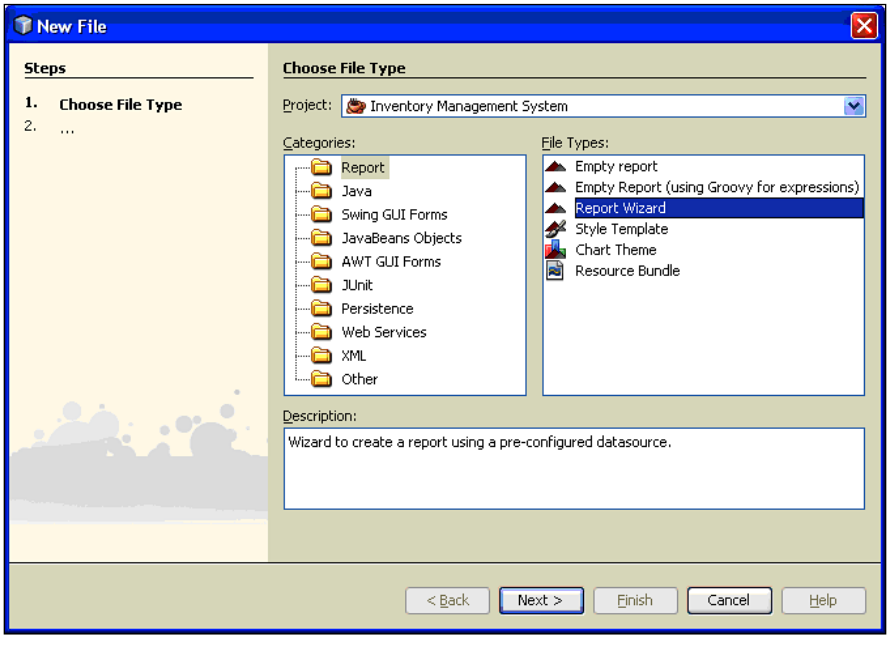
5. Press **Test**, and if the connection is successful, press **Save** and close the **Connections / Datasources** dialog box.



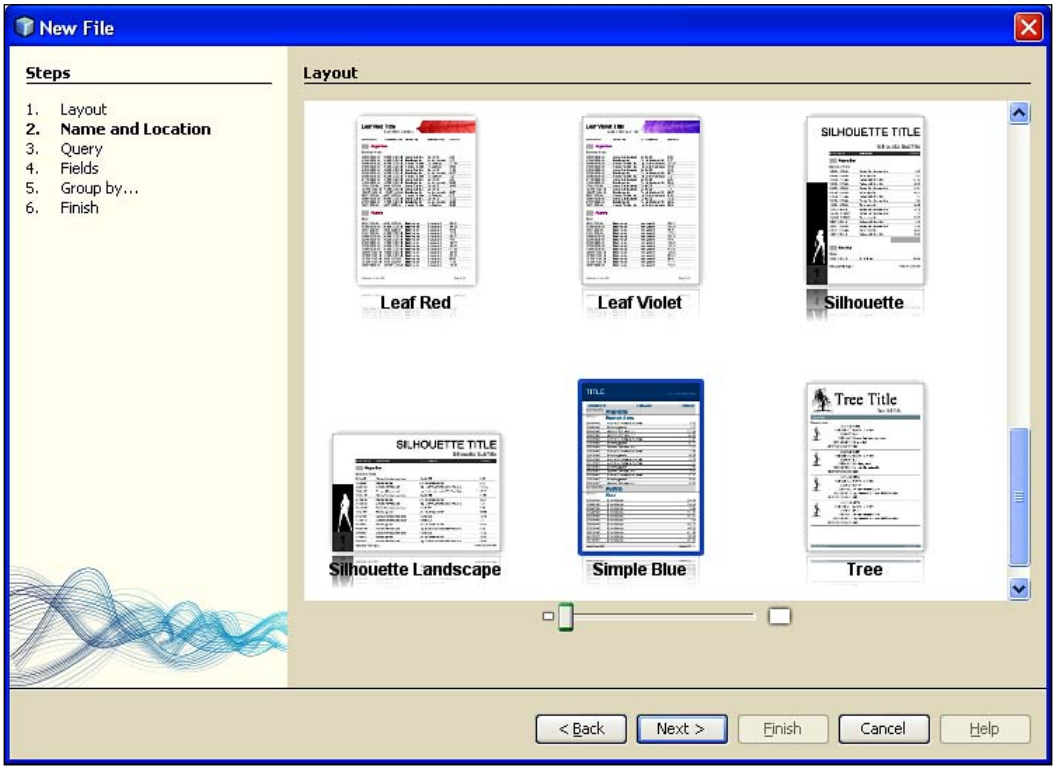
Creating a simple report

We are going to create a report, which shows the list of products. Just follow the listed steps:

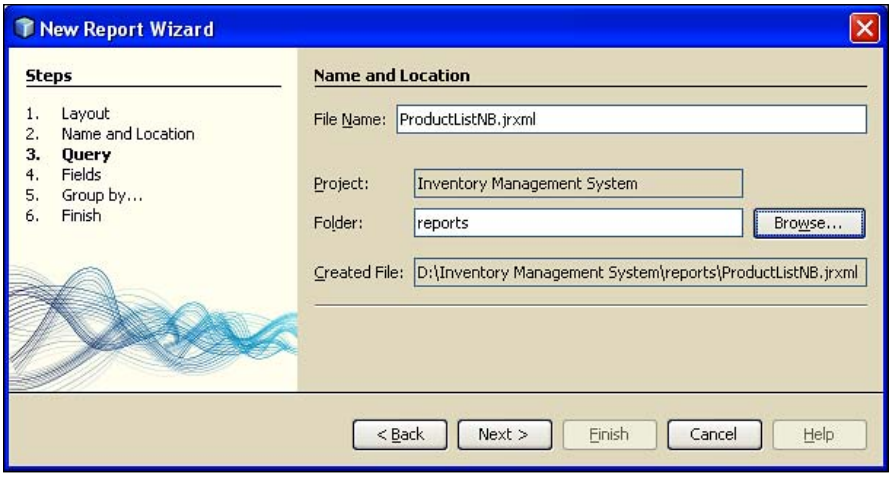
1. Go to **File** | **New File...**.
2. Select **Report** from the **Categories:** section and **Report Wizard** from the **File Types:** section, as shown in the next screenshot:



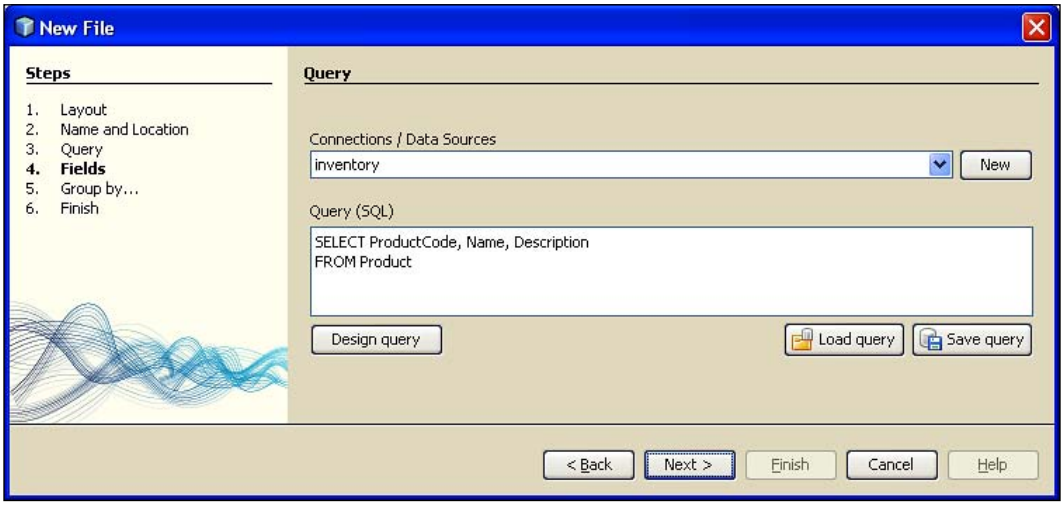
3. Press **Next >**. Select the **Simple Blue** layout and press **Next >** again.



4. Enter **ProductListNB.jrxml** as **File Name:**, and **Browse…** to the `reports` folder.



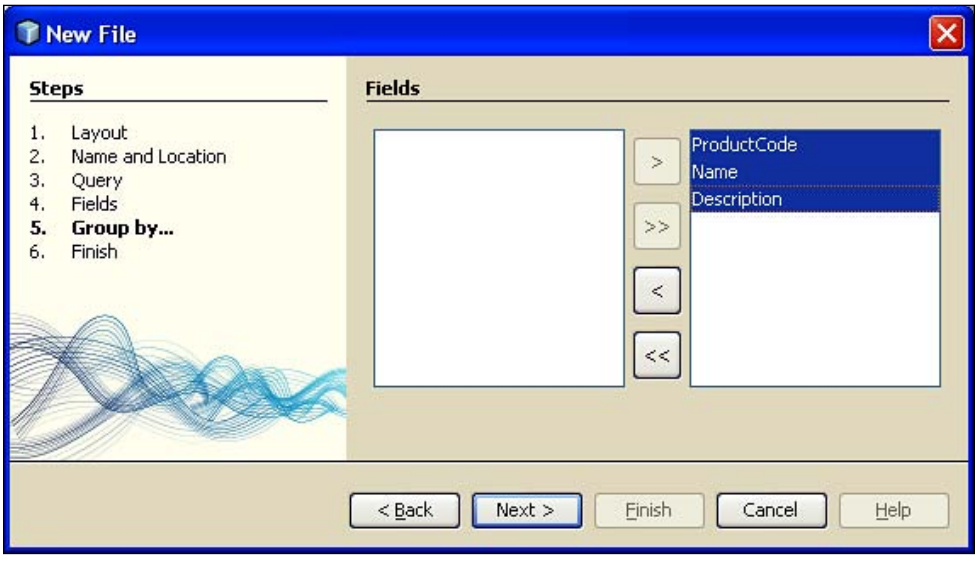
5. Press **Next>**.



6. Select **inventory** from the **Connections / Data Sources** options.
7. Write the following SQL command as the query:

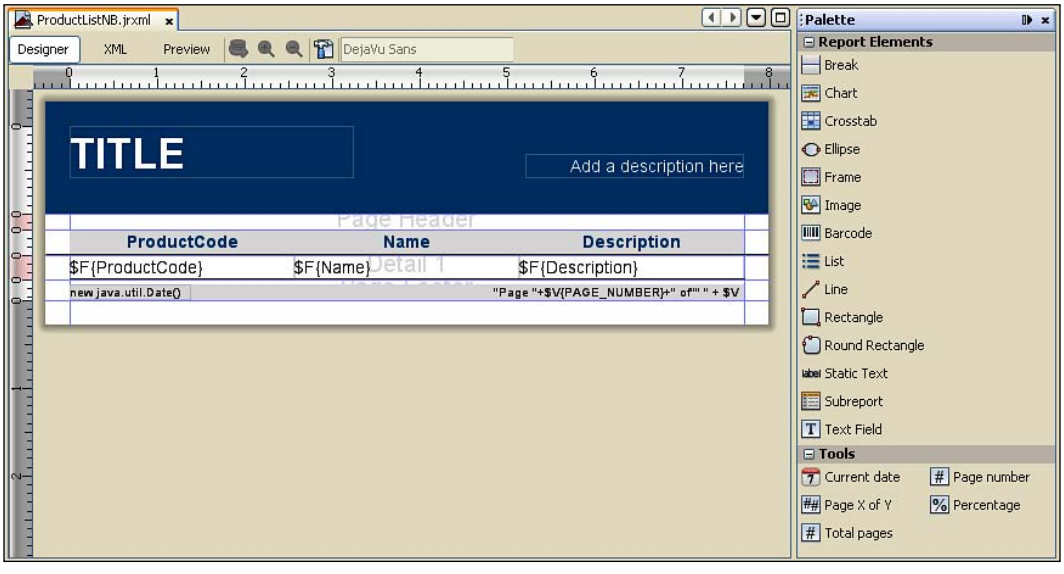
```
SELECT ProductCode, Name, Description
FROM Product
```

8. Press **Next >**. You will see the following screen:

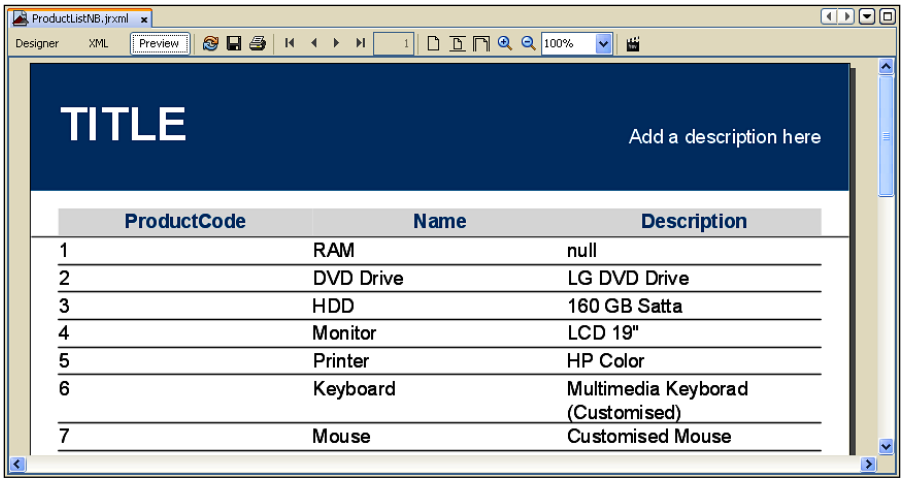


9. Select all the fields, press **>>**, and then press **Next >**.
10. Press **Next >** again without selecting any group.

11. Press **Finish**. You will see the following output:



12. Click on the **Preview** button to see the output, as shown in the following screenshot:



13. You can design the report in the **Designer** section as per your design requirements.

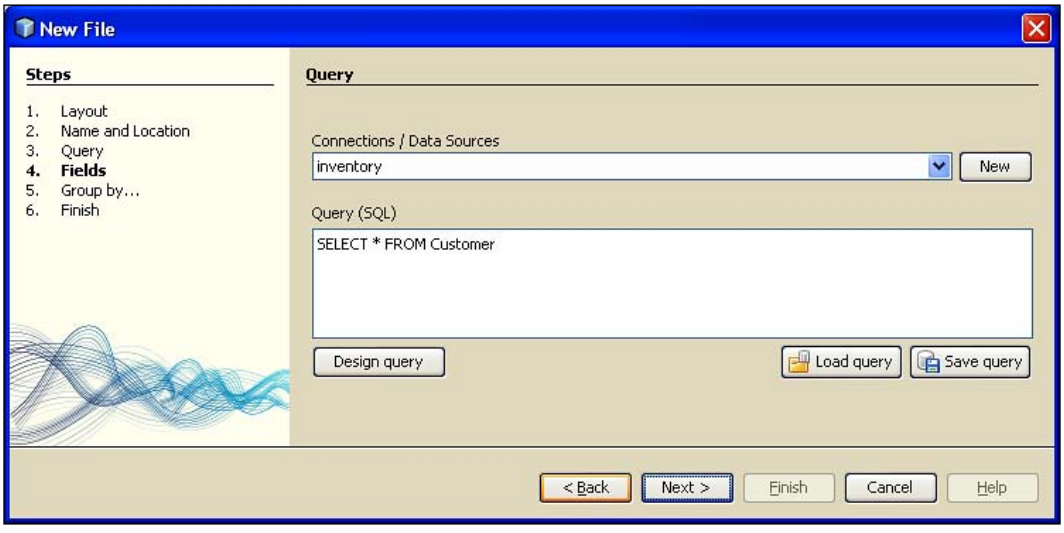
Creating a parameterized report

We are going to create a report that shows the personal information of a particular customer. You already have an idea about parameterized reports, which was covered in Chapter 5, *Using Parameters*. Here you will see how to create the same in NetBeans. Follow the listed steps:

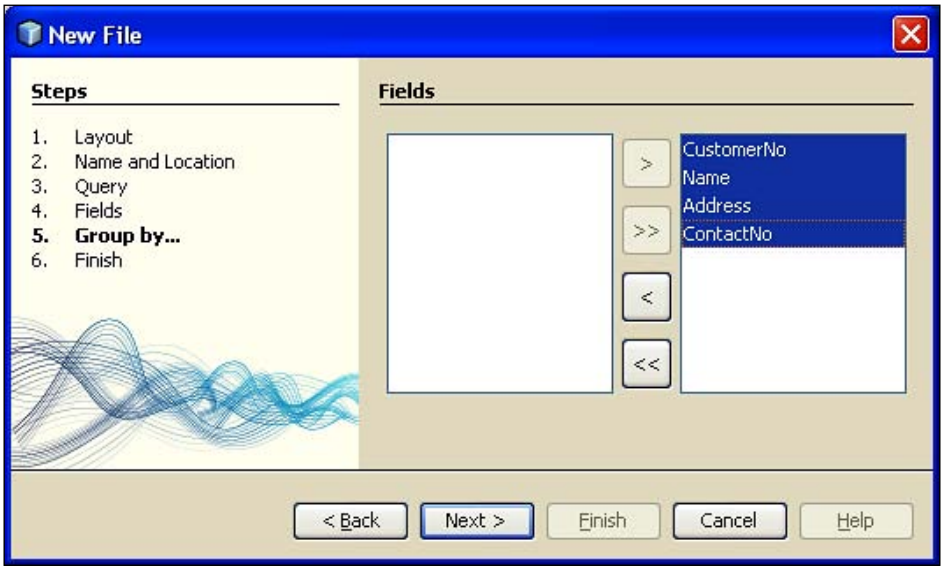
1. Go to **File** | **New File…**.
2. Select **Report** from the **Categories:** section and **Report Wizard** from the **File Types:** section.
3. Press **Next >** and select **Simple Blue** and press **Next >** again.
4. Enter **ParticularCustomerNB.jrxml** as the **File Name:**.
5. **Browse…** to the `reports` folder.
6. Press **Next >**.
7. Select **inventory** from the **Connections / Data Sources** drop-down list.

8. Write the following SQL command as the query:

```
SELECT * FROM Customer
```



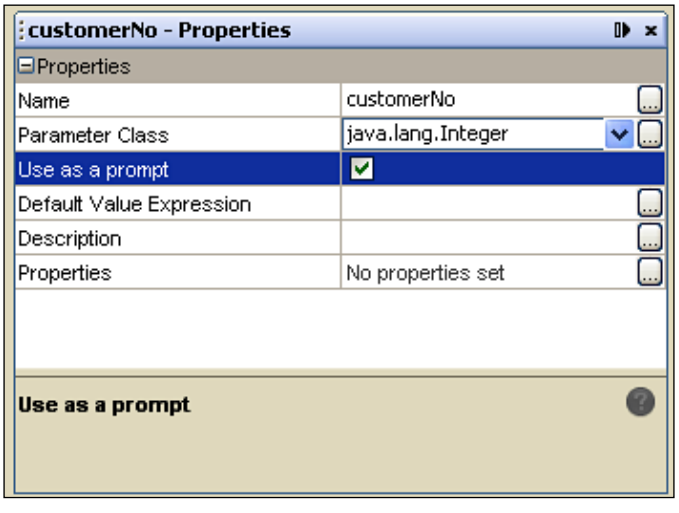
9. Press **Next >**.



10. Select all the fields, press **>>**, and then press **Next >**.

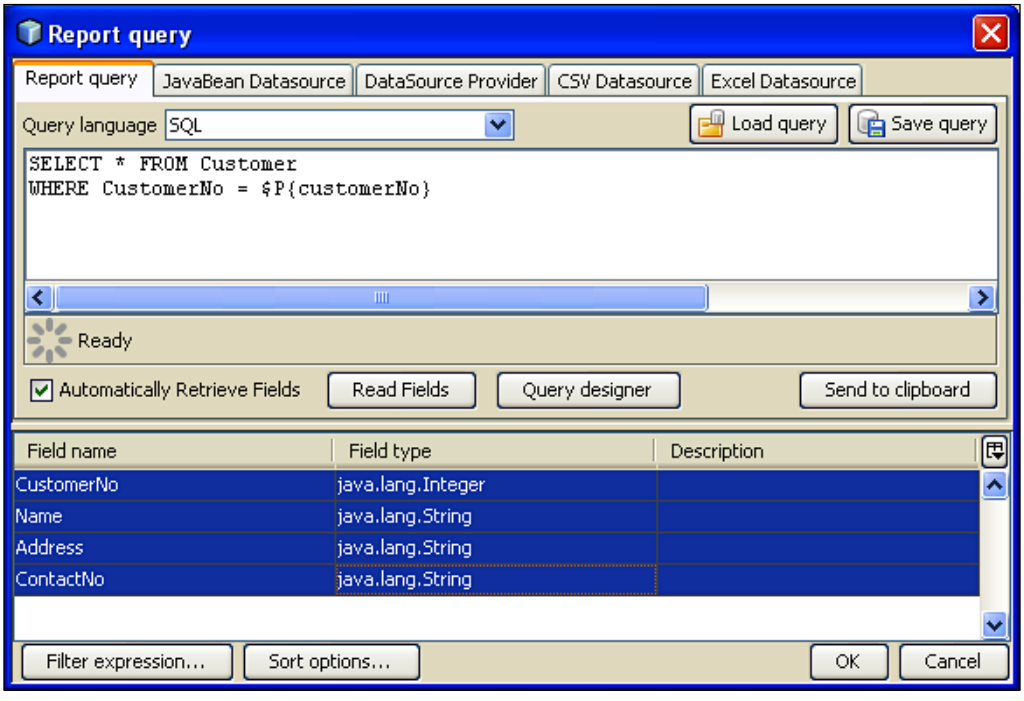
11. Press **Next >** again without selecting any group.
12. Press **Finish**. You will see the **Designer** view of the report.
13. From the **Report Inspector** (see bottom left of the designer), select **Parameters**. Right-click on it, and click on **Add Parameter**.

14. A parameter named **parameter1** is added to the **Parameters** list. Select **parameter1**, and go to the parameter **Properties** window (see bottom-right of the designer).



15. Change the **Name** to **customerNo**.
16. Change the **Parameter Class** to **java.lang.Integer**.
17. Check the **Use as a prompt** checkbox.
18. Now, click on **Report query** (beside the **Preview** button), and replace the query with the following one:

```
SELECT * FROM Customer
WHERE CustomerNo = $P{customerNo}
```



19. Press **OK** and **Preview** the report. Input the **customerNo**, and see the output.

Summary

We have seen that creating iReport reports in NetBeans makes the process of developing and managing reports easier. We have got an initial idea of creating reports in NetBeans. In the same way, we can create the more complex reports (subreports, crosstab reports, reports with charts, and so on) as well.

A Sample Database

We have used a MySQL database as the data source for our various reports. In this appendix, we will learn about:

- The design of the database
- Installing MySQL
- Configuring a MySQL server
- Creating a database and tables in MySQL using MySQL GUI tools
- Backing up and restoring database

Designing the database

Before developing the database in MySQL or any other database management system, we need to design the database properly. Database designing includes identifying the entities or tables, attributes, constraints, and the relationships among the entities. We are going to design and develop a database for monitoring the sales, purchase, and stock of products. It's an inventory management database.

List of entities

The list of entities of the database is as follows:

- `Product`
- `Supplier`
- `Customer`
- `Purchase`
- `PurchaseLine`
- `Sales`
- `SalesLine`
- `Stock`

Data dictionary

The following data dictionary provides you with the details of the database entities. The attribute name, data type, size, and the constraints (primary key, foreign key, and so on) are mentioned for each database table. This will help if you want to create the database schema on your own. Each table listed here gives us an overview of a particular single table of the database.

- **Product**: The product information will be stored in this table. Each product will be identified by a unique product code. The product image will also be stored in the database.

Attribute	Type	Size	Constraints
ProductCode	INT	11	Primary Key
Name	VARCHAR	50	Not Null
Description	VARCHAR	50	
Image	LONG BLOB		

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`product` (
  `ProductCode` int(11) NOT NULL,
  `Name` varchar(50) NOT NULL,
  `Description` varchar(50) DEFAULT NULL,
  `Image` longblob,
   PRIMARY KEY (`ProductCode`) USING BTREE
)  ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- **Supplier**: Supplier information will be stored in this table.

Attribute	Type	Size	Constraints
SupplierNo	INT	11	Primary Key
SupplierName	VARCHAR	50	Not Null
Address	VARCHAR	100	Not Null
ContactNo	VARCHAR	20	

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`supplier` (
  `SupplierNo` int(11) NOT NULL,
  `SupplierName` varchar(50) NOT NULL,
  `Address` varchar(100) NOT NULL,
  `ContactNo` varchar(20) DEFAULT NULL,
   PRIMARY KEY (`SupplierNo`)
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **Customer**: To store the information of a registered customer, this table will be used.

Attribute	Type	Size	Constraints
CustomerNo	INT	11	Primary Key
Name	VARCHAR	50	Not Null
Address	VARCHAR	100	
ContactNo	VARCHAR	20	

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`customer` (
  `CustomerNo` int(11) NOT NULL,
  `Name` varchar(50) NOT NULL,
  `Address` varchar(100) DEFAULT NULL,
  `ContactNo` varchar(20) DEFAULT NULL,
   PRIMARY KEY (`CustomerNo`)
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **Purchase**: When a purchase is made from the supplier, a record with the date and supplier number will be stored here.

Attribute	Type	Size	Constraints
PurchaseNo	INT	11	Primary Key
PurchaseDate	Date		Not Null
SupplierNo	INT	11	Foreign Key (References Supplier)

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`purchase` (
  `PurchaseNo` int(11) NOT NULL,
  `PurchaseDate` date NOT NULL,
  `SupplierNo` int(11),
   PRIMARY KEY (`PurchaseNo`) USING BTREE,
   KEY `supplierNo` (`SupplierNo`) USING BTREE,
   KEY `FK_purchase_supplierNo` (`SupplierNo`),
   CONSTRAINT `FK_purchase_supplierNo` FOREIGN KEY (`SupplierNo`)
      REFERENCES `supplier` (`SupplierNo`)
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **PurchaseLine**: A purchase may contain several `purchaseline` entries, as more than one product can be bought in a single purchase. One row will be inserted for each product bought.

Attribute	Type	Size	Constraints
PurchaseNo	`INT`	11	Part Composite Primary Key and Foreign Key (References Purchase)
ProductCode	`INT`	11	Part of Composite Primary Key and Foreign Key (References Product)
PurchaseQuantity	`INT`	11	Not Null
PurchaseUnitPrice	`Double`		Not Null

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`purchaseline` (
  `PurchaseNo` int(11) NOT NULL DEFAULT '0',
  `ProductCode` int(11) NOT NULL DEFAULT '0',
  `PurchaseQuantity` int(11) NOT NULL DEFAULT '0',
  `UnitPurchasePrice` double NOT NULL DEFAULT '0',
  PRIMARY KEY (`PurchaseNo`,`ProductCode`) USING BTREE,
  KEY `purchaseNo` (`PurchaseNo`) USING BTREE,
  KEY `productNo` (`ProductCode`) USING BTREE,
  KEY `FK_purchaseline_purchaseNo` (`PurchaseNo`),
  CONSTRAINT `FK_purchaseline_ProductCode` FOREIGN KEY
      (`ProductCode`) REFERENCES `product` (`ProductCode`),
  CONSTRAINT `FK_purchaseline_purchaseNo` FOREIGN KEY
      (`PurchaseNo`) REFERENCES `purchase` (`PurchaseNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **Sales**: When a sale is made to a customer, the (`SalesDate` and `CustomerNo`) will be recorded in this table.

Attribute	Type	Size	Constraints
SalesNo	`INT`	11	Primary Key
SalesDate	`Date`		Not Null
CustomerNo	`INT`	11	Foreign Key (References Customer)

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`sales` (
  `SalesNo` int(11) NOT NULL,
  `SalesDate` date DEFAULT NULL,
  `CustomerNo` int(11) DEFAULT '0',
  PRIMARY KEY (`SalesNo`) USING BTREE,
  KEY `customerNo` (`CustomerNo`) USING BTREE,
  KEY `FK_sales_customerNo` (`CustomerNo`),
  CONSTRAINT `FK_sales_customerNo` FOREIGN KEY (`CustomerNo`)
      REFERENCES `customer` (`CustomerNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **SalesLine**: Like the purchase, a sale may also contain several `salesline` entries, as more than one product can be sold in a single sale. One row will be inserted for each product sold.

Attribute	Type	Size	Constraints
SalesNo	`INT`	11	Part Composite Primary Key and Foreign Key (References Sales)
ProductCode	`INT`	11	Part of Composite Primary Key and Foreign Key (References Product)
SalesQuantity	`INT`	11	Not Null
SalesUnitPrice	`Double`		Not Null

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`salesline` (
  `SalesNo` int(11) NOT NULL DEFAULT '0',
  `ProductCode` int(11) NOT NULL DEFAULT '0',
  `SalesQuantity` int(11) NOT NULL DEFAULT '0',
  `UnitSalesPrice` double NOT NULL DEFAULT '0',
  PRIMARY KEY (`SalesNo`,`ProductCode`) USING BTREE,
  KEY `salesNo` (`SalesNo`) USING BTREE,
  KEY `productNo` (`ProductCode`) USING BTREE,
  KEY `FK_salesline_productCode` (`ProductCode`),
  KEY `FK_salesline_salesNo` (`SalesNo`),
  CONSTRAINT `FK_salesline_productCode` FOREIGN KEY
     (`ProductCode`) REFERENCES `product` (`ProductCode`),
  CONSTRAINT `FK_salesline_salesNo` FOREIGN KEY (`SalesNo`)
     REFERENCES `sales` (`SalesNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

- **Stock**: When a purchase or a sale is made, the `Quantity` field of `Stock` will be updated accordingly. This table will hold the current stock of the products.

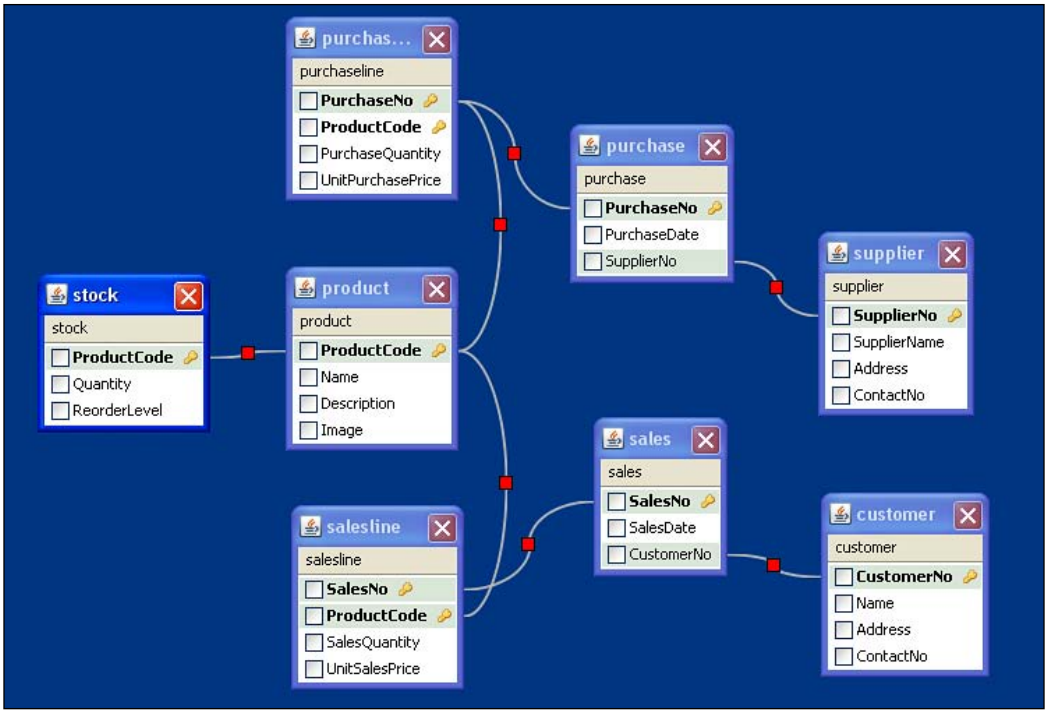
Attribute	Type	Size	Constraints
ProductCode	INT	11	Primary Key and Foreign Key (References Product)
Quantity	INT	11	Not Null
ReorderLevel	INT	11	Not Null

The statement for creating the table is as follows:

```
CREATE TABLE  `inventory`.`stock` (
  `ProductCode` int(11) NOT NULL DEFAULT '0',
  `Quantity` int(11) NOT NULL DEFAULT '0',
  `ReorderLevel` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ProductCode`) USING BTREE,
  UNIQUE KEY `productNo` (`ProductCode`) USING BTREE,
  KEY `FK_stock_productCode` (`ProductCode`),
  CONSTRAINT `FK_stock_productCode` FOREIGN KEY (`ProductCode`)
       REFERENCES `product` (`ProductCode`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
```

Entity Relationship Diagram (ERD)

An **Entity Relationship Diagram** (**ERD**) provides the degree of relationship (one to many, one to one, and so on) among entities. This diagram helps to define the primary and foreign keys appropriately. The following screenshot shows the ERD of the sample database. The ERD is produced using the **Design query** option of the Report Wizard:



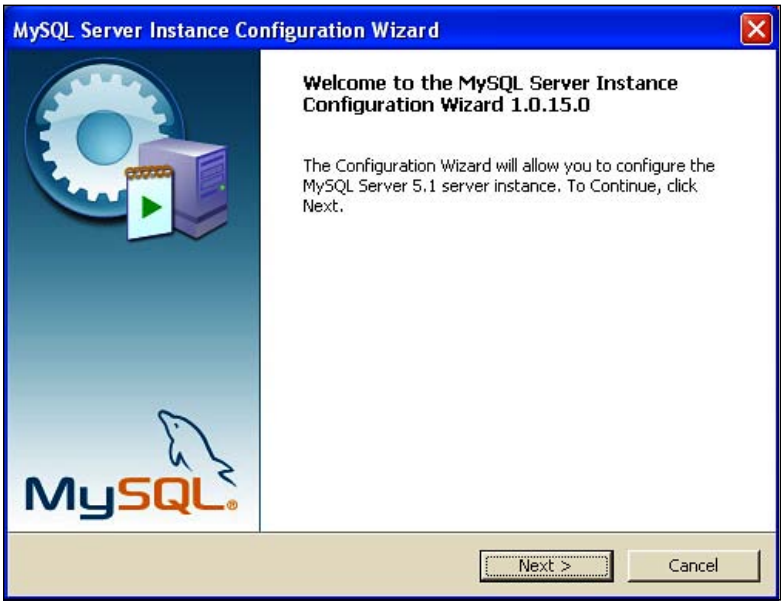
Installing MySQL and GUI tools

You can download the MySQL Community Server and MySQL GUI Tool from `http://dev.mysql.com/downloads/`. MySQL GUI Tool is a **Graphical User Interface** (**GUI**) application for administering the MySQL server and working with data. The single bundle GUI Tool includes MySQL Administrator, MYSQL Query Browser, and MySQL Migration Toolkit.

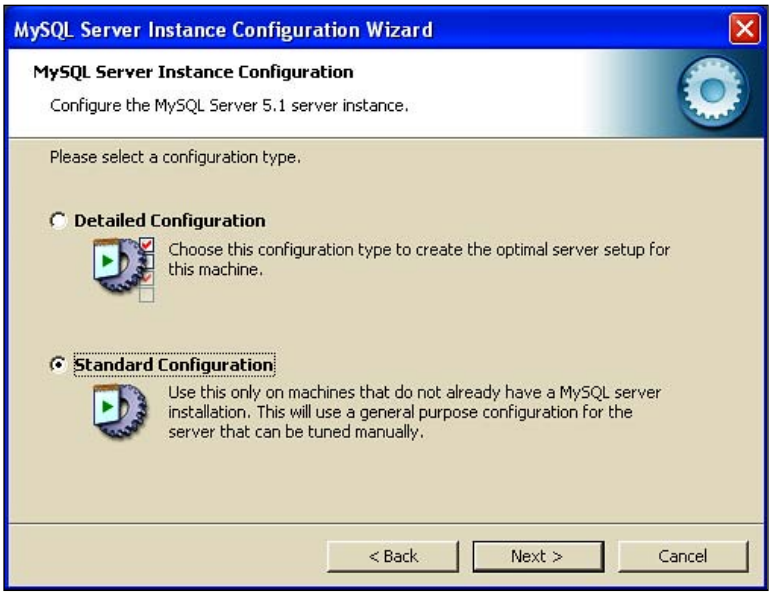
Download and install all of these and start working.

Configuring MySQL Server Instance

1. Start the **MySQL Server Instance Configuration Wizard** from your **Programs | MySQL | My SQL Server 5.0.1**.



2. Press **Next >**, and select **Standard Configuration**.



3. Press **Next >**.
4. Check the **Install As Windows Service** checkbox, enter **MySQL501** as the **Service Name:**, and check the **Launch the MySQL Server automatically** checkbox.

5. Set **packt** as the **New root password:**, and confirm the same password.

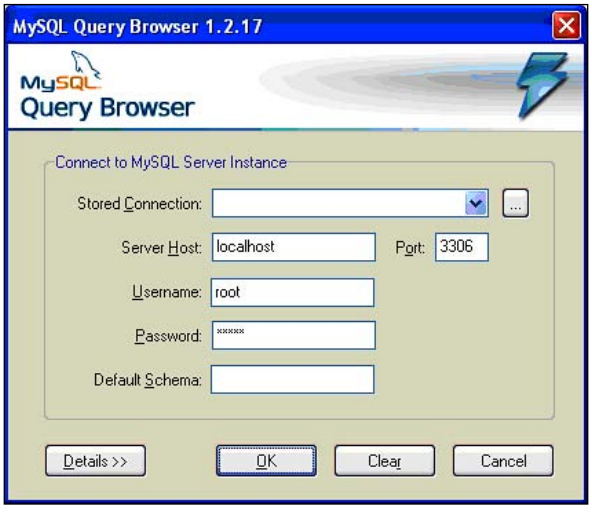
6. Press **Next >**, and click on **Execute**.

7. Press **Finish**. Now, the MySQL server is ready.

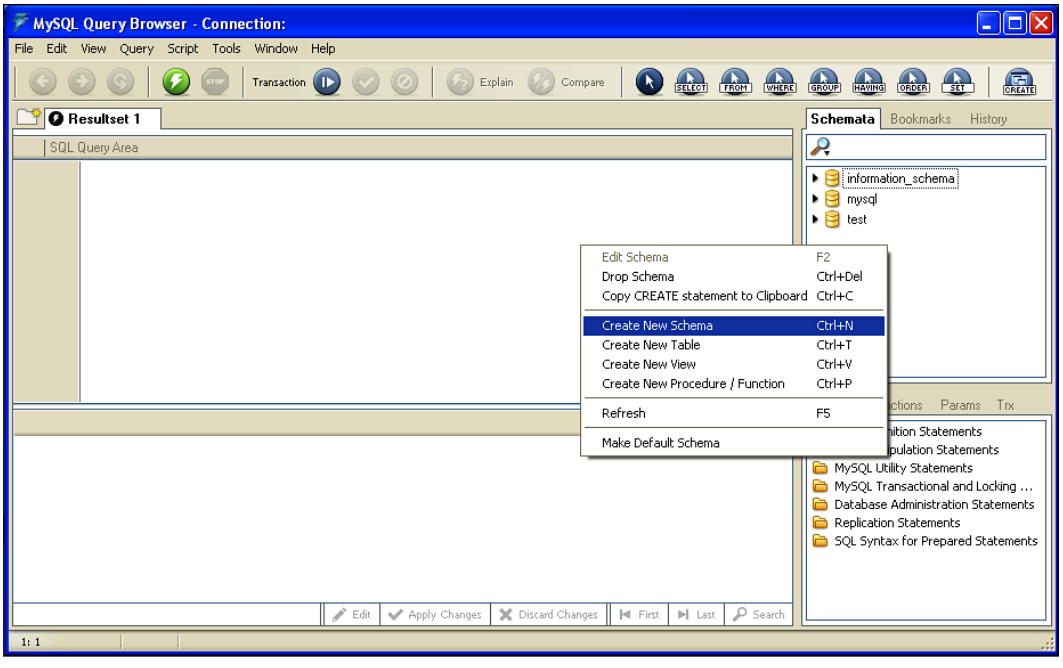
Creating a database

After installing MySQL Server, MySQL GUI Tool, and configuring the server, the first task is to create a database. We are going to create a database `inventory` according to the design shown earlier in this appendix.

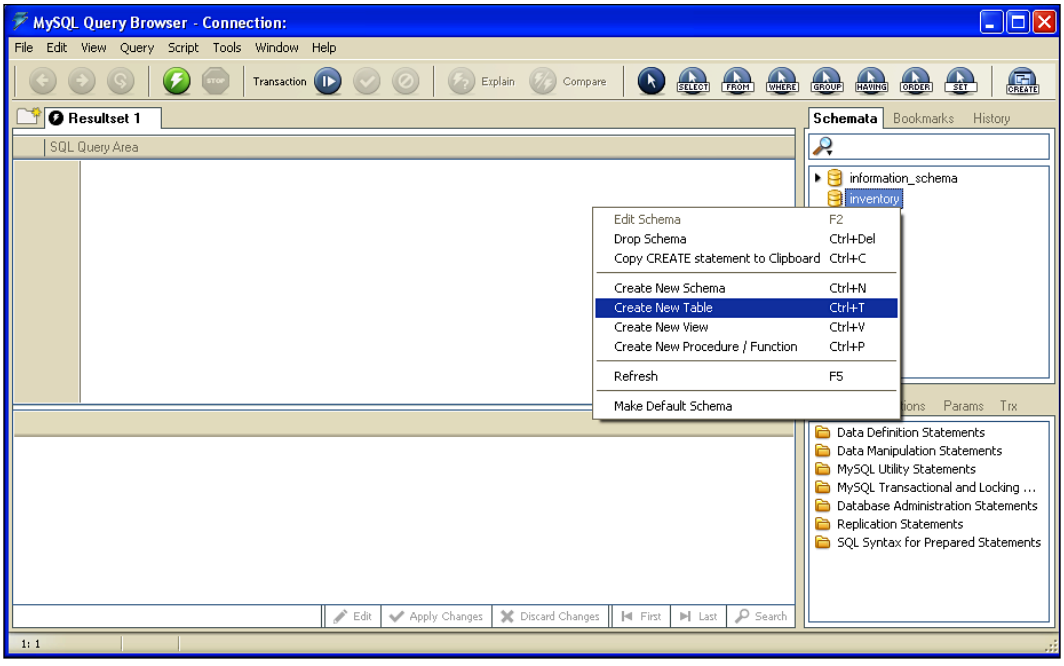
1. Start **MySQL Query Browser** from the **Programs** | **MySQL** menu.



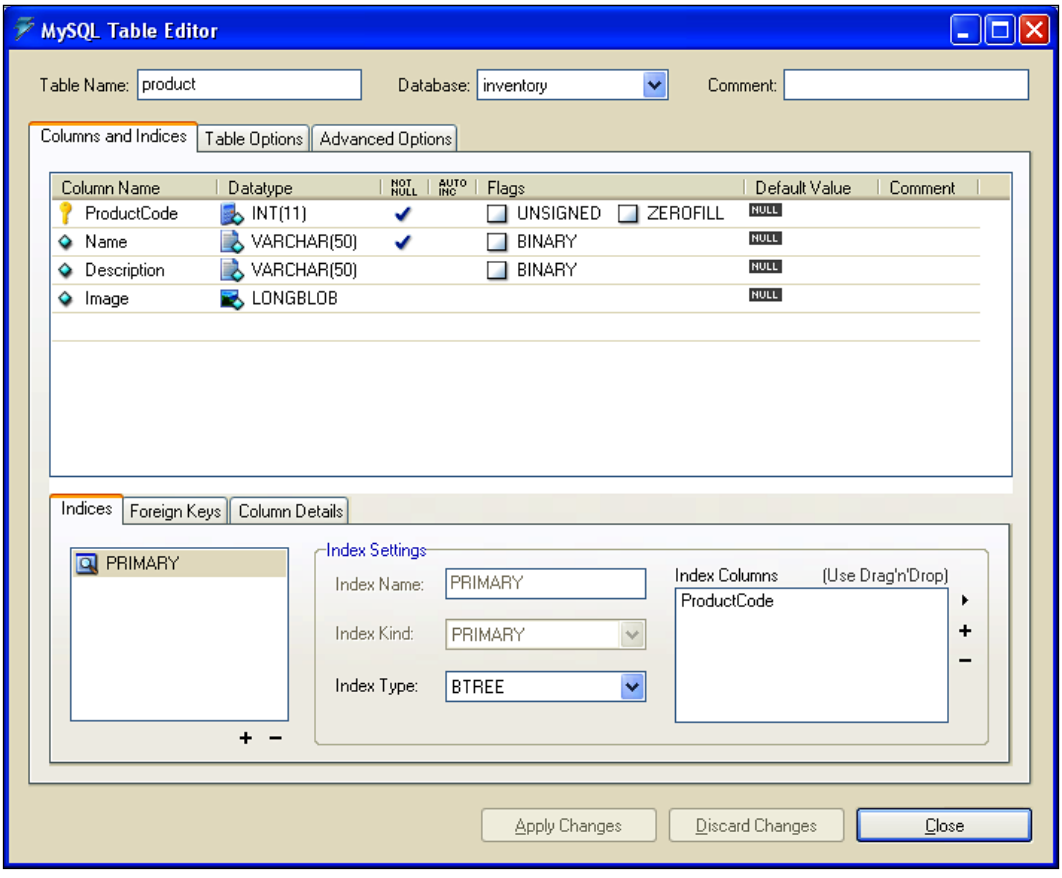
2. Input **localhost** in the **Server Host:** field.
3. Input **3306** in the **Port:** field.
4. Enter your **Username:** and **Password:** (**root** and **packt** respectively).
5. Press **OK**.



6. Right-click in the **Schemata** window, and click on **Create New Schema**.
7. Enter the **Schema Name** as **inventory**, and press **OK**.
8. Select the inventory database from the sidebar, right-click on it, and click on **Create New Table**.



9. In the **MySQL Table Editor**, input the **Table Name:** and the attributes with **Datatype** according to our design.



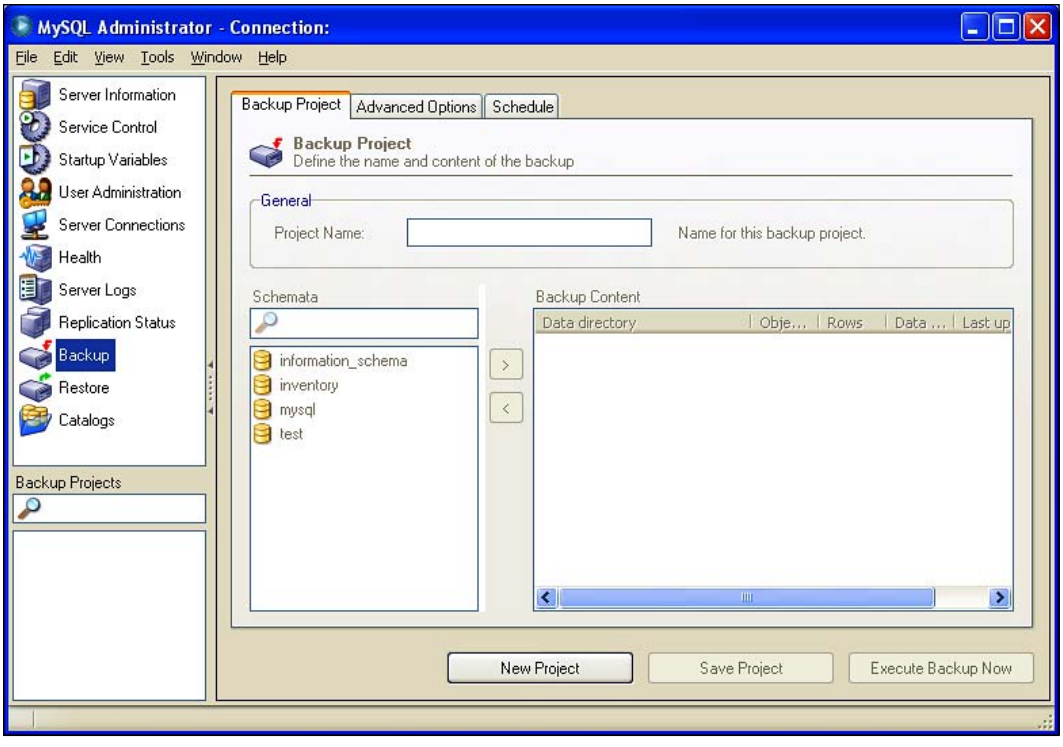
10. To set a primary key, click the icon on the left-hand side of a column name so that the key sign appears next to it. To make a column/attribute **Not Null**, click the corresponding **Not Null** row. To set a foreign key, go to the **Foreign Keys** tab, click the **+** symbol, give a name, then choose the **Ref. Table**.
11. Create all the tables in the same way.

Backing up and restoring database

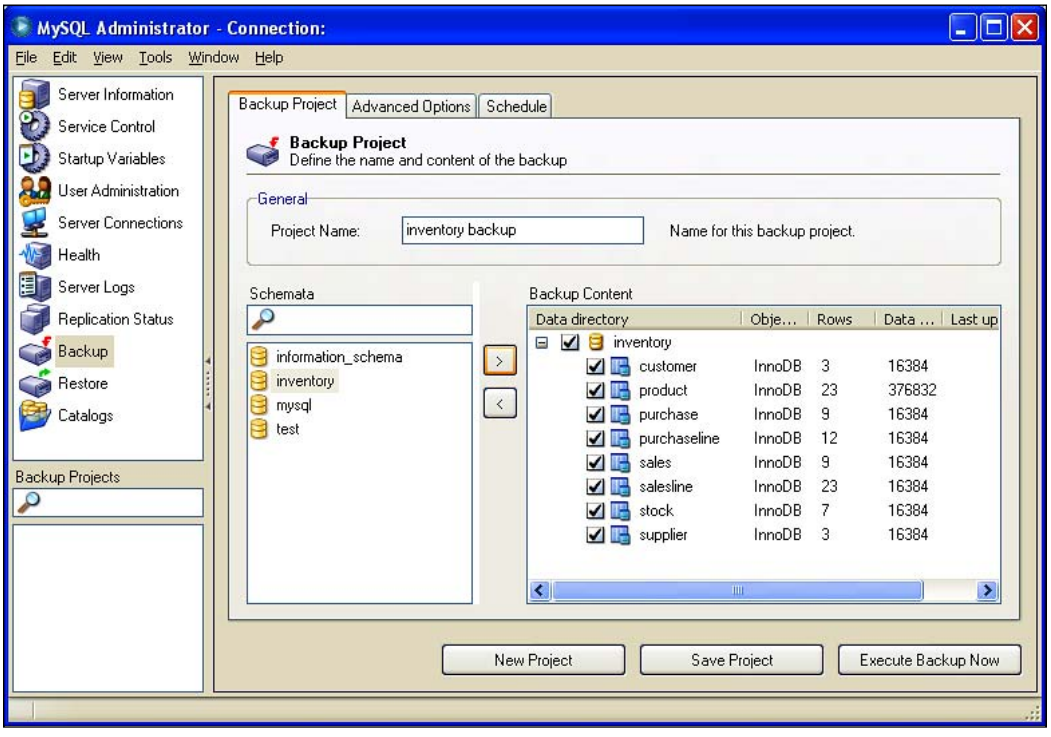
The MySQL GUI tools provide the facility to backup and restore databases. This feature helps to transfer a database from one server to another. The following are the steps to backup and restore a database using the GUI tools.

Backing up the database

1. Start **MySQL Administrator** from the **Programs | MySQL** menu.
2. Enter the **Server Host:**, **Username:**, and **Password:** as done previously, and then press **OK**.
3. Select **Backup**.



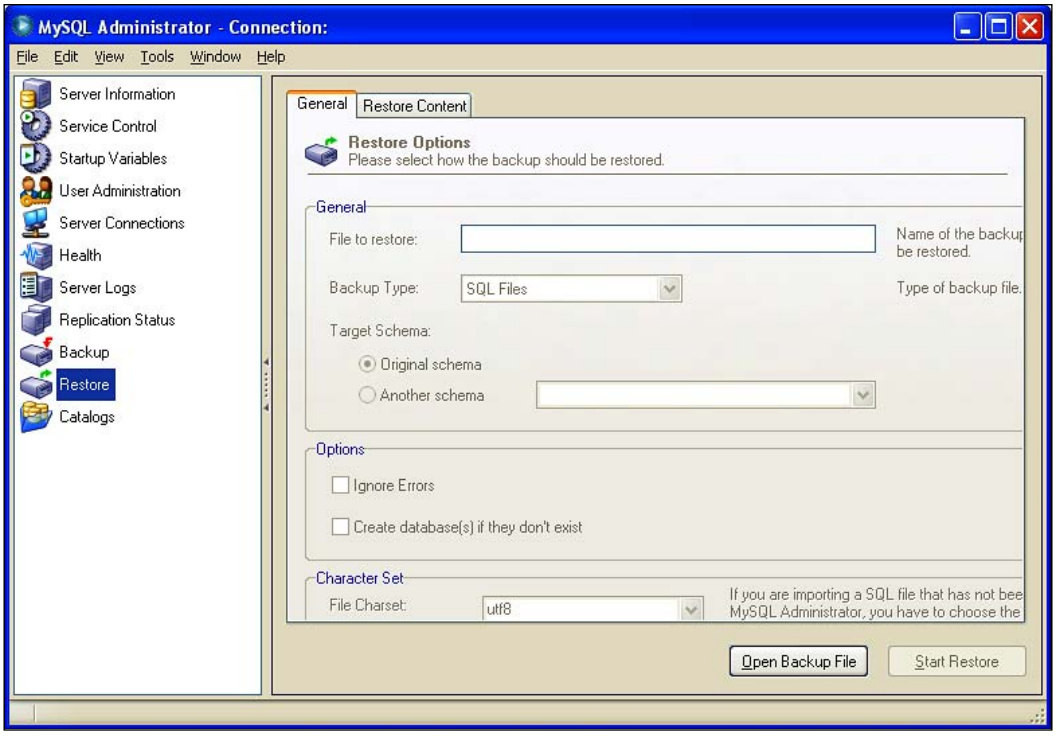
4. Press **New Project**.
5. Enter **inventory backup** as the **Project Name**.
6. Select **inventory** from the **Schemata** options.
7. Press the **>** button.



8. Press **Save Project**.
9. Press **Execute Backup Now**, select your desired directory, and then press **Save**.

Restoring the database

1. From the **MySQL Administrator**, select **Restore**.



2. Press **Open Backup File**.
3. Select your backup file, and press **Open**.
4. Press **Start Restore**.

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Thank you for buying
iReport 3.7

Packt Open Source Project Royalties

When we sell a book written on an Open Source project, we pay a royalty directly to that project. Therefore by purchasing iReport 3.7, Packt will have given some of the money received to the iReport project.

In the long term, we see ourselves and you—customers and readers of our books—as part of the Open Source ecosystem, providing sustainable revenue for the projects we publish on. Our aim at Packt is to establish publishing royalties as an essential part of the service and support a business model that sustains Open Source.

If you're working with an Open Source project that you would like us to publish on, and subsequently pay royalties to, please get in touch with us.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

About Packt Publishing

Packt, pronounced 'packed', published its first book "Mastering phpMyAdmin for Effective MySQL Management" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution-based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.PacktPub.com`.