



Quick answers to common problems

JasperReports 3.6 Development Cookbook

Over 50 recipes to create next-generation reports
using JasperReports

Bilal Siddiqui

[PACKT] open source 
PUBLISHING community experience distilled

www.it-ebooks.info

JasperReports 3.6

Development Cookbook

Over 50 recipes to create next-generation reports using JasperReports

Bilal Siddiqui

[PACKT] open source 
PUBLISHING community experience distilled

BIRMINGHAM - MUMBAI

JasperReports 3.6 Development Cookbook

Copyright © 2010 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: June 2010

Production Reference: 1220610

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

ISBN 978-1-849510-76-9

www.packtpub.com

Cover Image by Vinayak Chittar (vinayak.chittar@gmail.com)

Credits

Author

Bilal Siddiqui

Editorial Team Leader

Gagandeep Singh

Reviewers

Nadeem Ghafoor Chaudhry

Sagara Gunathunga

Lloyd H. Meinholz

Bhavani P. Polimetla

Project Team Leader

Priya Mukherji

Project Coordinator

Zainab Bagasrawala

Acquisition Editor

Dilip Venkatesh

Proofreader

Sandra Hopper

Development Editor

Mayuri Kokate

Graphics

Geetanjali Sawant

Technical Editors

Ishita Dhabalia

Vinodhan Nair

Production Coordinators

Aparna Bhagat

Arvindkumar Gupta

Indexer

Monica Ajmera Mehta

Cover Work

Aparna Bhagat

About the Author

Bilal Siddiqui is an Electronics Engineer, an XML consultant, and the founder of XML4Java.com, a company focused on simplifying e-business. After graduating in 1995 with a degree in electronics engineering from the University of Engineering and Technology, Lahore, he began designing software solutions for industrial control systems. Later, he turned to XML and used his programming experience in C++ to build web- and WAP-based XML processing tools, server-side parsing solutions, and service applications. Bilal is a technology evangelist and a frequently published technical author.

Bilal has been focusing exclusively on Java and XML-based open source tools and solutions since 2006. He has extensively used popular open source products such as JasperReports, ADempiere, Openbravo, and Eclipse. Bilal is a strong advocate of open source tools and is engaged not only in designing solutions based on open source tools but also collaborating with local universities in Lahore to train software and IT personnel in using open source technologies.

Special thanks to my wife's patience, which was often put to the test while I was working on this book. I'm also thankful for the support of Rainer Maier—my friend and business partner, who was quite convinced that I should just work on this book and sleep instead of working on our mutual projects. Finally, I am also thankful to my colleagues, Tariq and Zia, for their ideas to make this book more interesting and useful.

About the Reviewers

Nadeem Ghafoor Chaudhry received his B.S. and M.S. in Computer Science from the University of Massachusetts, Lowell, U.S.A. He worked in the software industry in the U.S. for about four years. Then he switched to academia in Pakistan and taught both undergraduate and graduate courses at different institutes of higher learning in Pakistan. Currently, he is engaged as Assistant Professor in the department of Computer Science of COMSATS Institute of Information Technology, Lahore, Pakistan.

COMSATS Institute of Information Technology (CIIT) is a renowned institute for higher learning in Pakistan. Currently, the CIIT is offering 47 different degree programs divided into 20 undergraduate and 27 graduate programs in the fields of Information and Communication Technology, Management Sciences, Electrical Engineering, Chemical Engineering, Mathematics, Physics, Bio-sciences, Development Studies, Environmental Science, Meteorology, and Architecture and Design. The present student strength is around 20,000 with faculty corpus of more than 1,600. The CIIT has already produced approximately 10,000 graduates.

Sagara Gunathunga holds a special degree in Computer Science (B.Sc.) from the University of Peradeniya, Sri Lanka and a degree in Information Technology (BIT) from the University of Colombo, Sri Lanka. He is an Apache Axis PMC member, an Apache committer for several projects that include Apache Web Service and Apache Woden. His research interest focuses on SOA, web services, distributed systems, and modularity systems.

He also has a sound industry experience in J2EE-related technologies, including ORM tools, dependency injection frameworks, and reporting. Currently, he is playing a Tech-Lead role in a U.S.-based software services company named Aeternum Inc.

Sagara also maintains his blog at <http://ssagara.blogspot.com/>, and you can get in touch with him at sagaragu@gmail.com.

All my thanks go to my mother for supporting me.

Lloyd H. Meinholz works as an IT professional in the Washington D.C. area. He has more than 20 years of experience in developing software in a variety of programming languages and production environments. Lloyd has focused on Java, Linux, and other open source technologies for the past 12 years. He has been employed by start-up companies as well as Fortune 500 companies and worked in both the public and private sectors. Lloyd has found JasperReports and iReport to be a great reporting solution for many of his employers.

Bhavani P. Polimetla has been learning and working in the IT Industry since 1990. He graduated as a Bachelor of Computer Science and Master of Computer Applications from Andhra University, India. He worked on standalone Swing applications to Grid computing and N-tire architecture. He has worked with the world's top clients including three from Fortune 50 companies. At present, he is working as an independent Java consultant in Atlanta, Georgia, U.S.A.

To demonstrate his skills, he completed 25+ certifications in the spectrum of J2EE, database, and project management subjects. He also achieved many awards for many of his projects. He reads and writes poetry in his free time. More information is available at his website www.polimetla.com.

Table of Contents

Preface	1
Chapter 1: Creating Static and Dynamic Titles and Headers	7
Introduction	7
Downloading, installing, and running JasperReports and iReport	8
Creating your first "Hello World" report	10
Creating and sizing the title for your report	15
Using dynamic titles that can change during report processing	20
Inserting a company logo in the title of your report	25
Adding a simple header to your report	28
Setting margins for your report and aligning the report header relative to report margins	34
Chapter 2: Working with the Body and Footer of your Report	41
Introduction	41
Displaying a field along with its label in the body of your report and handling null values	43
Creating a simple table of records along with labels for each column	51
Inserting a heading for a group of records	59
Using parameters to filter records during report processing	68
Implementing groups within groups — a nested hierarchy	73
Adding a simple footer to your report	81
Displaying general information or summary at the end of your report	90
Chapter 3: Enhancing the Look and Feel of your Report	99
Introduction	99
Deploying and reusing styles in your report	100
Setting background color for data	114
Using HTML tags and bullet lists	123

Expanding a field vertically to accommodate large text	135
Applying formatting pattern to the value of a data field	140
Using background images and watermarks in your report	146
Chapter 4: Working with a Variety of Data Sources	151
Introduction	151
Creating a report from relational data	152
Connecting to an XML datasource	158
Creating a report from XML data using XPath	162
Using multiple relational databases to generate a report	171
Creating a report from model beans of Java applications	184
Chapter 5: Multi-page Reports	195
Introduction	195
Building a cover page for your multi-page report	196
Creating a simple, one-page TOC for your report	201
Applying a style to your simple TOC	210
Resetting page numbering with the start of a particular record	218
Implementing complex multi-dimensional page numbering	228
Showing multiple types of data in the same report	239
Managing pagination of multiple types of data in a report	250
Chapter 6: Multi-column Reports	261
Introduction	261
Dividing the body of a report into multiple columns	261
Displaying groups of data in separate columns	266
Displaying data as name-value pairs in multiple columns	270
Filling your report horizontally in multiple columns	276
Using subreports to design a multi-column report	283
Chapter 7: Summary Report, Crosstabs, and Graphs	297
Introduction	297
Designing a simple summary report	297
Designing a multi-level summary report	303
Designing a crosstab—a table with dynamic rows and columns	315
Displaying data trends as a graph in your report	321
Embedding a bar graph inside a tabular view	331

Chapter 8: Java Wrappers for your JasperReports	343
Introduction	343
Creating a Java wrapper for your report	344
Compiling and viewing your report in a Java Swing application	351
Printing the hardcopy of your report using a Java Swing application	357
Creating an Excel report from a Java Swing application	361
Creating a JasperReport on the fly in a Java web application	366
Chapter 9: Using Mathematical and Logical Expressions	
This chapter is not present in the book but is available as a free download from: http://www.packtpub.com/sites/default/files/downloads/0769os-chapter-9.zip	
Index	375

Preface

JasperReports is the world's most popular embeddable Java open source reporting library, providing Java developers with the power to easily create rich print and web reports. While such reports are pivotal in managing business information more effectively, creating and customizing them can get tedious.

This book will give you recipes to solve common JasperReports problems to make your life easier. It will take you through complex examples related to JasperReports with step-by-step instructions on how to solve them.

The author's experience in creating reports enables him to share over 50 recipes to develop crystal-clear business reports using the capabilities of JasperReports and the amazing features provided by its visual report designer tool: iReport.

Create and enhance your business reports using the various functionalities of JasperReports and iReport.

What this book covers

Chapter 1, Creating Static and Dynamic Titles and Headers, focuses on basic things such as creating, sizing, positioning, and enhancing the titles and headers of the report.

Chapter 2, Working with the Body and Footer of your Report, covers working with the body and footer of the report, including using parameters to filter records during report processing and implementing nested hierarchy.

Chapter 3, Enhancing the Look and Feel of your Report, shows you how to enhance the look and feel of your report by deploying and reusing styles, and by using designs, textual effects, background images, and watermarks.

Chapter 4, Working with a Variety of Data Sources, focuses on advanced things such as working with a variety of data sources: relational data, XML data, model beans of Java applications, and also multiple relational databases at once.

Chapter 5, Multi-page Reports, teaches you to build a cover page and table of contents for multi-page reports, conditional reset page numbering, display multi-dimensional page numbering, design multi-page reports using sub-reports, create separate header and footer for each sub-report in your Master report, and much more.

Chapter 6, Multi-column Reports, covers topics such as dividing the body of a report in multiple columns, grouping records with common value in a report column, filling data in report columns horizontally and vertically, inserting column break in your report, controlling the size of each column independently, and using sub-reports to design multi-column reports.

Chapter 7, Summary Report, Crosstabs, and Graphs, covers designing simple and multi-level summary reports, cross-tabs with dynamic rows and columns, creating bar charts, embedding a bar inside a tabular view, and displaying data trend graphs in your reports.

Chapter 8, Java Wrappers for your JasperReports, covers generating a report from a Java Swing application, generating a report from your web application, printing a hard copy of a report on the fly from within your Java code, viewing your report as a PDF, or a web page, or an XLS sheet, and so on.

Chapter 9, Using Mathematical and Logical Expressions, covers mathematical and logical expressions such as counting the number of records with a particular field value, grouping records, applying styles to your data based on logical or mathematical conditions, and so on.



Chapter 9, *Using Mathematical and Logical Expressions*, is not present in the book but is available as a free download from: <http://www.packtpub.com/sites/default/files/downloads/0769os-chapter-9.zip>

What you need for this book

The following software products are used in this JasperReports 3.6 Development Cookbook:

- JasperReports version 3.6.0
- iReport version 3.6.0
- PostgreSQL version 8.4
- Apache Tomcat version 6.0.16
- Java Development Kit (JDK) version 1.6.0_12

Who this book is for

This book is for Java developers who want to use JasperReports to create user-friendly business reports. It is for those who may be familiar with JasperReports but want to dive into advanced JasperReports activities.

If you know how to use Microsoft Word, you can also learn how to design business reports using iReport by following the recipes of this cookbook.

Although the primary audiences of this cookbook are report designers, Java and XML developers will find this book useful in understanding how Jasper's XML code works and how to wrap JasperReports functionality in your Java Swing or web 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: Open the `Pattern.jrxml` file from the `Task5` folder in the source code for this chapter.

A block of code is set as follows:

```
<staticText>
  <reportElement x="107" y="10"
                width="340" height="33"/>
  <textElement>
    <font fontName="Verdana" size="24"/>
  </textElement>
  <text><![CDATA[Monthly Customer Invoices]]></text>
</staticText>
```

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

```
<jasperReport
  <title>
    <band height="94" splitType="Stretch">
      <!-- other JasperReports XML tags -->
      <image>
        <reportElement x="365" y="10"
                      width="180" height="80"/>
        <imageExpression class="java.lang.String">
          <![CDATA["X:\\Images\\packt.png"]]>
        </imageExpression>
      </image>
    </band>
  </title>
  <!-- other JasperReports XML tags -->
</jasperReport>
```

Any command-line input or output is written as follows:

```
uncompress iReport-nb-xx.tar.gz
```

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: "Select **Currency** from the **Category** column and click the **OK** button."



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 email 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

You can download the example code files for all Packt books you have purchased from your account at <http://www.PacktPub.com>. If you purchased this book elsewhere, you can visit <http://www.PacktPub.com/support> and register to have the files e-mailed directly to you.

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 to 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 added to any list of existing errata. 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 web site 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

Creating Static and Dynamic Titles and Headers

In this chapter, you will learn:

- ▶ Downloading, installing, and running JasperReports and iReport
- ▶ Creating your first "Hello World" report
- ▶ Creating and sizing the title for your report
- ▶ Using dynamic titles, which can change during report processing
- ▶ Inserting company logo in the title of your report
- ▶ Adding a simple header to your report
- ▶ Setting margins of your report and aligning the report header relative to report margins

Introduction

This chapter is about the titles and headers of your reports.

Imagine you are looking for a particular recipe in a cookbook. Perhaps the quickest way of doing this is to go through the table of contents, which is just a list of the titles of all the recipes contained in this cookbook. So it is important that each title says what the recipe teaches.

Similarly, the title of your report will help identify the report among other reports. The title perhaps decides the objective of designing a report.

A title normally appears on just the first page of your report. The header appears on every page. While a title helps to identify the report among other reports, a header distinguishes the report among other copies of the same report. You can understand this with an example.

Consider you are designing a report showing the data for all invoices issued for a particular customer during a specific month. "Monthly Customer Invoices" can be an appropriate title for this report. Once you have designed the monthly customer invoices report, you will periodically generate copies of this report for a specific customer and for a specific month (that is, each copy of the report will be specific for these two parameters). Therefore, it is a good idea to include these two parameters in the header of your report.

This means that a report header represents the data on which a report is generated.

Downloading, installing, and running JasperReports and iReport

iReport is a powerful visual tool for JasperReports to graphically design reports. It is much easier to generate JASPER code using the graphical features of iReport as compared to writing the JASPER code by hand.

In order to create reports using JasperReports, you will only need to download iReport from the iReport project website hosted at SourceForge.net (<http://sourceforge.net/projects/ireport>).

As the JasperReports library comes bundled with iReport, you don't need to separately download and install JasperReports.

This recipe just shows you how to download, install, and run iReport.

How to do it...

1. Visit the iReport project website at <http://sourceforge.net/projects/ireport>. The iReport project homepage at SourceForge.net will open.
2. Click on the **View all files** button, and a list of all available versions of **iReport** will appear. I used version 3.6.0 of iReport, which is the latest at the time of writing these recipes. Therefore, expand **iReport-3.6.0**, by clicking on it. It will show all the files related to **iReport-3.6.0**. Download the file named `iReport-nb-3.6.0-windows-installer.exe`. This is an executable file.
3. Double-click the Windows executable file. A setup wizard will run, which will show you the installation steps and ask you for the folder where you want to install iReport. Just provide the destination folder and the installation will finish.

4. After the installation is finished, iReport will run and show a welcome window. Click on **File** and select **New** from the **File** menu. A **New file** dialog will appear containing several icons such as **Report**, **Style**, **Chart Theme**, and so on. Each icon represents a type of file you can create. The **Report** icon comes pre-selected, which means iReport allows you to create reports by default.
5. Now your iReport is up and running and you are ready to try the recipes of this cookbook.

There's more...

The five steps given above show you how to download and install iReport for Microsoft Windows. iReport also supports other operating systems such as Linux, Mac OS X, and Solaris. The following is a short guideline to download and install iReport for these operating systems.

Download and install iReport for Linux, Mac OS X, and Solaris

You can download iReport for Linux, Mac OS X, or Solaris by following the steps listed below:

1. Visit the sourceforge.net website at <http://sourceforge.net/projects/ireport>. The iReport project homepage at SourceForge.net will open.
2. Click the **View all files** button, to the right of the **Download Now!** button. A page will open containing the latest downloads for all operating systems in the **Newest Files** section of the download page. You will find that separate downloads of iReport are available for Linux and Solaris as `tar.gz` files and for Mac OS X as DMG files. Download iReport for your operating system.
3. For Linux and Solaris, you just need to unzip the `iReport-nb-XX.tar.gz` files to your required folder. If you are using Linux, open the Linux Terminal and type the following command in the terminal:

```
tar -zxvf iReport-nb-xx.tar.gz
```



Visit <http://www.linuxhelp.net/newbies/#unzip> to learn more about unzipping `tar.gz` files.

If you are using Solaris, you will extract the package files with the `uncompress` command, as shown below:

```
uncompress iReport-nb-xx.tar.gz
```

4. After you finish extracting the archive, run the `ireport.exe` file from the `iReport/bin` folder.

Download older versions of iReport for Windows, Linux, Mac OS X, or Solaris

If you need to download an older version of iReport, click on the **View all files** button to the right of the **Download Now!** button on the iReport project home page. A page containing a list of available versions will open. You will find the latest downloads for all operating systems in the **Newest Files** section and older versions in the **All Files** section of the download page.

Note that I have used iReport version 3.6.0 in this cookbook. iReport versions dated before 2008 are referred to as iReport classic, which might behave differently from the recipes of this cookbook.

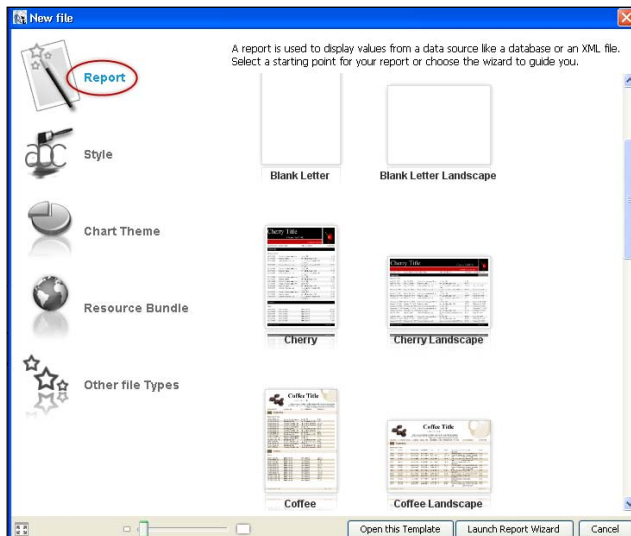
Creating your first "Hello World" report

This recipe is a quick starter for someone who is using iReport for the first time. It shows you what views of your report the tool offers and what type of report components are available. This recipe also teaches you how to quickly make a simple "Hello World" report.

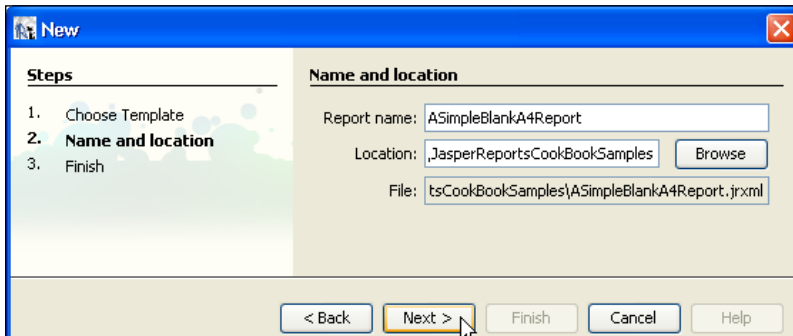
How to do it...

The following ten steps will show you how to make your first "Hello World" report.

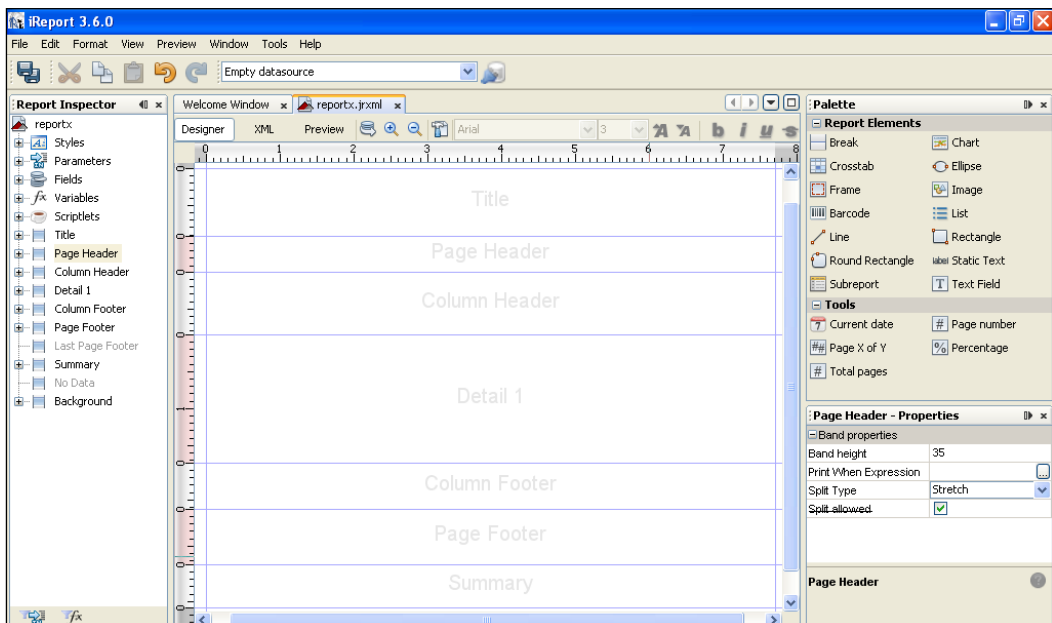
1. Run iReport. Click on **File** and select **New** from the **File** menu. A **New file** dialog will appear, which contains several icons such as **Report**, **Style**, **Chart Theme**, and so on. Each icon represents a type of file you can create. The **Report** icon comes pre-selected, which means iReport allows you to create reports by default.
2. Click on the **Report** icon on the top-left corner of the **New file** dialog. You will see a number of report templates available to start building your report.



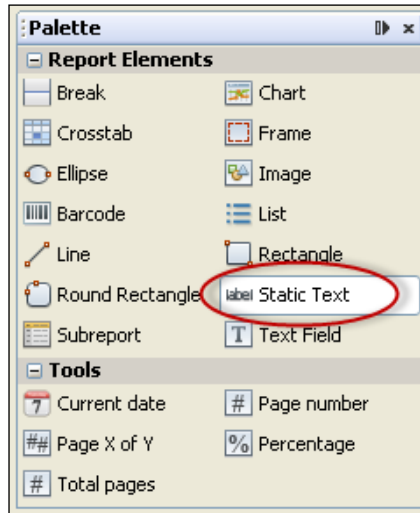
3. Select the **Blank A4** report template and click the **Open this Template** button at the bottom. **Blank A4** is the simplest of all templates and, therefore, is a good starting point to learn iReport.
4. Next, you will be prompted to enter a filename for your report. Specify a name of your choice. I call my first report **ASimpleBlankA4Report**, as shown next. Click the **Next** button.



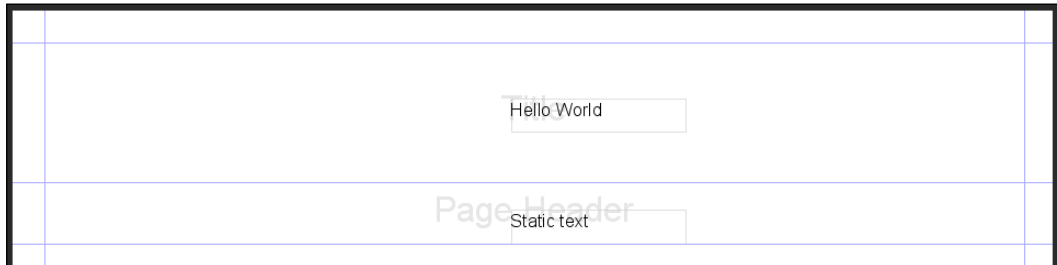
5. The **Finish** dialog will appear saying that you have successfully created your new report. Click on **Finish** to dismiss it.
6. iReport will open your blank report in its default view, which shows three tabs: **Designer**, **XML**, and **Preview**. The **Designer** tab shows the various sections of your report such as **Title**, **Page Header**, **Column Header**, and so on, as shown in the following screenshot:



- You will see a **Palette** of alphabetically arranged components (for example, a break, a chart, a static text, or a text field component) on the right of the main iReport window, as shown next. Drag a **Static Text** component from the **Palette** and drop it into the **Title** section of your report. Similarly drag-and-drop another **Static Text** component into the **Page Header** section of your report.

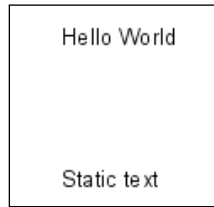


- Double-click on the **Static Text** component of the **Title** section and type **Hello World** in it as shown:



- Click on the **XML** tab, which shows the **JasperReports XML (JRXML)** code that iReport authors in response to your designer actions in the **Designer** tab. You will see that XML code for JasperReports changes every time you drag-and-drop a component from the palette into the designer view of your report.

10. Finally, switch to the **Preview** tab, which shows what your "Hello World" report will actually look like:



How it works...

JasperReports has defined its own XML-based markup language called JasperReports XML, or JRXML for short. JRXML code contains all the information that is required to specify everything about your report. Therefore, designing a report using JasperReports means you have to perform the following steps:

1. Write JRXML code manually according to the requirements of your report.
2. Parse and compile the JRXML code to generate a JASPER file for your report.
3. Combine your JASPER files with application data to generate your report view. This process is called filling.
4. Export your report view to a popular format such as PDF or XLS.

You normally need to write Java code to compile, fill, and export your JasperReport.

iReport is a JRXML authoring, compiling, filling, and exporting tool. This means that if you can author JRXML manually and write Java code to perform these tasks (compiling, filling, and exporting), you don't need iReport.

However, it is really troublesome trying to write JRXML manually. The graphical tools and components in iReport make it very easy and quick to author the JRXML according to the design requirements of your reports. iReport performs the following tasks for you to help you generate your report:

1. As a visual authoring tool, it helps you to generate JRXML by just dragging-and-dropping of components. It actually translates your visual actions to JRXML code.
2. It compiles the JRXML to generate a JASPER report file.
3. It also combines JASPER files with application data to generate a print preview for your screen and printer.
4. Then it may export this report preview to a popular format such as PDF or XLS.

iReport provides three integrated views to make your work easier: the designer view (where you design your report graphically), the preview view (where you can see the effects of your design), and the XML view (where you can view and edit the actual XML of a JRXML file).

To see the XML view of iReport, switch to the **XML** tab and you will find the following JRXML code for the **Blank A4** report:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <background><band splitType="Stretch"/></background>
  <title>
    <band height="79" splitType="Stretch"/>
  </title>
  <pageHeader>
    <band height="35" splitType="Stretch"/>
  </pageHeader>
  <columnHeader>
    <band height="61" splitType="Stretch"/>
  </columnHeader>
  <detail>
    <band height="125" splitType="Stretch"/>
  </detail>
  <columnFooter>
    <band height="45" splitType="Stretch"/>
  </columnFooter>
  <pageFooter>
    <band height="54" splitType="Stretch"/>
  </pageFooter>
  <summary>
    <band height="42" splitType="Stretch"/>
  </summary>
</jasperReport>
```

You can see from this code that there is a JRXML tag for every section of your report. For example, there is a `<title>` tag which wraps the title of your report. Similarly, you can see a `<pageHeader>` tag, which wraps the header.

Each JRXML tag for a section has a `<band>` child, which specifies two things about the section: the height of the section through a `height` attribute and the splitting behavior of the section.

In steps 8, 9, and 10 of this recipe, you played a little with the **Designer**, **XML**, and **Preview** tabs, respectively. You will need these tabs in almost every report you prepare using iReport.

The **Designer** tab is perhaps where you will work most of the time. It shows a window in which you will design your report by using different components such as static text, text field, and so on. These components are available in a palette of components on the right of your main iReport window.

In step 7 of this recipe, you played with the palette of components to quickly have an idea of how components are used in your report. The **Palette** window is split into two sections named **Report Elements** and **Tools**. The former contains several graphical components (for example, **Static Text**, **Text Field**, **Frame**, and so on), whereas the latter contains tools such as **Current date** and **Page number**. You will learn how to use these tools in Chapter 2, *Working with the Body and Footer of your Report*.

Creating and sizing the title for your report

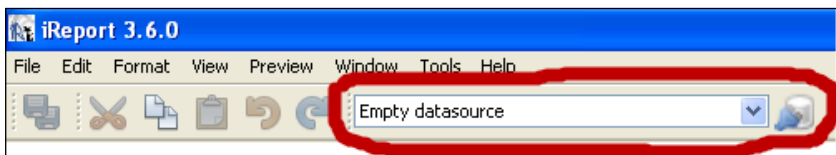
A self-explanatory title for your report helps identify the report among other reports. A popular style of designing self-explanatory titles is to have a main title and a subtitle. The main title is normally brief and the subtitle is a more elaborate form of the main title.

In this recipe, you will learn how to provide a main title and a subtitle for your business report. We are going to create, size, position, and color the title.

Getting ready

There is no database connection required in this recipe.

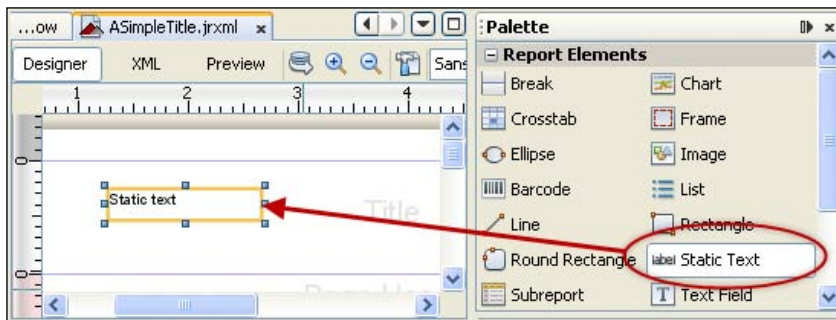
If you have not made any database connection so far in your iReport installation, you will see an **Empty datasource** selected in a drop-down list just below the main iReport menu. This is the requirement to run this recipe. If any other option (for example, some database connection) is previously selected, then change it to **Empty datasource** from the drop-down list, as follows:



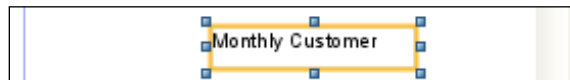
How to do it...

the following steps are very simple, and demonstrate how you can design a basic title for your report:

1. Open the `ASimpleTitle.jrxml` file from the `Task3` folder of the source code for this chapter. The **Designer** tab of iReport will show a blank report in the **Designer** tab of its main window. You can see various sections of your report. The first section at the top of the report is **Title**, which is a placeholder for the title of your report.
2. Now you will insert a title into your report. A palette of components is available on the right of the main iReport window. Select the **Static Text** component from the palette and drag it into the **Title** section in the **Designer** tab.

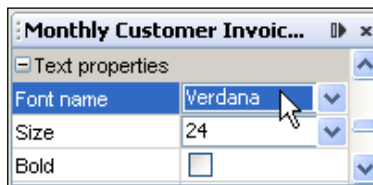


3. Double-click and edit the **Static Text** component to provide a title for your report (for example "Monthly Customer Invoices").

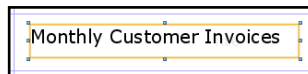


4. You will see that some of the title text will disappear from view. This is because your title is bigger than the standard rectangular boundary of the static text component, which contains the title. So, enlarge the rectangle boundary of this component by dragging one of its corners.
5. Now you will give your title a prominent look by increasing its font size. Select the title; the properties of the title field will appear in the **Properties** window just below the palette of components. Find the **Size** property among the **Text properties** and set it to **24**.

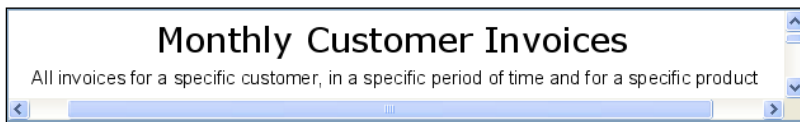
6. Similarly, find the **Font** property and set it to **Verdana**, as shown next:



7. Your report title will get a bold prominent look, as shown here:



8. You can see from the previous image that the title is not well positioned within the **Title** section. In order to position the title, right-click on the main title, and a pop-up menu will appear; select **Position**, and another pop up will appear; select **Center Horizontally**. This will center your title horizontally in the **Title** section.
9. Next, you insert a subtitle for your report by dragging another static text component from the palette into the **Title** section of your report and repeating steps 2, 3, and 4. I have provided a self-explanatory subtitle "All invoices for a specific customer, in a specific period of time and for a specific product."
10. Now, repeat step 5 for the subtitle and choose **13** as the font size.
11. Switch to the **Preview** tab, which shows what your report looks like.



12. You have successfully designed the title of your report. Save it by selecting **Save** from the **File** menu.

How it works...

Switch to the **XML** tab and you will see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other JasperReports XML tags -->
  <title>
    <band height="79" splitType="Stretch">
      <staticText>
```



```

        <reportElement x="107" y="10"
            width="340" height="33"/>
        <textElement>
            <font fontName="Verdana" size="24"/>
        </textElement>
        <text><![CDATA[Monthly Customer Invoices]]></text>
    </staticText>
    <staticText>
        <reportElement x="24" y="48"
            width="512" height="21"/>
        <textElement>
            <font size="13"/>
        </textElement>
        <text>
            <![CDATA[All invoices for a specific customer,
                in a specific period of time and for a
                specific product]]>
        </text>
    </staticText>
</band>
</title>
<!-- other JasperReports XML tags -->
</jasperReport>

```

I have shown only the `<title>` tag in the code and omitted other JRXML tags that are not relevant to this discussion.

The `<title>` tag is a wrapper for both of the static text components you dropped into the **Title** section of your report. Each static text component is represented by a `<staticText>` child tag of the `<title>` tag. The first child of the `<staticText>` tag is named `<reportElement>`. A `<reportElement>` tag describes the position (using `x` and `y` attributes) and dimensions (using `height` and `width` attributes) of the static text component.

The `x` and `y` attributes control the position of a component. Both of these represent the positioning of the component with respect to the top-left corner of the **Title** section. You can play around by changing the value of the `x` and `y` attributes of the first `<reportElement>` tag to 0 in the JRXML code. The **Designer** tab will show that the main title of your report will move to the top-left corner of the **Title** section.

When you centered your main title horizontally in step 8 of this recipe, iReport calculated the values of these `x` and `y` attributes for you. This just shows the usefulness of iReport as a graphical report editor.

The `height` and `width` attributes control the size of a component. You can also play with these attributes to change the size of your title and subtitle.

The second child, `<textElement>`, of the `<staticText>` tag contains a `` tag, which carries the text formatting information (for example, the name and size of font). You will notice that the `` tag in the second `<textElement>` does not have the `fontName` attribute. That is because you did not change the `font` property of your subtitle. This means that the subtitle uses the default font, which is `SansSerif`.

You can see that the title (Monthly Customer Invoices) and subtitle (All invoices for a specific customer, in a specific period of time and for a specific product) of your report are wrapped inside two `<text>` tags, which go inside `<staticText>` tags. This means that whenever you drop a static text into the **Title** section of your report, iReport simply inserts a `<staticText>` child tag in the `<title>` tag of your report. Later, when you type the actual title, iReport simply wraps it in the `<text>` child tag of the `<staticText>` tag.

You can try editing the title directly from your JRXML. You will see that manual edits in the JRXML are reflected in the designer view of your report.

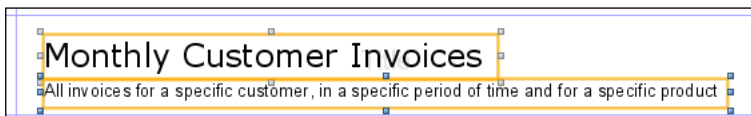
There's more...

You can also play with the `x`, `y`, `height`, and `width` attributes from the **Designer** tab by changing the properties of your main title and subtitle. The `x` attribute appears as the `Left` property shown in the **Properties** window just below the palette of components. Similarly, the `y` attribute appears as the `Top` property. The `height` and `width` attributes appear as properties with the same name.

Aligning multiple components simultaneously

iReport also allows you to align two or more JasperReports components with respect to each other in your report. For example, you may like aligning the subtitle parallel to the main title. In order to do this, you simply follow the following steps:

1. Open the `ASimpleTitle_Final.jrxml` file from the `Task2` folder of the source code for this cookbook. The **Designer** tab of the iReport shows that the report contains a title and a subtitle.
2. Hold down the `Ctrl` key of your keyboard. First, click on the subtitle, which is a reference for you to set alignment of the target component (that is the main title).
3. Secondly, click on the main title and release the `Ctrl` key. You will see that now both components are viewed as selected.
4. Right-click on any of the components, and a pop-up menu will appear; select **Align**, and another pop up will appear; select **Align Left**. This will align the main title with the subtitle, as shown next:





Sometimes, it is difficult to drag to a specific width. It is better to enter the width in the XML View. For example, it is difficult to drag the width from 58 to 60. Instead of struggling with the mouse, it is better to enter 60.

Using dynamic titles that can change during report processing

A report title can be static or dynamic. A static title is fixed at design time, that is while designing the report. It remains unchanged and, therefore, every time you generate the report, it will have the same title.

An example of a fixed title would be "Monthly Customer Invoices" for a report that shows all invoices for a specific customer in a month.

In many cases, you may need every copy of your report to have some copy-specific words in its title. For example, if you are generating a copy of the "Monthly Customer Invoices" report, you may like to display the name of the customer in the title. This will make your title dynamic.

In this recipe, I will show you how to generate dynamic titles, which have some words that are not fixed while you are designing the report. JasperReports will decide these copy-specific words every time you generate a copy of your report.

Getting ready

In order to work with this recipe, you need to have a relational database installed on your computer. I prefer a simple and easy-to-use open source database, which makes a small footprint on your PC. That's because I am providing sample data with this recipe, so I would like to make it as easy and quick as possible for readers to import the sample data into their databases before executing the steps of this recipe.

I have used the PostgreSQL database, which fulfills these criteria. You can visit the web site (<http://www.postgresql.org/>) and get all the information to install and start using PostgreSQL. You may also refer to the `installPostgreSQL.txt` file included in the source code download for this chapter, which shows how you will install and run PostgreSQL. Make sure that PostgreSQL is up and running before you proceed.

The source code for this chapter also includes a `copySampleDataIntoPGS.txt` file, which will help you create a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data for all the recipes of this chapter into the table.

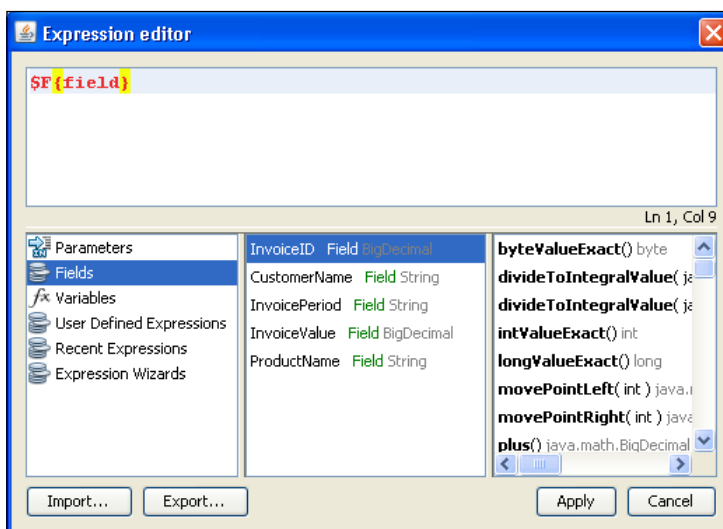
After inserting the required data, you will connect iReport to your database hosted on PostgreSQL using the steps described in the Creating a report from relational data recipe later in Chapter 4.

How to do it...

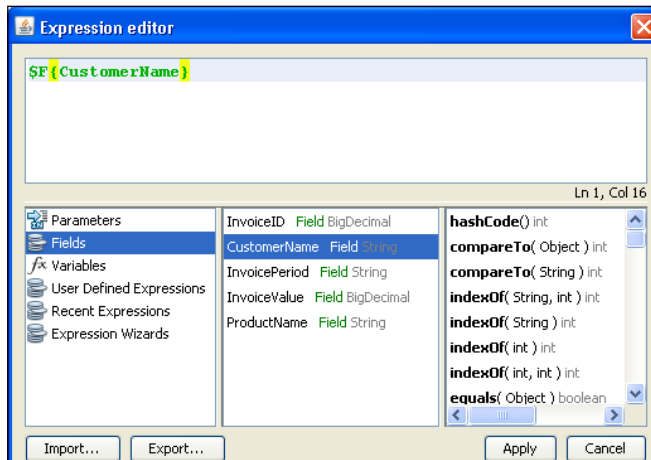
1. Open the `DynamicTitle.jrxml` file from the `Task4` folder in the source code for this chapter. The **Designer** tab of iReport shows an empty report with a simple main title (Monthly Customer Invoices) as shown here:




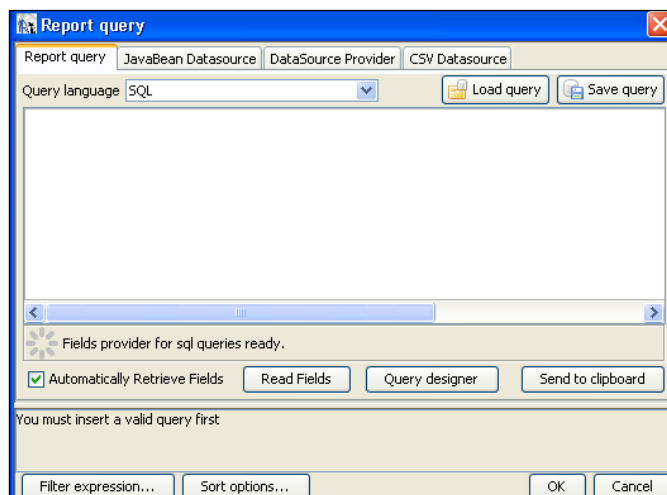
2. You are about to insert a dynamic subtitle into your report. Drag a new **Text Field** from the palette of components available on the right of the main iReport window and drop it into the **Title** section of your report, just below the main title.
3. Right-click on the **Parameters** node in the **Report Inspector** window on the left of the **Designer** tab. Choose **Add Parameter** from the pop-up menu.
4. The **Parameters** node will expand to show all the parameters including the newly added parameter named `parameter1`, which appears at the end of the list of parameters. Select `parameter1` from the expanded view. You will see the properties of this parameter in the **Properties** window on the right of the **Designer** tab just below the palette of components.
5. Click on the **Name** property of the parameter and type `CustomerName` as its value. This sets `CustomerName` as the name of this parameter. Similarly, click on the **Default Value Expression** property of the parameter and type `" "` (two double quotes) as its value. Leave the rest of the properties at their default values.
6. Link the text field you dragged-and-dropped in step 2 earlier with the `CustomerName` parameter. For this purpose, right-click on the text field and select the **Edit expression** option from the pop-up menu. An **Expression editor** window will open as shown below:



7. Delete the default expression (`$F{field}`) from the **Expression editor** window.
8. Click on **Parameters** in the first column of the lower half of the **Expression editor** window. This will list all the parameters in the adjacent second column. Scroll down the list of parameters in the second column to find the `CustomerName` parameter. Double-click on it. This will set `$P{CustomerName}` in the expression editor, as shown below:



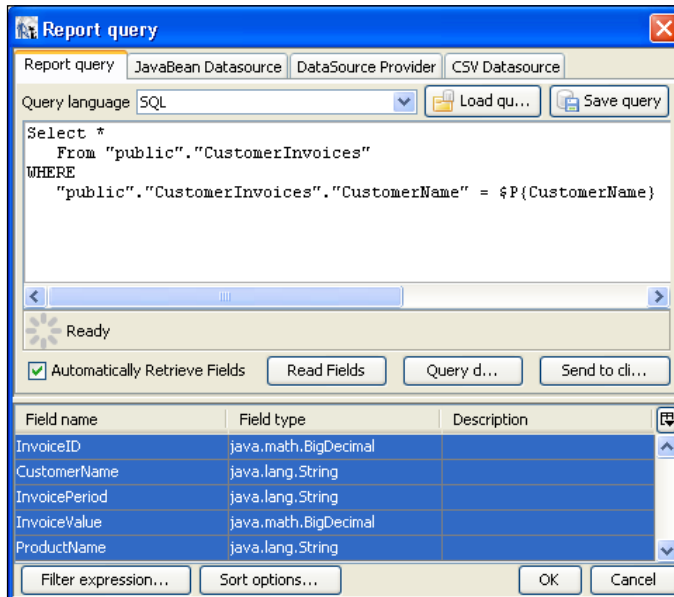
9. Click the **Apply** button. This will attach the `CustomerName` parameter with your text field of step 2.
10. Now open the **Report query** window by clicking the **Report query** button to the right of the **Preview** tab. Unfortunately, it is a bit difficult to find this button in iReport 3.6.0, as it shows no tool tip when you move your mouse over it. Its icon is similar in shape to a standard database icon and looks like . A **Report query** window will appear, as follows:



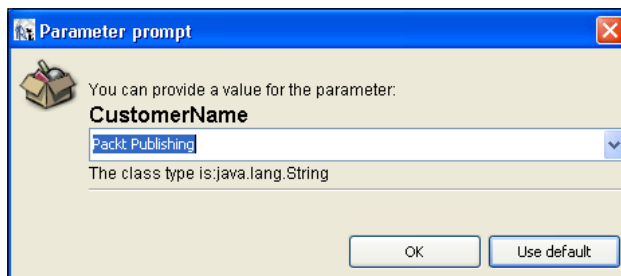
11. Enter the following SQL query into the **Report query** window:

```
SELECT *
FROM "public"."CustomerInvoices"
WHERE
"public"."CustomerInvoices"."CustomerName" = $P{CustomerName}
```

12. The **Report query** window will automatically fetch all the columns of the `CustomerInvoices` table (that is, `InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and show them in the lower part of the **Report query** window, as shown next:



13. Click the **OK** button at the bottom of the **Report query** window.
14. Your text field of step 2 is now fully dynamic. Switch to the **Preview** tab to see your dynamic title. As soon as you switch to the **Preview** tab, the **Parameter prompt** dialog will appear. This dialog will ask you for the name of the customer, as shown in the following screenshot:



15. Type Packt Publishing as the name of the customer and click **OK**. You will see a dynamic title, as shown next.



How it works...

Open the **XML** tab of your report and you will see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <parameter name="CustomerName" class="java.lang.String">
  </parameter>
  <queryString>
    <![CDATA[SELECT * from "public"."CustomerInvoices"
      WHERE "public"."CustomerInvoices"."CustomerName" =
        ${CustomerName}]]>
  </queryString>
  <!-- other JasperReports XML tags -->
  <title>
    <band height="79" splitType="Stretch">
      <staticText>
        <!-- other JasperReports XML tags -->
        <text><![CDATA[Monthly Customer Invoices]]></text>
      </staticText>
      <textField>
        <reportElement x="113" y="49"
          width="128" height="20"/>
        <textElement/>
        <textFieldExpression class="java.lang.String">
          <![CDATA[${CustomerName}]]>
        </textFieldExpression>
      </textField>
      <!-- other static text and text field XML tags -->
    </band>
  </title>
  <!-- other JasperReports XML tags -->
</jasperReport>
```

You can see that this JRXML contains `<parameter>`, `<queryString>`, and `<title>` tags in expanded view, whereas all other tags that are not relevant to the present discussion of the recipe are omitted.

iReport has generated the `<parameter>` tag (shown highlighted in the code) in response to steps 3 and 4 of the recipe. In these steps, you configured the `CustomerName` parameter. The `<parameter>` tag wraps the name of the parameter (`CustomerName`) and the type of data (that is, a Java string) through its `name` and `class` attributes, respectively.

Now look at the `<queryString>` tag (shown highlighted in the code), which wraps the SQL query you authored in step 11 of the recipe to fetch table data from your database.

Also notice from the SQL query that it contains a string `$P{CustomerName}` on the right of the `=` sign in the `where` clause. This `$P` is not SQL syntax. This `$P` will be internally processed by the JasperReports engine, which resolves it to the `CustomerName` parameter before executing the query.

This way, the SQL query becomes part of your JRXML code and the JasperReports engine will process and execute this query to fetch the customer name from your database. JasperReports will also copy the name of the customer into the `CustomerName` parameter while processing your report.

Now look at the `<staticText>` tag whose `<textFieldExpression>` child is shown highlighted in the code. This `<textFieldExpression>` tag specifies the parameter (`CustomerName`) that contains the name of the customer fetched from the database. Therefore, when JasperReports processes this `<staticText>` tag, it will show the name of the customer in the title of your report.



Use the `overflow` attribute. At runtime, if the title width exceeds the given width, it won't appear.

Also, it is better to externalize the static text into PROPERTIES files so that it will be easy to support multilingual support down the line. Also, it will be easy to fix spelling mistakes or modifications, without touching the report.

Inserting a company logo in the title of your report

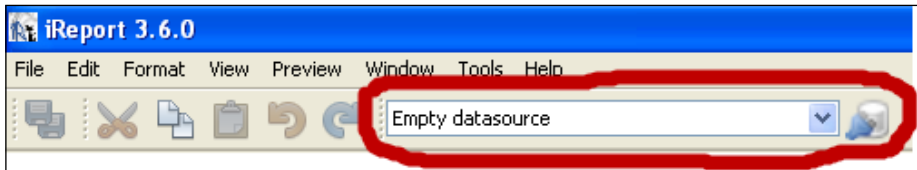
JasperReports allows you to place your company logo anywhere in your report. You may even use your logo as a background image or a watermark in a report.

This recipe shows you how you can place, position, and size your company logo within the title of a business report.

Getting ready

There is no database connection required in this recipe.

If you have not made any database connection so far in your iReport installation, you will see an **Empty datasource** selected in a drop-down list just below the main iReport menu. This is the requirement to run this recipe. If any other option (for example, some database connection) is previously selected, then change it to **Empty datasource** from the drop-down list, as shown in the following screenshot:



How to do it...

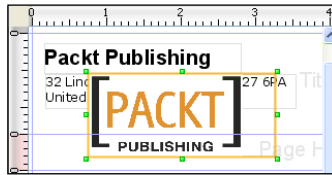
1. Open the `ASimpleTitleWithLogo.jrxml` file from the `Task5` folder of the source code for this chapter. The **Designer** tab of iReport shows an empty report with a title (Packt Publishing) as shown in the following screenshot:



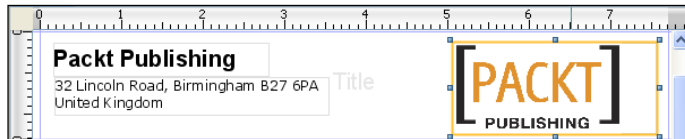
2. Use the palette of components available on the right of your iReport main window to drag an **Image** component into the **Title** section of your report. Now you can browse to your image file that contains your company logo. JasperReports supports the rendering of all popular image formats such as JPG, GIF, BMP, PNG, and so on.
3. Once you have selected the image file, click on **Open**. Your image will be placed in the **Title** section of your report, as follows:



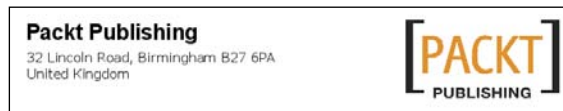
- The logo shown here does not fit and looks a bit too big. You need to adjust the size of the image. Click on the logo; its properties will appear in the **Properties** window just below the **Palette** of components. Look for the **Width** property and change its value to 180. Similarly, look for the **height** property and change it to 80. Now it looks quite normal, but still its placement in the **Title** section looks inappropriate, as shown in the following screenshot:



- Click on the logo and then use the mouse or arrow keys to place it in the top-right corner of the **Title** section, as shown in the following screenshot:



- Switch to the **Preview** tab; you will see the preview of your report as shown here:



How it works...

Switch to the **XML** tab and you will see the following JRXML code:

```
<jasperReport
  <title>
    <band height="94" splitType="Stretch">
      <!-- other JasperReports XML tags -->
      <image>
        <reportElement x="365" y="10"
          width="180" height="80"/>
        <imageExpression class="java.lang.String">
          <![CDATA["X:\\Images\\packt.png"]]>
        </imageExpression>
      </image>
```

```
        </band>
    </title>
    <!-- other JasperReports XML tags -->
</jasperReport>
```

Notice the `<image>` tag, shown highlighted in the preceding code. This `<image>` tag represents the logo you inserted into the **Title** section.

The `<image>` tag has a child named `<imageExpression>`, which specifies the `<image>` that you want to insert. The `<imageExpression>` tag provides two important bits of information:

- ▶ The `class` attribute of the `<imageExpression>` tag specifies what type of image you want to use. In this case, you just want to provide the URL of the image. So it is a Java string. Another possibility can be that you want to pass a Java file object to JasperReports. In that case, the value of the `class` attribute will be `java.io.File`.
- ▶ The content of the `<imageExpression>` tag specifies the actual URL.



Add images to source folders and check the images with code.
Also, try to avoid spaces in image filenames.

Adding a simple header to your report

A header normally appears on every page of your report. The header of your report contains the data that was used to generate the report. For example, if you generate a "Monthly Customer Invoices" report, the header will tell you the name of the customer and the month for which you generated a particular copy of the report.

The header consists of a number of name-value pairs, in which the name is actually the label of a field (for example, the label can be "Customer Name" when the header shows the name of a customer) and is normally simple text. On the other hand, the value in the name-value pair is actually some application data (for example, actual name of a customer) that comes from a data source such as a relational database. In this recipe, you will learn how to place a header just below the title of your report. As an example, I am going to generate the header of a report containing list of invoices issued to a particular customer in a specific month.

Getting ready

In order to work with this recipe, you will need the PostgreSQL database. Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter, which shows how you can install and run PostgreSQL. Note that your installation of PostgreSQL should be up and running before you proceed.

The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`. The `copySampleDataIntoPGS.txt` file will help you to create a database named `jasperdb1`, a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`), and copy sample data for this recipe into the table.


After inserting the required data, you will connect iReport to your database hosted on PostgreSQL using the steps described in the *Creating a report from relational data* recipe later in Chapter 4.

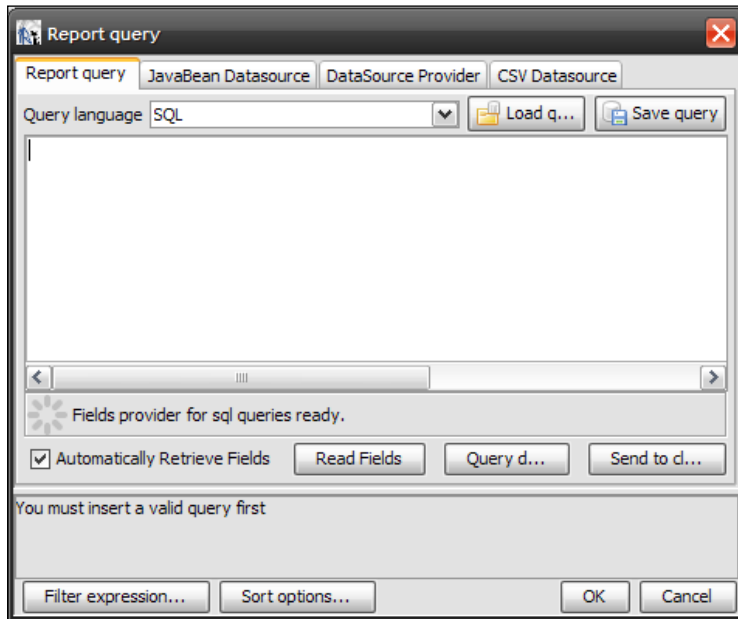
How to do it...

1. Open the `ASimpleHeader.jrxml` file from the `Task6` folder of the source code for this chapter. The **Designer** tab of iReport shows an empty report with just a title (Monthly Customer Invoices).
2. You are about to enhance the report by adding a header. You can start by placing the label for your first name-value pair. Use the palette of components on the right of the main iReport window and drag a static text component into the **Page Header** section in the **Designer** tab of your report.
3. Double-click the static text and enter a label for your component (for example, "Customer Name"). Now use the mouse or keyboard arrow keys to place the static text appropriately, as shown in the following screenshot:



4. Now you will add a value field for your name-value pair. Drag a text field from the palette of components into the **Page Header** section and place it to the right of the **Static Text** labeled "Customer Name." A text field is just like static text except that it is attached to some data source at the backend through a database query or an XPath expression.

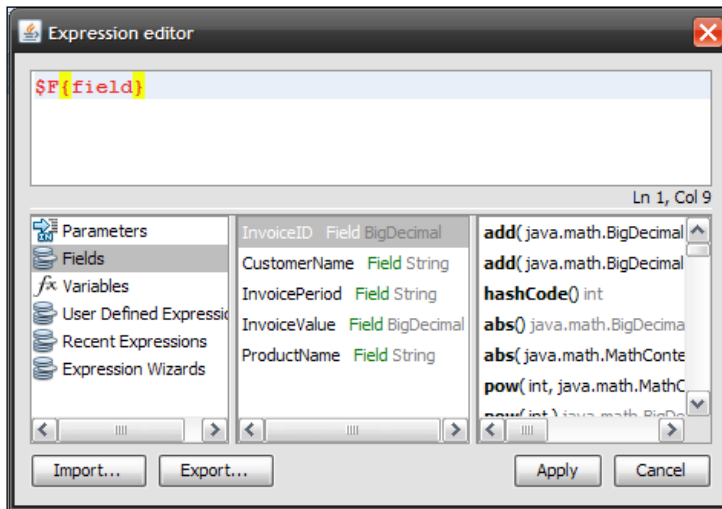
5. Open the **Report query** window by clicking the report query button to the right of the **Preview** tab. Unfortunately, it is a bit difficult to find this button in iReport 3.6.0, as it shows no tool tip when you move your mouse over it. Its icon is similar in shape to a standard database icon and looks like . A **Report query** window will appear, as shown in the following screenshot:



6. Enter the following SQL query into the **Report query** window:


```
SELECT *
from "public"."CustomerInvoices"
WHERE
"public"."CustomerInvoices"."CustomerName" = 'Packt Publishing'
AND "public"."CustomerInvoices"."InvoicePeriod" = 'Mar09'
```
7. The **Report query** window will automatically fetch all columns of the `CustomerInvoices` table according to the condition given in the SQL query. You will see that all the columns of the `CustomerInvoices` table (that is `CustomerName`, `CustomerPhone`, `InvoicePeriod`, `ProductName`, and `TotalValue`) will be displayed in the lower part of the **Report query** window, as shown next. Click the **OK** button and exit the **Report query** window.

8. Now you have to link the CustomerName column of the CustomerInvoices table with the text field you placed into your **Page Header** section in step 5. Right-click the text field and select the **Edit Expression** option from the pop-up window, as shown here:



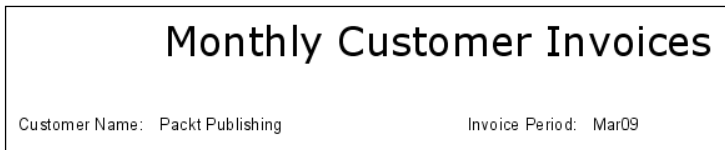
9. An **Expression editor** window will open. Delete the existing text (`$F{field}`) from the **Field Expression** window and then double-click the CustomerName field in the second column of the lower half of the **Field Expression** window and click **Apply**. This will attach the CustomerName column of the CustomerInvoices table with the text field.
10. Now drag another static text and text field components as per steps 2, 3, and 4 for your second name-value pair (that is, Invoice Period = "invoicing month") as shown next. This time the static text will get Invoice Period as its label and the text field will link to the InvoicePeriod column of the CustomerInvoices table.



11. Repeat steps 8 and 9 to attach the InvoicePeriod column with the Invoice Period text field of your **Page Header** section. Eventually, your header will look like the following:



12. Switch to the **Preview** tab; you will see the preview of your report as shown here:



How it works...

Open the **XML** tab of your report and you will see the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other tags of JasperReports XML -->
  <queryString>
    <![CDATA[SELECT * from CustomerInvoices
              WHERE CustomerName = 'Packt Publishing' AND
              ProductName = 'Offset Paper' AND
              InvoicePeriod = 'Mar09']]>
  </queryString>
  <pageHeader>
    <band height="39" splitType="Stretch">
      <staticText>
        <reportElement x="120" y="11" width="115"
                      height="22"/>
        <textElement textAlignment="Left">
          <font size="12"/>
        </textElement>
        <text><![CDATA[Customer Name]]></text>
      </staticText>
      <textField>
        <reportElement x="128" y="12" width="100"
                      height="20"/>
```

```

        <textFieldExpression class="java.lang.String">
            <![CDATA[$F{CustomerName}]]>
        </textFieldExpression>
    </textField>
    <!-- other static text and text field tags -->
</band>
</pageHeader>
<!-- other tags of JasperReports XML -->
</jasperReport>

```

Notice the `<queryString>` tag that I have shown highlighted in the preceding JRXML code. The `<queryString>` tag wraps the SQL query that you authored in step 6 of this recipe. This query brings all the five columns of the customer invoice table, each of which can be attached to a text field.

You can also see a `<pageHeader>` tag in this JRXML code. The `<pageHeader>` tag represents the header of your report, which you just authored in this recipe. The `<band>` child of the `<pageHeader>` tag contains two pairs of `<staticText>` and `<textField>` tags. I have shown only one pair for clarity. You can guess that each pair corresponds to a name-value pair you dragged-and-dropped in the **Page Header** section of your report. Notice that the `<textField>` tag has a child named `<textFieldExpression>`. The `<textFieldExpression>` has an expression (`$F{CustomerName}`) as its content. `$F{CustomerName}` refers to the name of a column in the `CustomerInvoices` table you linked with the text field in steps 8 and 9 of this recipe.

Changing this expression in the **XML** tab will affect the report in the **Designer** and **Preview** tabs as well. For example, if you delete the `$F{CustomerName}` expression in the **XML** tab, you will see that there is no content shown as the value of that text field in the **Designer** tab.

There's more...

If you look at the SQL query that you entered into the **Report query** window, you will see that the name of the customer and the invoicing month have been hardcoded in the query. This data normally comes from the user who generates the report.

In order to allow the user to provide these values, you will need to declare a parameter and link it to your text fields (that is, the customer name and invoicing month) according to steps 3 to 9 of the *Using dynamic titles that can change during report processing* recipe of this chapter.

In this case, you also need to have a dynamic query, which will use the parameters that you have linked to your text fields. So the fixed SQL query of step 6 of this recipe will change to the following dynamic query:

```
SELECT * from "public"."CustomerInvoices"
WHERE "public"."CustomerInvoices"."CustomerName" = ${CustomerName}
      AND "public"."CustomerInvoices"."InvoicePeriod" =
                                                ${InvoiceMonth}
```

Now when you switch to the **Preview** tab, iReport will not show you the preview immediately. It will first ask you to provide a customer name and invoicing month. Enter *Packt Publishing* as the customer name and *Mar09*, as invoicing month; iReport will now display the preview.

Also note that, once entered, iReport remembers parameter values, so it will not ask you to re-enter parameter values each time you generate the preview. It will ask you for new parameter values only when you close and reopen your report in iReport or make any change in the design view.

Setting margins for your report and aligning the report header relative to report margins

You will often need to set margins for your report as well as its individual sections such as the report title, header, body, and footer.

This recipe shows you how you can set the margins for your report and also how to align the different components of your header with respect to each other as well as the report margin.

Getting ready

In order to work with this recipe, you will need the PostgreSQL database. Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter, which shows how you can install and run PostgreSQL. Note that your installation of PostgreSQL should be up and running before you proceed.

The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`. The `copySampleDataIntoPGS.txt` file will help you create a database named `jasperdb1`, a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data for this recipe into the table.

After inserting the required data, you will connect iReport to your database hosted on PostgreSQL using the steps described in the *Creating a report from relational data* recipe later in *Chapter 4*.

How to do it...

1. Open the `HeaderMarginAndAlignmentSample.jrxml` file from the `Task7` folder of the source code for this chapter. This will display a report with a title and a multi-line header containing three name-value pairs of components, as shown next. Each name-value pair contains a static text and a text field component. For example, the first name-value pair shown contains a static text component (that is, `Customer Name:`) and a text field component (that is, `#{CustomerName}`).

Monthly Customer Invoices

Customer Name: `#{CustomerName}` Product Name: `#{ProductName}`

Invoice Period: `#{InvoicePeriod}`

2. Click on the **Format** menu and then choose the **Page format** item from the drop-down menu, as shown here:

Page format...

Format: `A4`

Width: `8.264` inches

Height: `11.694` inches

Page orientation

☒ Portrait

☐ Landscape

Margins

Top: `0.278` inches

Bottom: `0.278` inches

Left: `0.278` inches

Right: `0.278` inches

Columns

Columns: `1`

Column width: `7.708` inches

Space: `0` inches

Unit: `inches`

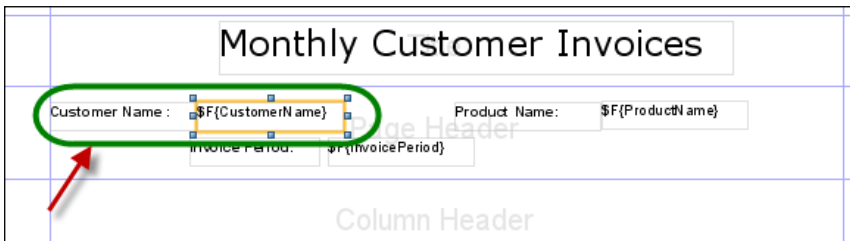
Ok Cancel

3. From the **Page format** window you may choose the format, report dimensions, margins, and page orientation of your report. This recipe focuses on margins. Other features will be covered in later recipes. Choose **0.5** inches as the left margin (a bit more than the default value) because most reports need some binding on the left side. Leave all other margins (that is right, top, and bottom) at their default values.

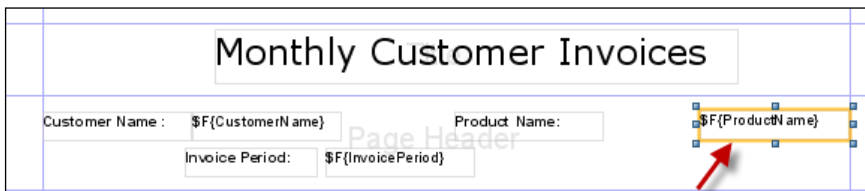
- You will notice that the name-value pairs of the multi-line header are not properly aligned. You can align first two pairs in two columns and center-align the third. Right-click on the **Static Text** component (**Customer Name:**) of the first name-value pair. A pop-up window will appear. Select **Align** from the pop-up menu, then select the **Align to Left Margin** option from the pop-up menu. You will see that the static text component is shifted to the extreme left side of the report margin, as shown next:



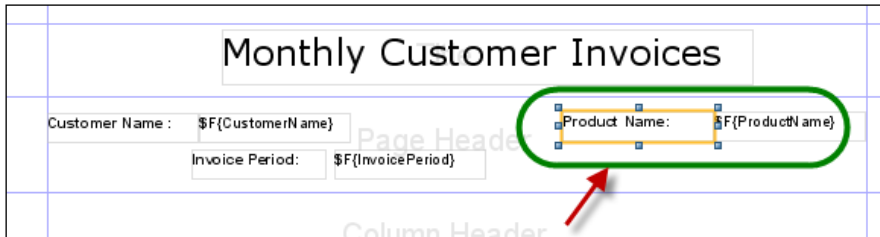
- Click on the text field component (**\${CustomerName}**) of the **Customer Name:** pair. Use the mouse or arrow keys to drag it to the left till it sits adjacent to the **Customer Name** static text component.



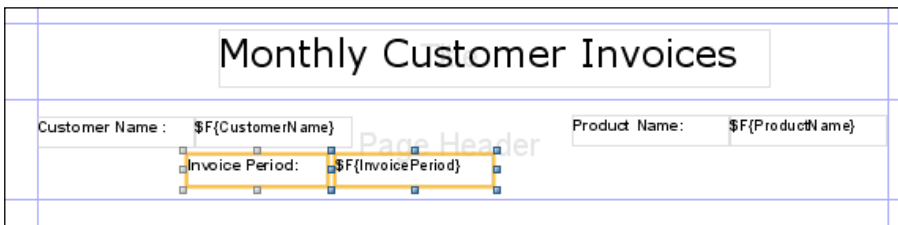
- Right-click on the text field component of the second name-value pair. A pop-up menu will appear; select **Align** from the pop-up menu, and then choose the **Align to Right Margin** option from the menu. This will shift the text field component to the extreme right of the report margin, as follows:



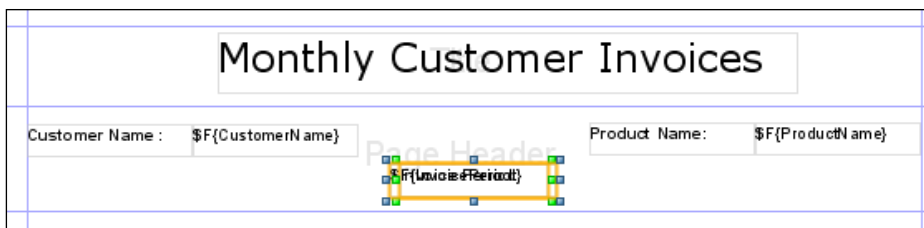
7. Now click on the static text component of the second name-value pair (**Invoice Period:**). Use the mouse or arrow keys to drag it to right till it sits adjacent to the accompanying text field component you right-aligned in step 6, as shown below.



8. Now look at the third name-value pair, which is on the second line of the header. You want to centre-align this name-value pair. Unfortunately, there is no way to accurately center-align this name-value pair using iReport. You will have to do it manually. For this purpose, hold down the *Ctrl* key and select the static text component (that is **Product Name:**). While holding the *Ctrl* key down, select the **Text Field** component of this name-value pair. This will select both components simultaneously, as shown next:



9. Right-click on the selected name-value pair; a pop-up menu will appear. Select the **Position** option from the pop-up menu. A submenu will pop up; choose the **Center Horizontally** option from the submenu. This will center-align the selection, making the two components overlap each other, as shown in the following screenshot:



10. Left-click on the name-value pair, which is now overlapped and center-aligned. This will select the text field component of the name-value pair. Press the right key 48 times to move the component to the right. Now left-click on the static text component and press the left-arrow key 48 times to move the static text component to the left. Now this name-value pair is approximately center-aligned, as shown here:

<h2>Monthly Customer Invoices</h2>	
Customer Name :	\$F{CustomerName}
Product Name:	\$F{ProductName}
Invoice Period:	\$F{InvoicePeriod}

11. Save your report by selecting **Save** from the **File** menu and switch to the **Preview** tab. You will get a preview as shown here:

<h2>Monthly Customer Invoices</h2>	
Customer Name :	Packt Publishing
Product Name:	Packing Material
Invoice Period:	Mar09

How it works...

Switch to XML tab to see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport leftMargin="36" rightMargin="20"
  topMargin="20" bottomMargin="20">
  <!-- other tags of JasperReports XML -->
  <pageHeader>
    <band height="63" splitType="Stretch">
      <staticText>
        <reportElement x="0" y="11"
          width="100" height="20"/>
        <textElement>
          <font isBold="true"/>
        </textElement>
        <text><![CDATA[Customer Name :]]></text>
      </staticText>
      <textField>
```

```

        <reportElement x="100" y="11"
                      width="100" height="20"/>
        <textElement/>
        <textFieldExpression class="java.lang.String">
            <![CDATA[${F{field}}]>
        </textFieldExpression>
    </textField>
    <!-- other tags of JasperReports XML -->
</band>
</pageHeader>
<!-- other tags of JasperReports XML -->
</jasperReport>

```

In this JRXML, you can see from the `<jasperReport>` tag that its attribute named `leftMargin` is highlighted. Notice that you set the `leftMargin` attribute to **0.5** inches in step 3 of this recipe. iReport automatically translates the value **0.5** inches into a corresponding value of the `leftMargin` attribute.

The `<band>` tag of `<pageHeader>` contains three pairs of `<staticText>` and `<textField>` tags. I have shown only one name-value pair for the sake of clarity. The `<staticText>` and `<textField>` tags contain a `<reportElement>` child. Look at the `x` and `y` attributes of the `<reportElement>` tag that I have shown highlighted. The `y` attributes of both `<reportElement>` tags have the same values. The `x` attributes are different and their difference is equal to the width of the first component (the `width` attribute of the first `<reportElement>` tag).



The best way to judge these attribute values is by printing the final report.

2

Working with the Body and Footer of your Report

In this chapter, you will learn:

- ▶ Displaying a field along with its label in the body of your report and handling null values
- ▶ Creating a simple table of records along with labels for each column
- ▶ Inserting a heading for a group of records
- ▶ Using parameters to filter records during report processing
- ▶ Implementing groups within groups—a nested hierarchy
- ▶ Adding a simple footer to your report
- ▶ Displaying general information or summary at the end of your report

Introduction

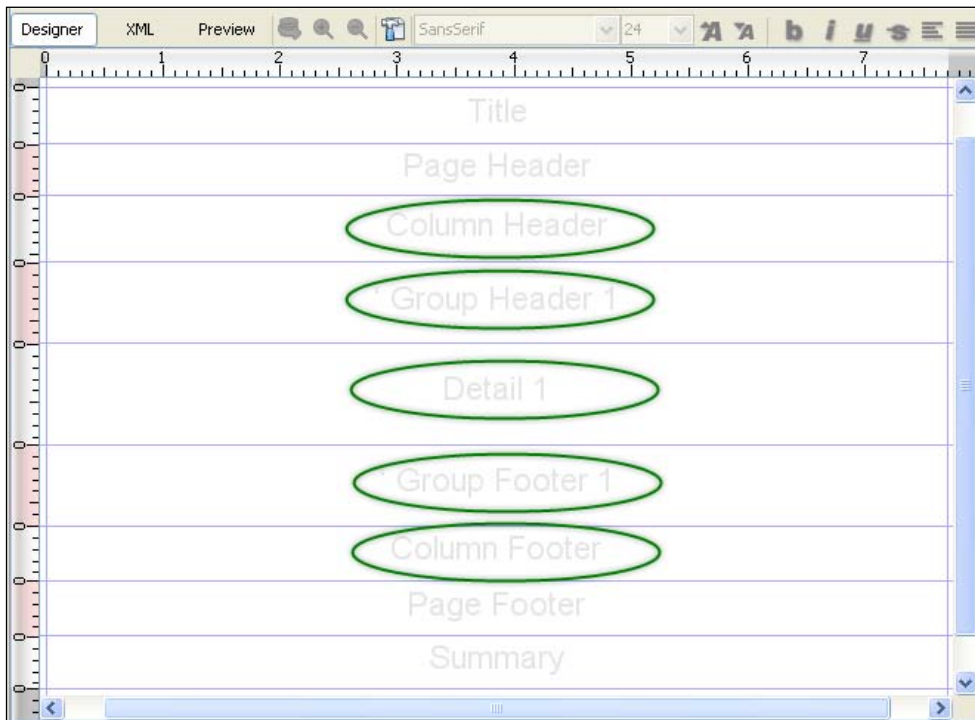
This chapter is about the main body and footer of your report.

The body contains the actual data that constitutes your report, such as tables and the output of calculations performed on your data (such as the sum or average of important field values). For example, if you are designing a production report, the body will contain the actual production data in tabular or another format.

The Design preview of iReport provides several sections that are all used as parts of the report body portion:

- ▶ **Column Header**
- ▶ **Group Header 1**
- ▶ **Detail 1**
- ▶ **Group Footer 1**
- ▶ **Column Footer**

The following is a screenshot showing these sections:



As you go through the recipes of this book, you will notice that there are some sections that appear in the body of your report by default, such as the Column Header section, the Detail 1 section, and the Column Footer section.

The main body of a report normally consists of a table. The **Column Header** and **Column Footer** sections provide the header and the footer for the table. The **Detail 1** section sits between the **Column Header** and **Column Footer** sections and forms its main body, showing actual records.

The rest of the sections (**Group Header** and **Group Footer** sections) shown in the preceding screenshot do not appear by default. These sections are specific and appear only when required by the report designer. These are custom sections and the recipes of this chapter show you when and how to include these sections into your report.

The recipes of this chapter cover these sections in detail. You will learn different tricks and techniques to design tables as well as group the data in your report.

The last two recipes of this chapter cover page footer and summary sections, which you will normally find at the end of your report in the designer view.

A **page footer** normally provides useful information about the report, such as:

- ▶ Name of your report
- ▶ Date and time when you generated the report
- ▶ Page numbering
- ▶ Sum or average of important field values

Displaying a field along with its label in the body of your report and handling null values

This recipe teaches you the simplest operation you can do inside the body of your report. This includes displaying your report data as name-value pairs in the body. The name-value pair consists of a label (the name) and a field value (the value). The recipe also teaches you how to associate the field value with a column in your database table.

Getting ready

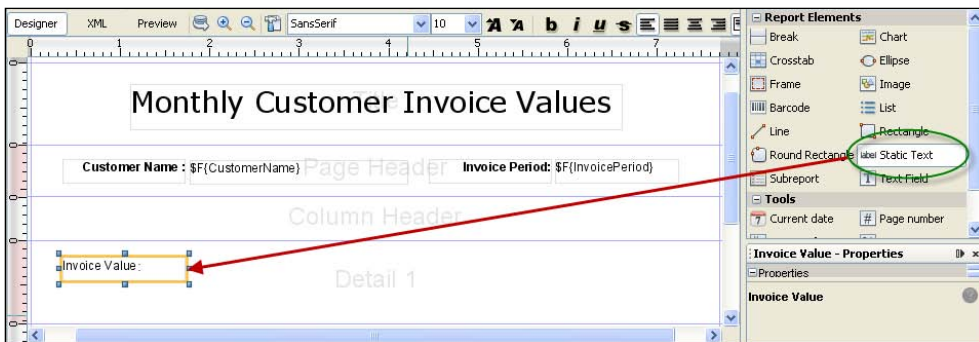
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb2` and copy sample data for this recipe into the database.

How to do it...

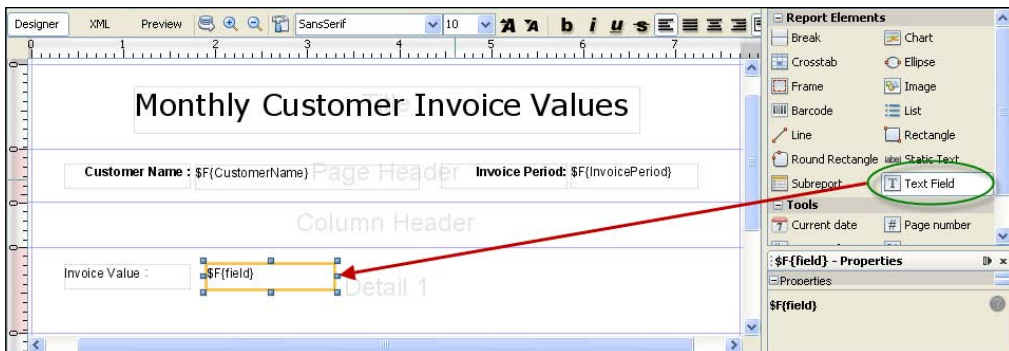
1. Open the `SingleFieldBody.jrxml` file from the `Task1` folder of the source code for this chapter. The **Designer** tab of iReport shows a title and a header:




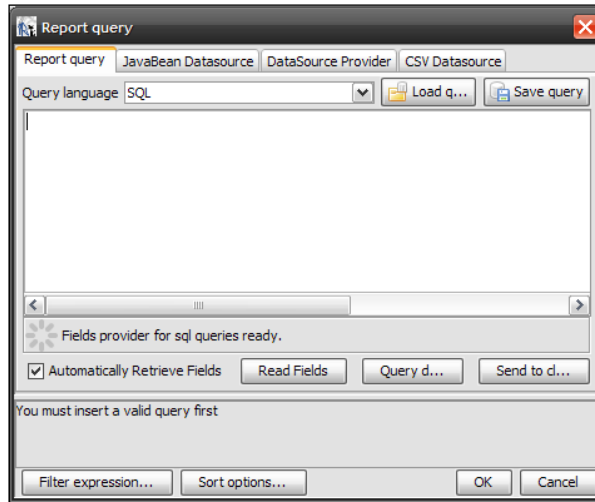
2. Drag-and-drop a new static text component from the **Palette** of components into the **Detail 1** section of your report. Double-click the static text component and enter `Invoice Value` as its label. Position it to the extreme left of your report.



3. Now, drag-and-drop a text field component into the **Detail 1** section. Place the text field to the right of the static text you already dropped and positioned in step 2. This text field will show the value of each invoice issued to a customer in a month.



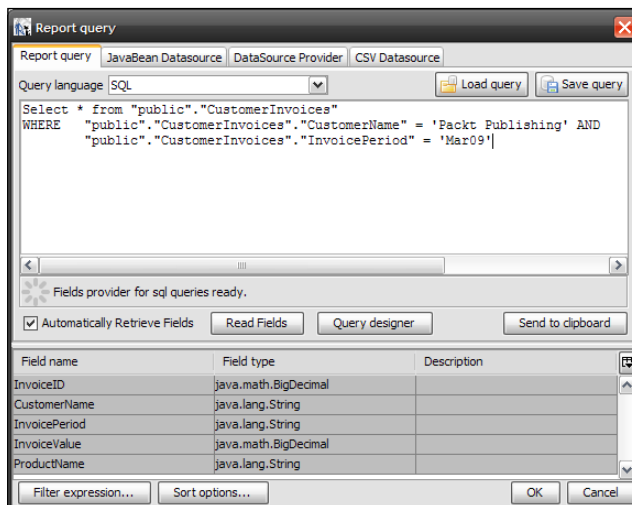
- Open the **Report query** window by clicking the **Report query** button to the right of the **Preview** tab. Its icon is similar in shape to a standard database icon and looks like . A **Report query** window will appear, as shown here:



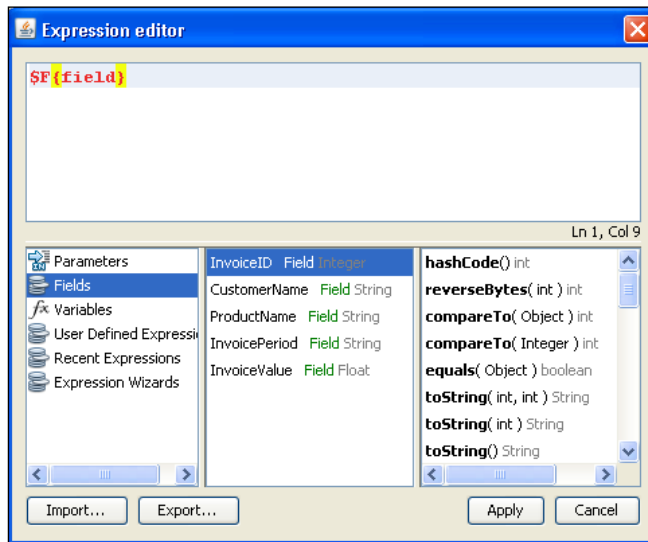
- Enter the following SQL query into the **Report query** window:

```
Select * from "public"."CustomerInvoices"
WHERE  "public"."CustomerInvoices"."CustomerName" = 'Packt Publishing'
AND "public"."CustomerInvoices"."InvoicePeriod" = 'Mar09'
```

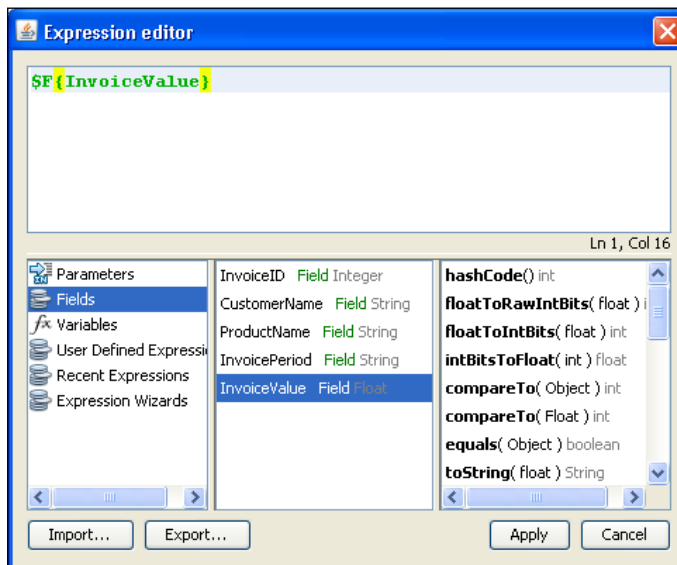
The lower part of the **Report query** window will show all five columns of the **CustomerInvoices** table. Click **OK**.



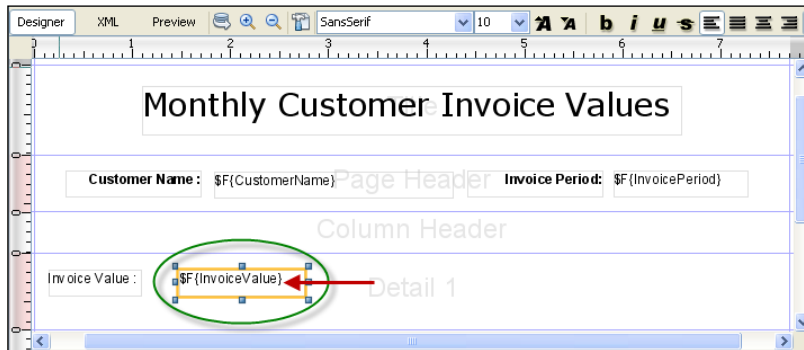
- Now you will link the text field you created in step 3 with the InvoiceValue column of the CustomerInvoices table. For this purpose, right-click the text field and select the **Edit expression** option from the pop-up menu. An **Expression editor** window will open, as shown:



- Delete the existing text (`${field}`) from the **Expression editor** window and select **Fields** in the first column of the lower half of the **Expression editor** window. Then select InvoiceValue in the second column, as shown.



8. Double-click InvoiceValue in the second column and click the **Apply** button. This will attach InvoiceValue with the text field:



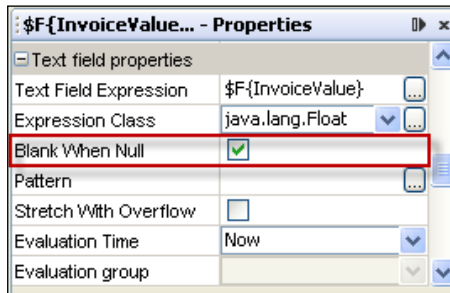
9. Now you will decrease the height of the Detail 1 section. For this purpose, click inside the Detail 1 section; its properties will appear below the **Palette** of components on the right of your iReport main window. Set **40** as the value of the **Band height** property in the **Properties** window. The Detail 1 section will look as follows:



10. Switch to the **Preview** tab and you will see that your report looks like the following. Also note that your report body contains null values, shown encircled in the screenshot here:

Monthly Customer Invoice Values	
Customer Name :	Packt Publishing
Invoice Month:	Mar09
Invoice Value :	5030.25
Invoice Value :	4180.05
Invoice Value :	2090.0
Invoice Value :	8098.5
Invoice Value :	5085.0
Invoice Value :	5085.0
Invoice Value :	8500.0
Invoice Value :	6522.0
Invoice Value :	8999.0
Invoice Value :	null
Invoice Value :	null
Invoice Value :	5030.25
Invoice Value :	7532.0
Invoice Value :	null
Invoice Value :	1589.0
Invoice Value :	4895.0

11. Click on the `#{InvoiceValue}` text field component in the **Detail 1** section. Its properties will appear in the **Properties** window just below the **Palette** of components. Find the **Blank When Null** property and check it, as shown in the following screenshot:



12. Click the **Preview** button and you will see that the null values are replaced with blank spaces, as shown in the following screenshot:

Monthly Customer Invoices	
All invoices for a specific customer, in a specific period of time and for a specific product	
Customer Name :	Packt Publishing
Invoice Period :	Mar09
Invoice Value :	5020.25
Invoice Value :	4150.05
Invoice Value :	2050.0
Invoice Value :	8058.5
Invoice Value :	5085.0
Invoice Value :	5085.0
Invoice Value :	8500.0
Invoice Value :	6522.0
Invoice Value :	8999.0
Invoice Value :	
Invoice Value :	
Invoice Value :	5020.25
Invoice Value :	7532.0
Invoice Value :	

How it works...

Switch to the **XML** tab and you will see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <queryString>
    <![CDATA[SELECT * from "public"."CustomerInvoices"
      WHERE "public"."CustomerInvoices"
        ."CustomerName" = 'Packt Publishing' AND
        "public"."CustomerInvoices"
        ."InvoicePeriod" = 'Mar09']]>
  </queryString>
  <!-- other JasperReports XML tags -->
  <detail>
    <band height="40" splitType="Stretch">
      <staticText>
        <reportElement x="10" y="12" width="69" height="20"/>
        <textElement>
          <font isBold="false"/>
        </textElement>
        <text><![CDATA[Invoice Value :]]></text>
      </staticText>
      <textField isBlankWhenNull="true">
        <reportElement x="106" y="12" width="93" height="20"/>
        <textElement/>
        <textFieldExpression class="java.lang.String">
          <![CDATA[${InvoiceValue}]]>
        </textFieldExpression>
      </textField>
    </band>
  </detail>
  <!-- other JasperReports XML tags -->
</jasperReport>
```

You can see that the `<queryString>` tag is highlighted in this JRXML code. The `<queryString>` tag wraps the SQL query that you authored in step 5 of this recipe. This query fetches all the records of the `CustomerInvoices` table.

You will also see a `<detail>` tag in the preceding JRXML code. The `<detail>` tag represents the body of your report, which you just authored in this recipe. The `<band>` child of the `<detail>` tag specifies the height of the section that you set in step 9.

The `<band>` tag contains a pair of `<staticText>` and `<textField>` tags. This pair of tags represents the label and value components you dragged-and-dropped in steps 2 and 3.

The `<textField>` tag has a child named `<textFieldExpression>`, shown highlighted in this JRXML code. The `<textFieldExpression>` tag has an expression (`$F{InvoiceValue}`) as its content. This `$F{InvoiceValue}` expression attaches the text field with the `InvoiceValue` column of your database. You authored this attachment in step 7 of the recipe, when you double-clicked `InvoiceValue` in the field expression window.

Finally, note that anything put inside the `<detail>` tag will be repeated for every record. You can simply say that JasperReports evaluates the data of the **Detail 1** section once for every record. Then it appends the evaluated data at the end of what it already contains from the previous evaluation. This way JasperReports evaluates the **Detail 1** section record by record until all the records are delivered. This is why the preview of step 10 shows the invoice values for all the records.

Creating a simple table of records along with labels for each column

In this recipe you will learn how to design and display a table formulation in the main body of your report. Table formulation includes the following steps:

1. Placing column headings in the table.
2. Linking your report to fields in your database.
3. Placing data fields under their corresponding column headings.

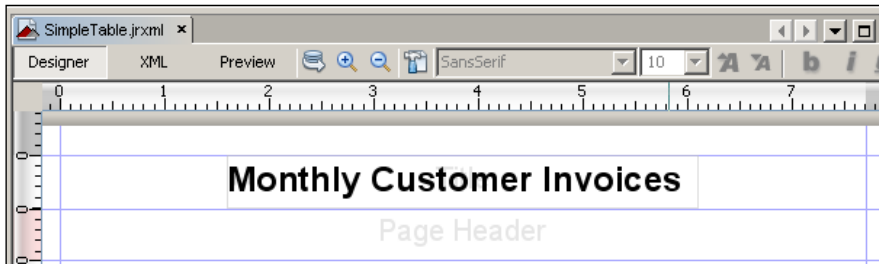
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb2` and copy sample data for this recipe into the database.

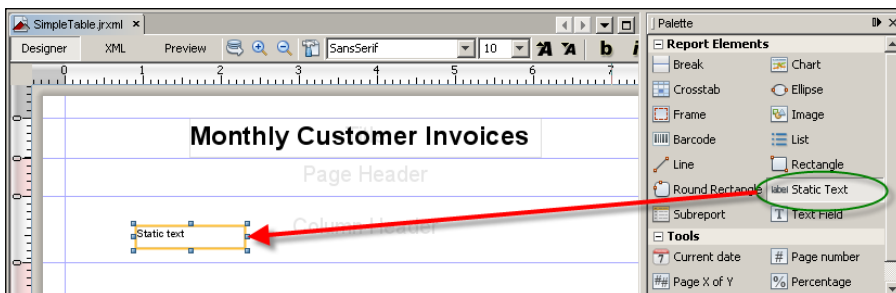
How to do it...

The following steps take you through creating a table:

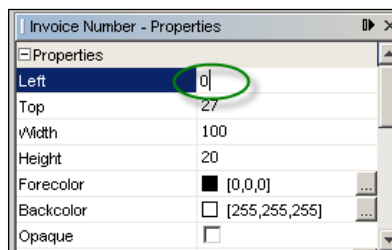
1. Open the `SimpleTable.jrxml` file from the `Task2` folder in the source code for this chapter. The **Designer** tab of iReport shows an empty report with a title (**Monthly Customer Invoices**) as shown in the following screenshot:



2. Select a static text component from the palette and drag it into the Column Header section of your report, as shown in the following screenshot:



3. Double-click on the static text component, and the text inside will be selected; type `Invoice Number` and press **Enter**.
4. While the static text component is selected, select the **Left** property from the **Properties** section of the **Invoice Number - Properties** window below the palette. Change the value of this property to `0`, as shown in the following screenshot:



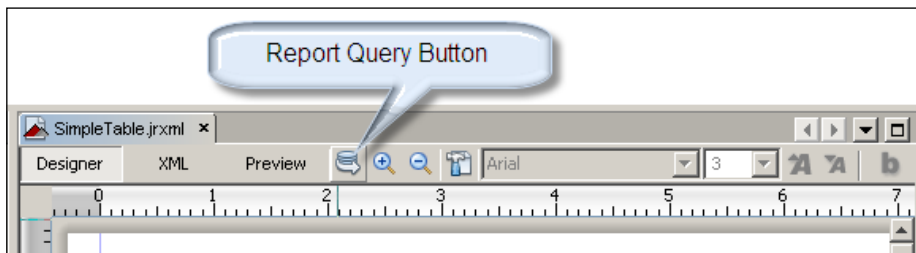
5. Similarly, change the **Top** property value to 0. The static text component will be positioned to the top-left area in the Column Header section of your report, as shown in the following screenshot:



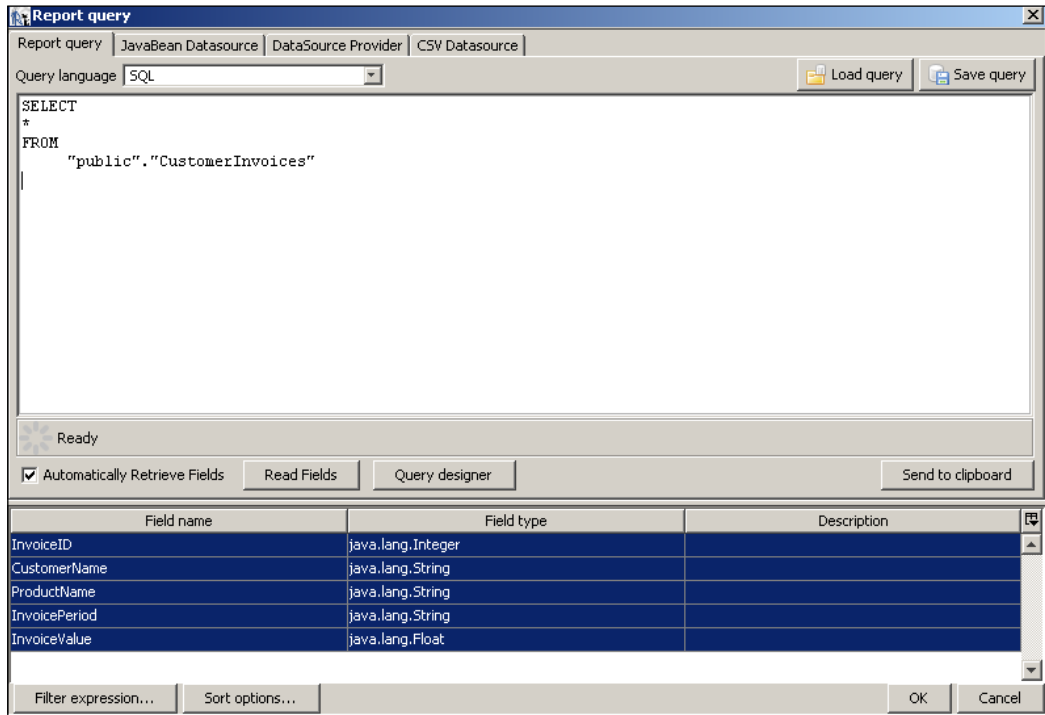
6. From the **Report Inspector** window on the left side of the report window, try to expand the **Fields** node. You will notice that it is empty and, therefore, not expandable, as shown in the following screenshot:



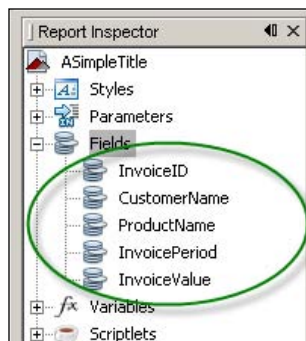
7. Click the **Report Query Button** at the top of the report window.



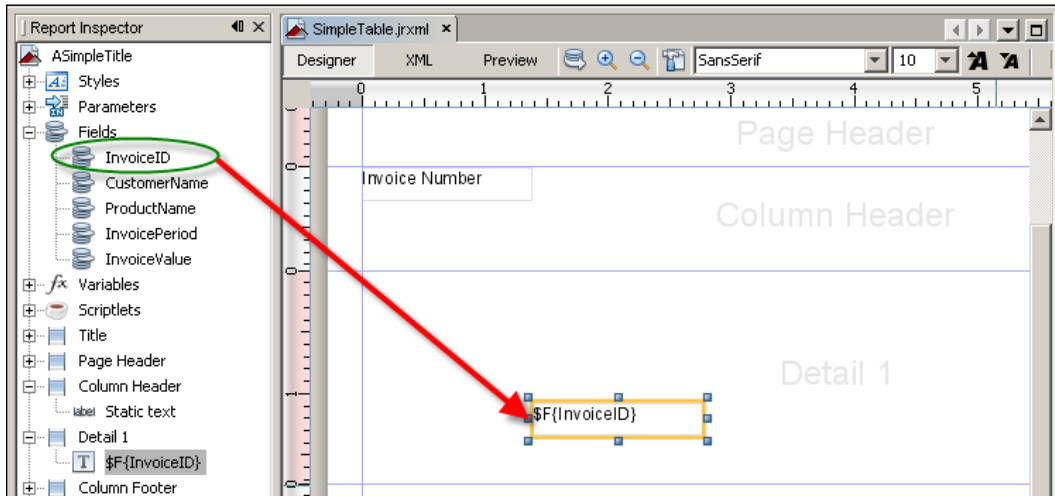
8. A **Report query** dialog will appear. While your cursor is in query editor, type `SELECT * FROM "public"."CustomerInvoices"`. Five field names of the `CustomerInvoices` table (**InvoiceID**, **CustomerName**, **ProductName**, **InvoicePeriod**, and **InvoiceValue**) will appear in the bottom section, as shown in the following screenshot:



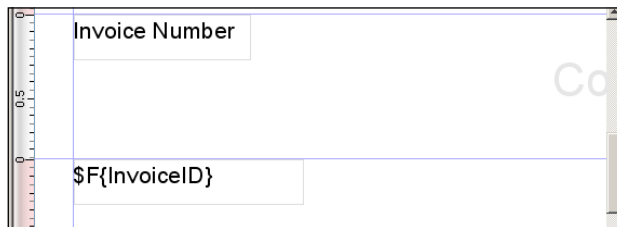
9. Click the **OK** button to dismiss the dialog. From the **Report Inspector** window on the left side of the report window, expand the **Fields** node. You will notice that this time it contains **InvoiceID**, **CustomerName**, **ProductName**, **InvoicePeriod**, and **InvoiceValue** fields, as shown in the following screenshot:



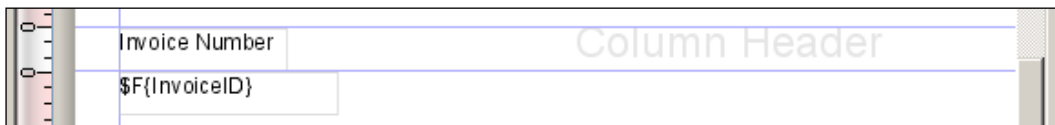
10. Drag-and-drop the **InvoiceID** field from the **Fields** node into the **Detail 1** section of your report. A text field component with the expression `$F{InvoiceID}` will appear in the **Detail 1** section of your report, as shown in the following screenshot:



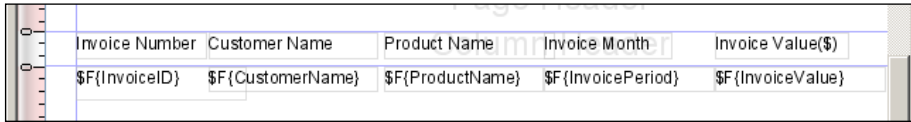
11. Using the arrow keys, move your text field component, so that it appears immediately below the blue line that separates the **Column Header** and the **Detail 1** sections.
12. Now align this text field component using the arrow keys just below the static text component of step 2 (which is a label for the `InvoiceID` Field), as shown in the following screenshot:



13. Double-click the blue line that separates the **Column Header** section and the **Detail 1** section. This will adjust the band height for the **Column Header** section, as shown in the following screenshot:

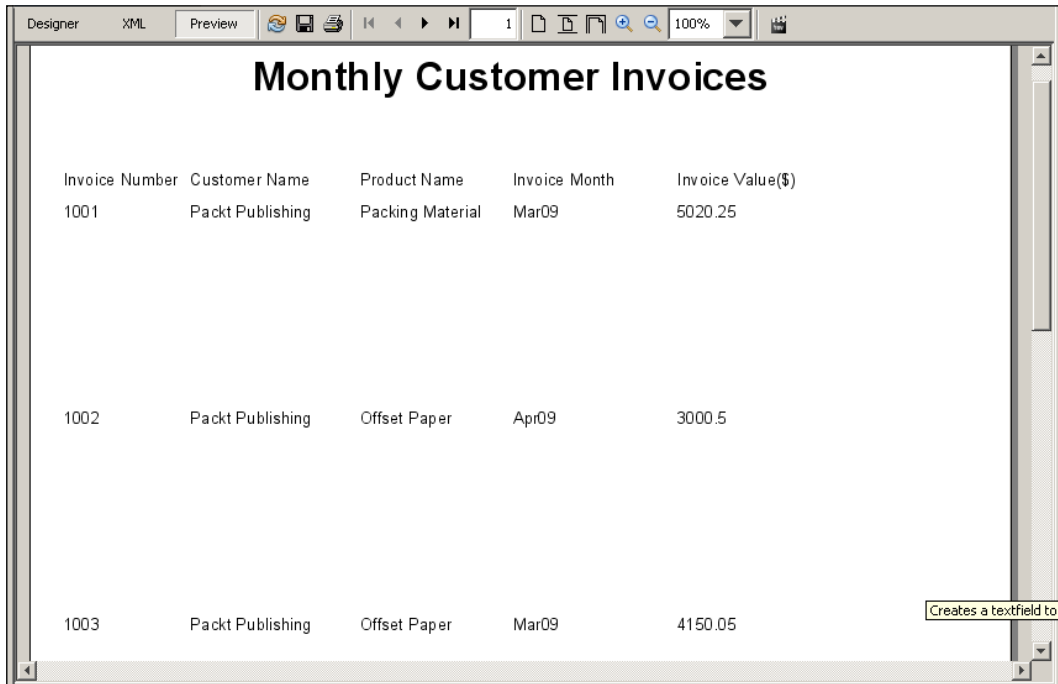


14. Similarly, add the `CustomerName`, `ProductName`, `InvoicePeriod`, and `InvoiceValue` table fields along with their corresponding static text labels into your report. After adding these components your report will look similar to the one in the following screenshot:



Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\${InvoiceID}	\${CustomerName}	\${ProductName}	\${InvoicePeriod}	\${InvoiceValue}

15. Switch to the **Preview** tab to see the result of your work, as follows:



Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05

16. Notice that in your table in the **Preview** tab there is a large whitespace between each record of your table.

17. Switch back to the **Designer** tab. Double-click on the blue line that separates the **Detail 1** section and the **Column Footer** section. This will adjust the band height for the **Detail 1** section, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\${InvoiceID}	\${CustomerName}	\${ProductName}	\${InvoicePeriod}	\${InvoiceValue}

18. Switch to the **Preview** tab to see the result of your work, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1006	Bob	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1037	Bob	JasperReports	Mar09	50.0
1038	Alice	JasperReport	Mar09	49.5
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5

How it works...

Switch to the **XML** tab to see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other JasperReports XML tags -->
  <columnHeader>
```



```
<band height="20" splitType="Stretch">
  <staticText>
    <reportElement x="0" y="0" width="77" height="16"/>
    <textElement/>
    <text><![CDATA[Invoice Number]]></text>
  </staticText>
  <!-- other static text tags -->
</band>
</columnHeader>
<detail>
  <band height="20" splitType="Stretch">
    <textField>
      <reportElement x="0" y="0" width="100" height="20"/>
      <textElement/>
      <textFieldExpression class="java.lang.Integer">
        <![CDATA[${F{InvoiceID}}]>
      </textFieldExpression>
    </textField>
    <!-- other text field tags -->
  </band>
</detail>
<columnFooter>
  <band height="45" splitType="Stretch"/>
</columnFooter>
<!-- other JasperReports XML tags -->
</jasperReport>
```

You can see the `<columnHeader>`, `<detail>`, and `<columnFooter>` tags in this code. These three tags form the report body (the table) that you designed in this recipe. I have omitted other tags for clarity.

Note that when you executed steps 2 to 5 of the recipe, iReport authored the `<columnHeader>` tag. Similarly, when you executed steps 10 to 12, iReport authored the `<detail>` tag.

As the recipe did not do anything with the column footer, the `<columnFooter>` tag is empty.

Inserting a heading for a group of records

You may need to display headings for a subset of records in your report at certain places. For example, if you are generating a weekly or monthly report of all customer invoices, you may want to display a heading for each subset or group of records corresponding to an individual customer. This is sometimes convenient, as it allows you to generate just one report (that is, a report of all invoices issued in a month), which actually consists of many reports (that is, a report for each of your customers).

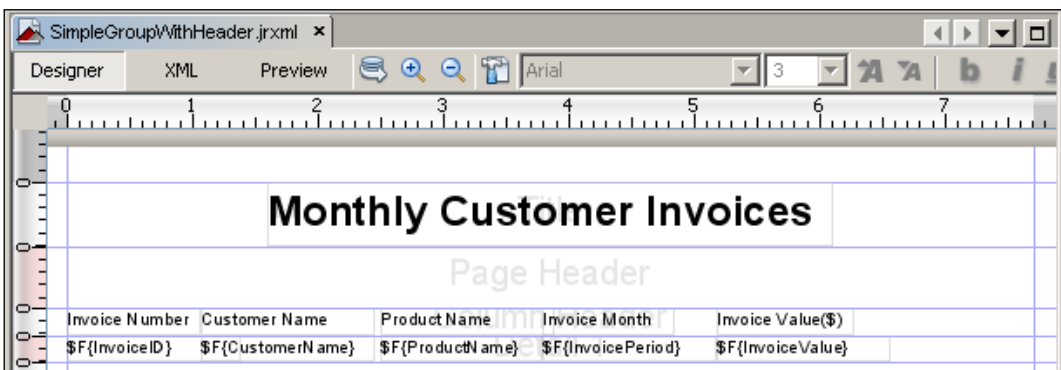
JasperReports offers an interesting feature known as grouping of records, which you can use to do many tricks in report designing. This recipe demonstrates how to use the grouping feature to generate a dynamic heading for a group corresponding to a customer name in your report.

Getting ready

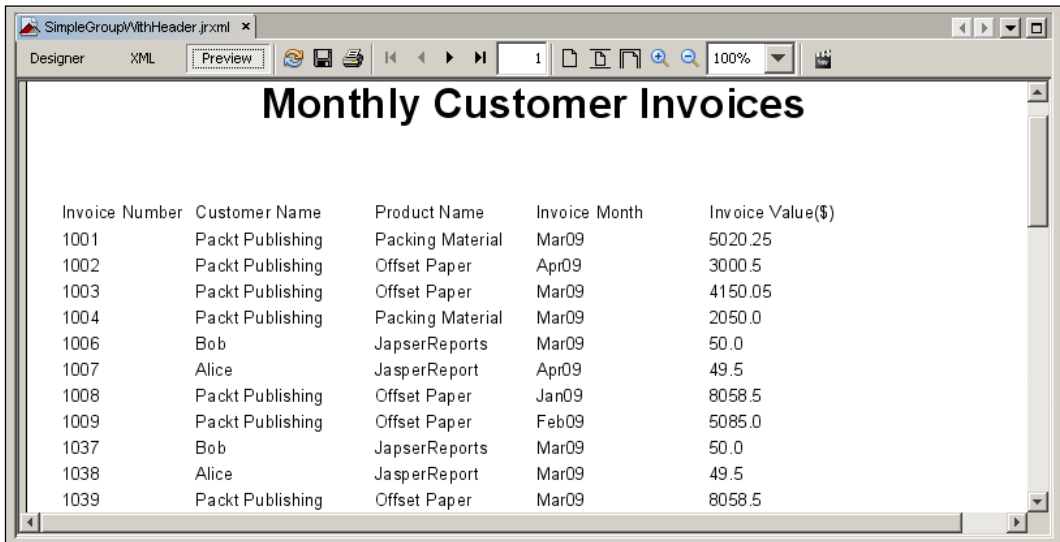
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb2` and copy sample data for this recipe into the database.

How to do it...

1. Open the `SimpleGroupWithHeader.jrxml` file from the `Task3` folder in the source code for this chapter. The **Designer** tab of iReport shows a report with a title **Monthly Customer Invoices**, as shown next:



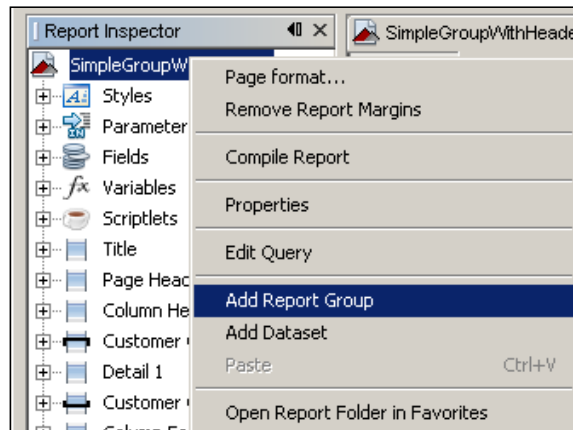
2. Switch to the **Preview** tab to see a current preview of your report. You will see a simple table with an arrangement of records, as shown in the following screenshot:



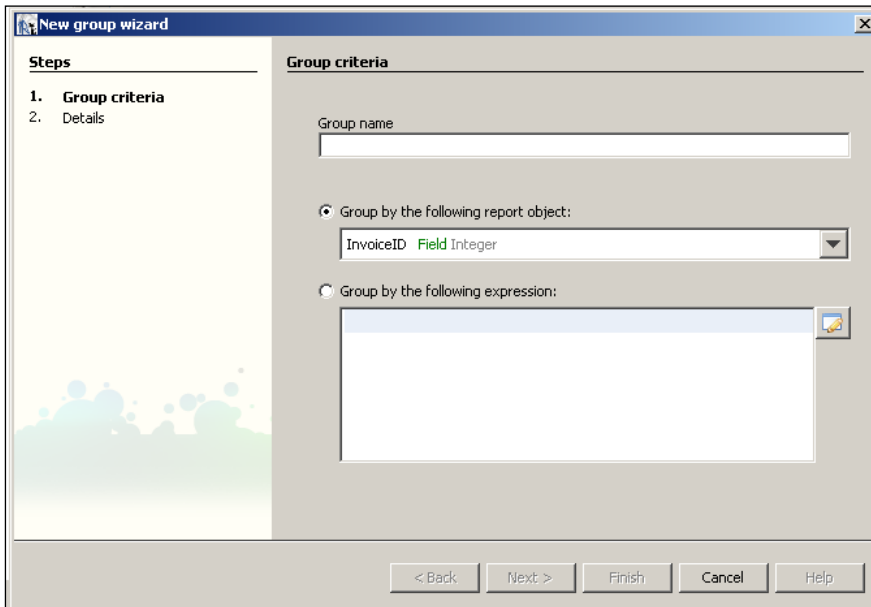
The screenshot shows a window titled 'SimpleGroupWithHeader.jrxml' with tabs for 'Designer', 'XML', and 'Preview'. The 'Preview' tab is active, displaying a report titled 'Monthly Customer Invoices'. The report contains a table with the following data:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1006	Bob	JapserReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1037	Bob	JapserReports	Mar09	50.0
1038	Alice	JasperReport	Mar09	49.5
1039	Packt Publishing	Offset Paper	Mar09	8058.5

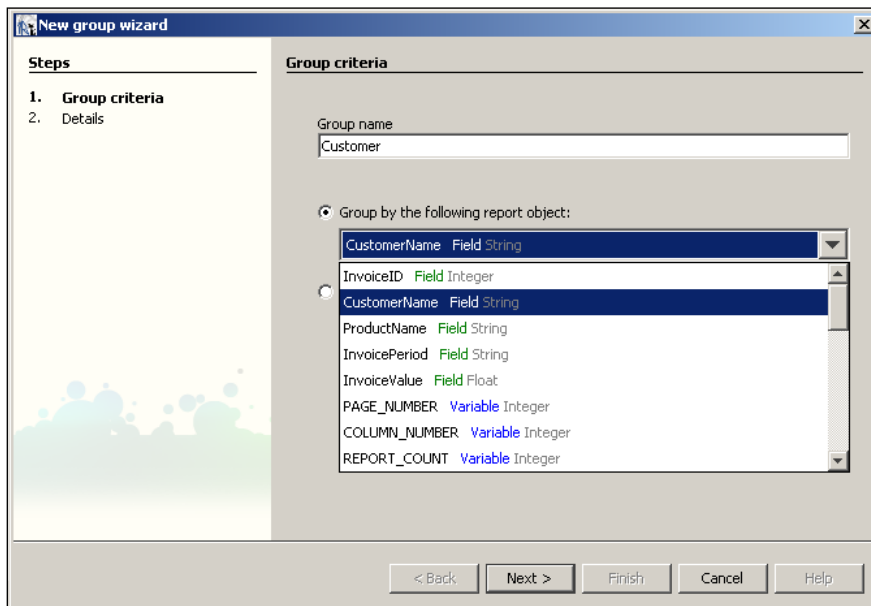
3. Switch to the **Designer** tab. Right-click on the top-most element `SimpleGroupWithHeader` in the **Report Inspector** window (on the left side of the **Designer** tab). A pop-up menu will appear. Click on **Add Report Group** option in the pop-up menu, as shown in the following screenshot:



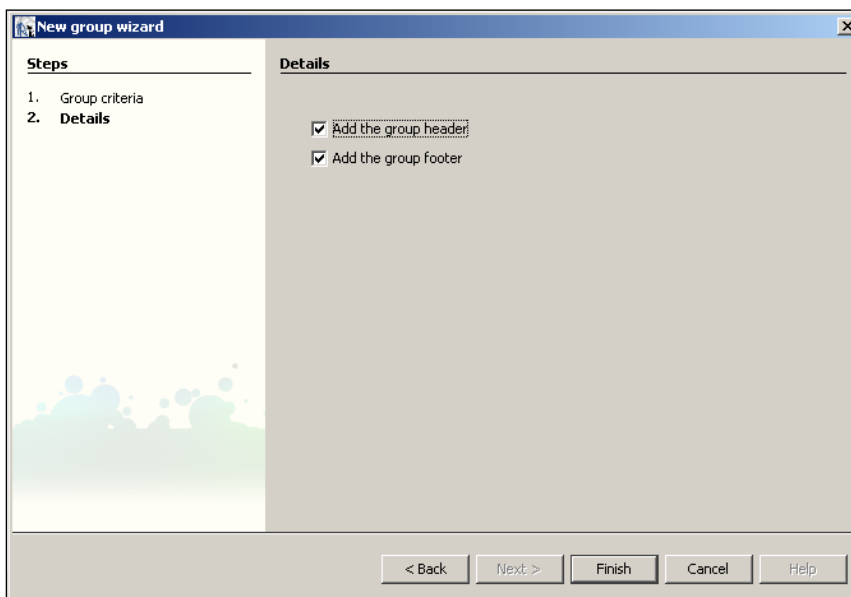
4. A **New group wizard** dialog will appear as shown follows:



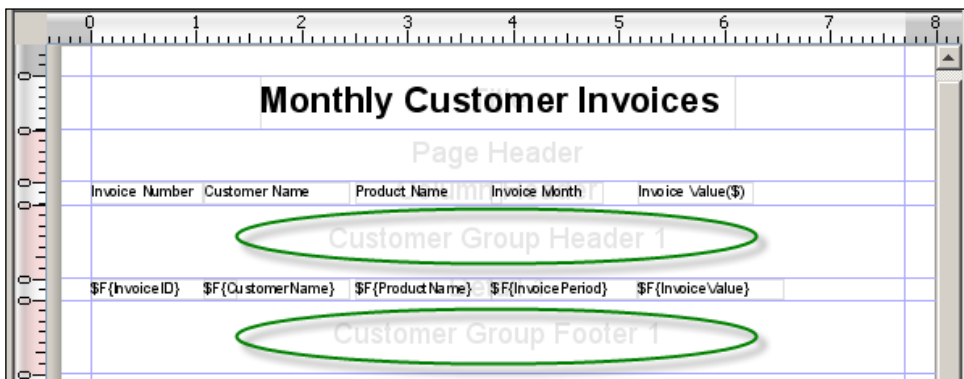
5. Type `Customer` in the **Group name** field and select **CustomerName** in **Group by the following report object** drop-down list, as shown in the following screenshot:



- Press the **Next** button, the dialog will change as shown in the following screenshot:



- Press the **Finish** button to dismiss the dialog. You will notice **Customer Group Header 1** and **Customer Group Footer 1** sections have been added into your report, as shown in the following screenshot:




8. Switch to the **Preview** tab to see the preview of your report. You will see clear grouping of records based on customer names separated by wide spaces, as shown in the following screenshot:

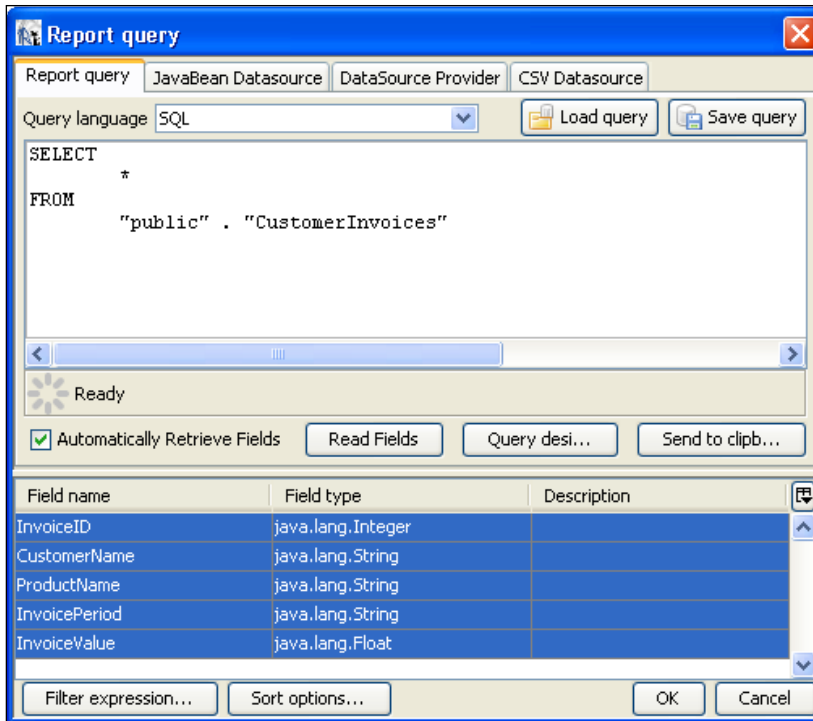
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1006	Bob	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5

9. Observe each page of your report and concentrate on customer names. You will notice that there are a number of groups for each customer spread randomly throughout the report, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1021	Packt Publishing	Binding Material	Jan09	2500.0
1022	Packt Publishing	Binding Material	Feb09	4808.0
1023	Packt Publishing	Packing Material	Jun09	5870.0
1024	Packt Publishing	Binding Material	Jan09	8770.0
1043	Packt Publishing	Binding Material	Mar09	6522.0
1025	Packt Publishing	Printing Ink	Aug09	5882.0
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1026	Packt Publishing	Binding Material	Sep09	4550.0
1027	Packt Publishing	Printing Ink	Oct09	4440.0
1044	Packt Publishing	Printing Film	Mar09	8999.0
1028	Packt Publishing	Offset Paper	May09	8660.0
1029	Packt Publishing	Offset Paper	Mar09	6522.0

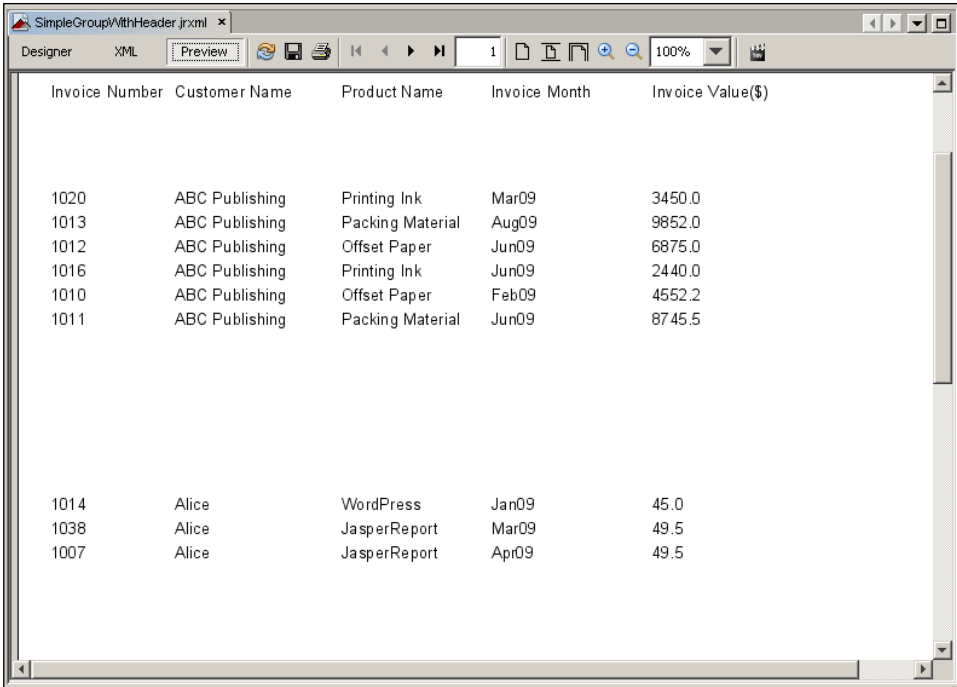
Packt Publishing shown grouped at more than one place in the report

10. Switch to the **Designer** tab. Press the **Report query** button on top of the report window (on right side of the **Preview** tab). Unfortunately, it is a bit difficult to find this button in iReport 3.6.0, as it shows no tool tip when you bring your mouse over it. However, you can find it by its icon. It is similar in shape to a standard database icon . As you click on this button, a **Report query** dialog will appear as shown in the following screenshot:



11. While your cursor is in query editor, press *Enter* key to bring it in the next blank line and type Order By 'CustomerName'. Click on **OK** button to dismiss the dialog.

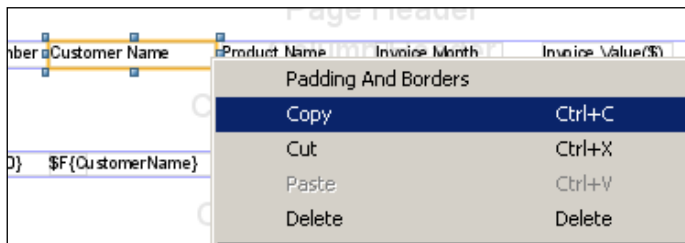
12. Switch to the **Preview** tab. Navigate through all pages of your report, focusing on grouping of customer names. This time you will notice that there is only one group for each customer name (all the records for a single customer are displayed in a single group), as shown in the following screenshot:



The screenshot shows the JasperReports Preview tab for a report named 'SimpleGroupWithHeader.jrxml'. The report is displayed in a table format with the following columns: Invoice Number, Customer Name, Product Name, Invoice Month, and Invoice Value(\$). The data is grouped by Customer Name.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1013	ABC Publishing	Packing Material	Aug09	9852.0
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1014	Alice	WordPress	Jan09	45.0
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5

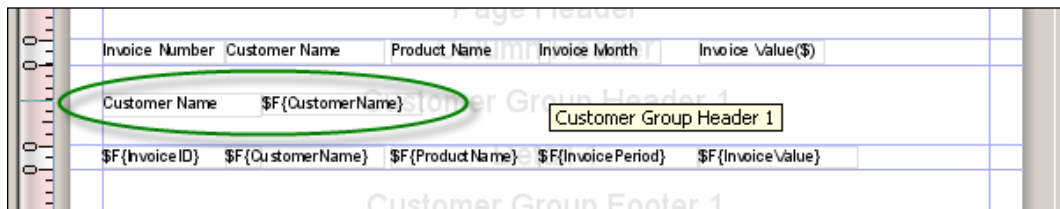
13. Switch to the **Designer** tab. Select **Customer Name** static text from the **Column Header** section of your report and right-click on it. A pop-up menu will appear. Select **Copy** option from the pop-up menu, as shown.



14. Right-click anywhere on whitespace in the **Customer Group Header 1** section of your report. A pop-up menu will appear. Select **Paste** option from the pop-up menu. A copy of customer name static text component will be generated in the section. Align the static text component to middle-left position in the **Customer Group Header 1** section, as shown in the following screenshot:



15. Select **Customer Name** text field component from **Detail 1** section of your report and right-click on it. A pop-up menu will appear. Select the **Copy** option from the pop-up menu.
16. Go to the **Customer Group Header 1** section of your report. Right-click anywhere on the whitespace of the section. A pop-up menu will appear. Select the **Paste** option from the menu. A copy of the customer name text field component will be generated in the **Customer Group Header 1** section. Align the text field component to the middle-left position in the **Customer Group Header 1** section beside the static text component of step 14, which you already have middle-left aligned in this section, as shown in the following screenshot:



17. Switch to the **Preview** tab. Navigate through all the pages of your report. You will notice a group heading for each customer, as shown in the following screenshot:

The screenshot shows the JasperReports Preview window for a report named 'SimpleGroupWithHeader.jrxml'. The report is displayed in the 'Preview' tab. It features a table with five columns: Invoice Number, Customer Name, Product Name, Invoice Month, and Invoice Value(\$). The data is grouped by Customer Name, with two distinct groups: 'ABC Publishing' and 'Alice'. Each group is preceded by a section header 'Customer Name' followed by the customer name. The 'ABC Publishing' group contains five records, and the 'Alice' group contains three records.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name ABC Publishing				
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1013	ABC Publishing	Packing Material	Aug09	9852.0
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
Customer Name Alice				
1014	Alice	WordPress	Jan09	45.0
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5

How it works...

In this recipe you have learned how to use the grouping feature. This is a useful feature, as it allows you to group records by dynamically inserting new sections in your report. Recall from step 5 that you chose `CustomerName` from the **Group by the following report object** drop-down list. When you did this, iReport created the group based on customer names. The result of this grouping is shown in the preview given in step 8, where you can see that wide spaces are inserted whenever JasperReports detects that a customer name has changed.

Another important point is that the grouping feature does not perform ordering of records. It only inserts a section (for example, whitespaces in the preview of step 8) when a group changes (that is, the customer name changes).

Therefore, you have to provide records in order form for grouping to work properly. That's SQL's job. Although SQL-related discussions are beyond the scope of this book, I have included an `Order by` clause in step 11 to demonstrate how you will use SQL to order your records so that JasperReports can group them correctly.

Using parameters to filter records during report processing

JasperReports allows you to filter records during report processing and show only those of interest in a particular report. This recipe teaches you how to use this feature of filtering of records during report processing.

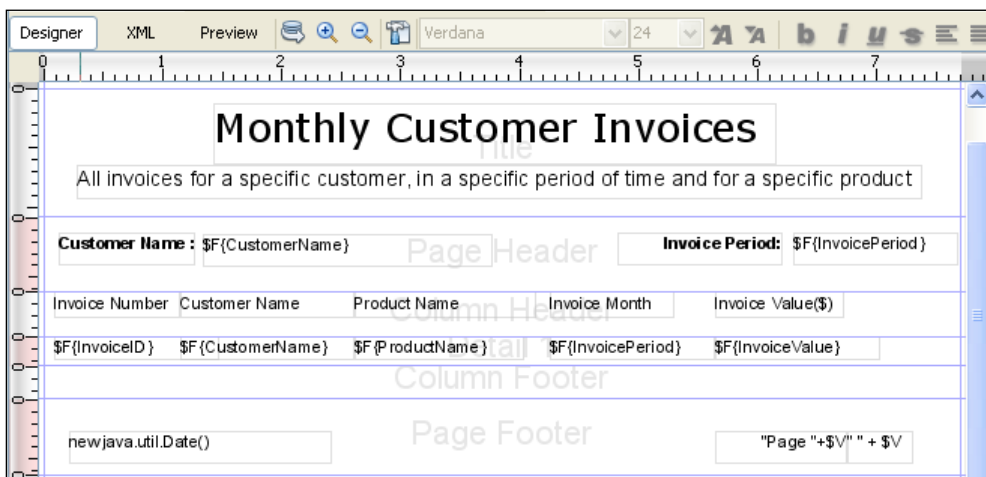
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb2` and copy sample data for this recipe into the database.

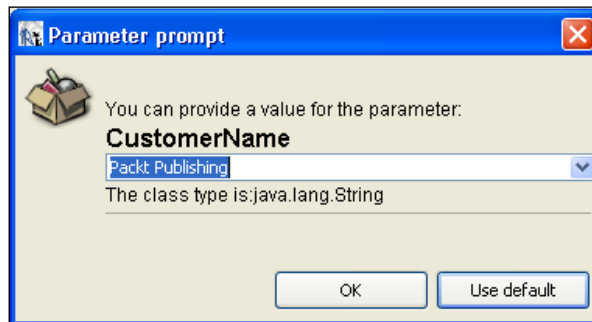
How to do it...

Following simple steps demonstrate how you can filter records to be shown in a report:

1. Open the `FilteringReportRecords.jrxml` file from the `Task4` folder of the source code for this chapter. The **Designer** tab of iReport shows a report with a simple table in its body, as shown in the following screenshot:



- Switch to the **Preview** tab, and **Parameter prompt** dialogs will appear, which will ask you for a CustomerName and InvoicePeriod parameters, as shown in the following screenshot:

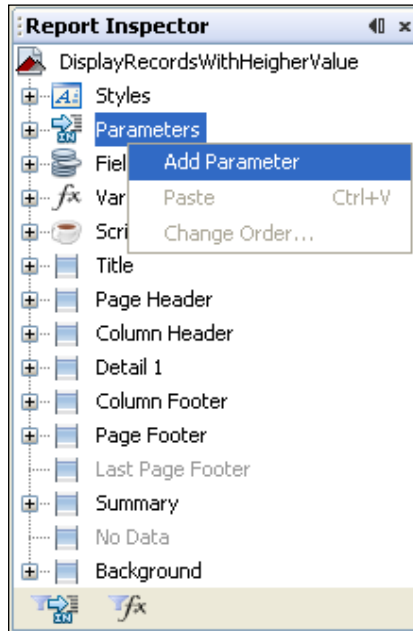


- Type Packt Publishing and Mar09 as the value for the customer name and invoice month parameters, respectively. You will see a report of invoices for the particular customer for a specific invoicing month, as follows:

Monthly Customer Invoices				
All invoices for a specific customer, in a specific period of time and for a specific product				
Customer Name : Packt Publishing			Invoice Period: Mar09	
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1042	Packt Publishing	Printing Ink	Mar09	8500.0
1043	Packt Publishing	Binding Material	Mar09	6522.0
1044	Packt Publishing	Printing Film	Mar09	8999.0
1032	Packt Publishing	Packing Material	Mar09	5020.25
1047	Packt Publishing	Positive Plates	Mar09	7532.0
1049	Packt Publishing	Developer	Mar09	1589.0
1050	Packt Publishing	Developer	Mar09	4896.0

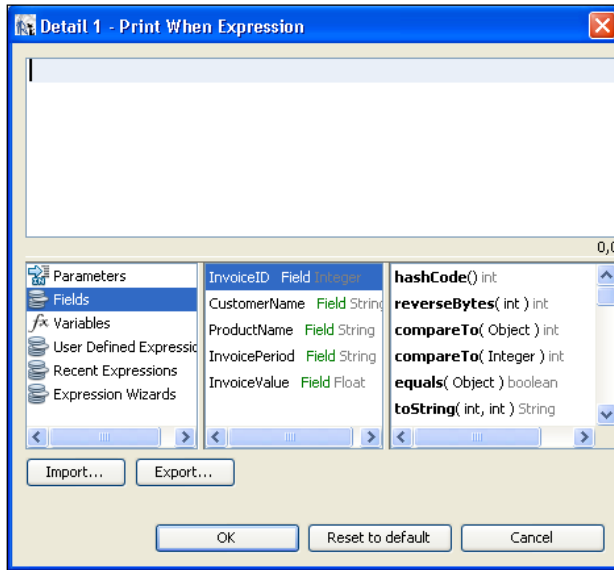
Saturday 31 October 2009 Page 1 of 1

4. Switch back to the **Designer** tab. Right-click on the **Parameters** node in the **Report Inspector** window on the left of the **Designer** tab, as shown next. A pop-up menu will appear. Choose the **Add Parameter** option from the pop-up menu.

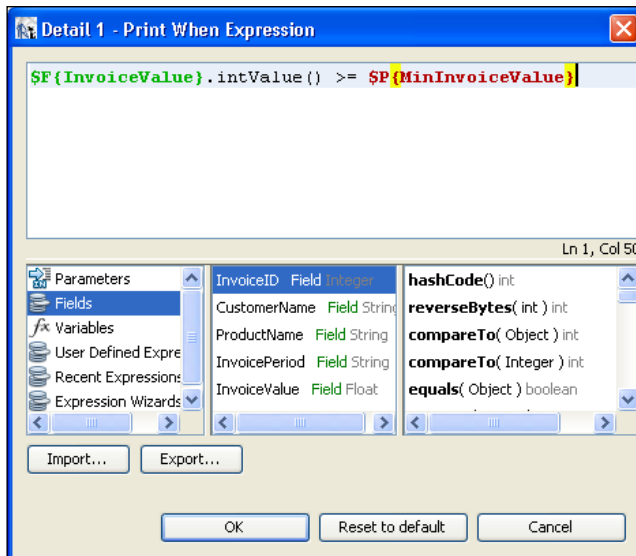


5. The **Parameters** node will expand to show the newly added parameter named `parameter1` at the end of the parameters list. Select `parameter1` from the list; its properties will appear in the **Properties** window below the palette of components on the right of your iReport main window.
6. Click on the **Name** property of the parameter and type `MinInvoiceValue` as its value. Now click the **Parameter Class** property and change its value to `java.lang.Integer`. Leave the rest of the parameter properties at their default values.

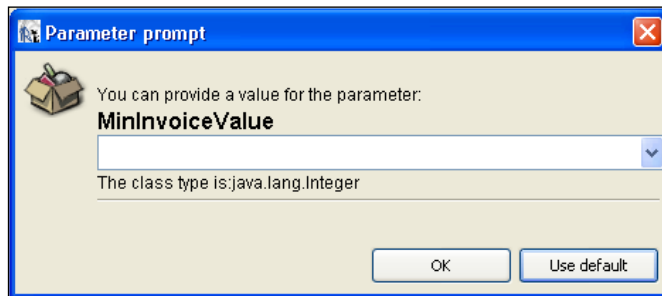
7. Click inside the Detail 1 section; its properties will appear below the palette of components. Click the button beside the **Print When Expression** property. A **Print When Expression** window will open, as shown in the following screenshot:



8. Add the expression text `$F{InvoiceValue}.intValue() >= $P{MinInvoiceValue}` in the editor window and click the **OK** button, as shown in the following screenshot:



- Switch to the **Preview** tab; **Parameter prompt** dialogs will appear, which will ask you for a customer name, invoicing month, and minInvoice value parameters, as shown in the following screenshot:



- Type Packt Publishing, Mar09, and 5000 as values for the customer name, invoicing month, and min invoice value parameters, respectively and click **OK**. You will see a report of invoices whose values are greater than 5000 for the particular customer for the specific invoicing month. Note that this report is a subset of the report shown in the preview of step 3.

Monthly Customer Invoices				
All invoices for a specific customer, in a specific period of time and for a specific product				
Customer Name : Packt Publishing			Invoice Period: Mar09	
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1042	Packt Publishing	Printing Ink	Mar09	8500.0
1043	Packt Publishing	Binding Material	Mar09	6522.0
1044	Packt Publishing	Printing Film	Mar09	8999.0
1032	Packt Publishing	Packing Material	Mar09	5020.25
1047	Packt Publishing	Positive Plates	Mar09	7532.0
<div>Saturday 31 October 2009</div> <div>Page 1 of 1</div>				

How it works...

You have used a combination of parameters and **Print When expression** features of JasperReports in this recipe. You use parameters to allow users of your report to enter data just before report processing. For example, you included a parameter named **MinInvoiceValue** in step 6 of the recipe.

Then you used the **MinInvoiceValue** parameter in the `$F{InvoiceValue}.intValue() >= $P{MinInvoiceValue}` expression in step 8. This expression prints a record only when the **InvoiceValue** field in a record is greater than or equal to the value that a user enters as the **MinInvoiceValue** parameter. You can see this by comparing the previews of steps 3 and 10.

Implementing groups within groups — a nested hierarchy

JasperReports allows you to implement a nested hierarchy of groups in a report. A hierarchy means that a group will contain subgroups.

Subgroups are useful when you want to design a report with multiple levels of logical grouping. For example, a report showing all customer invoices for a particular month may contain a large number of records. You can segregate the records with customer names; that is, all invoices of a particular customer will be grouped together. This is one level of logical grouping.

Now you can introduce another level of logical grouping by further segregating all invoices of a particular customer into subgroups based on products sold to a customer. This will become the second level of logical grouping.

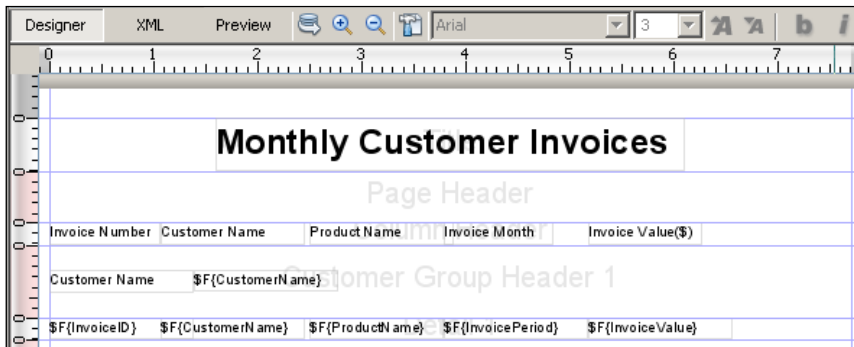
This recipe demonstrates how to design a report with two levels of logical grouping.

Getting ready

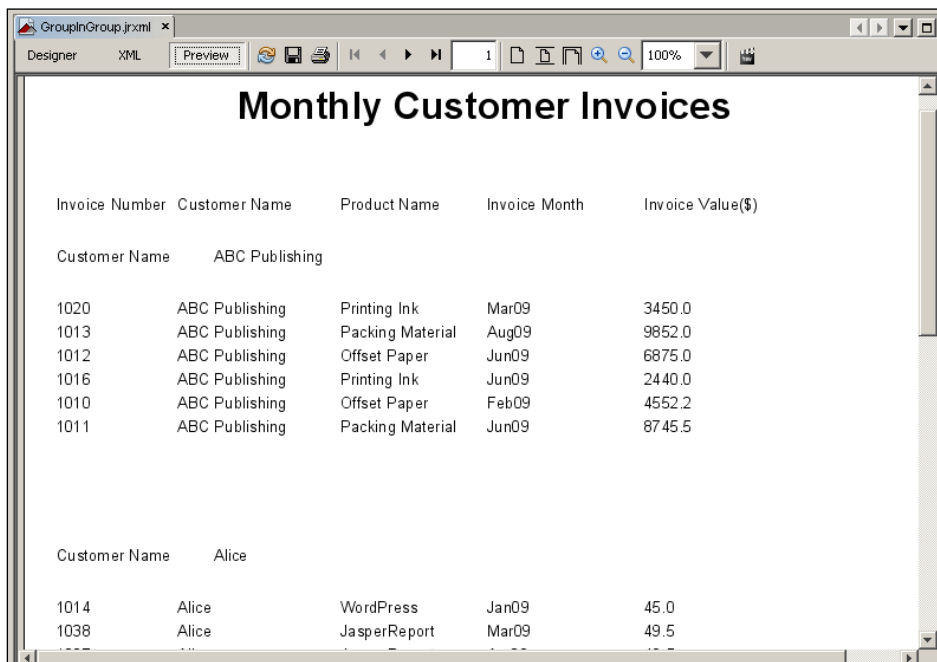
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb2` and copy sample data for this recipe into the database.

How to do it...

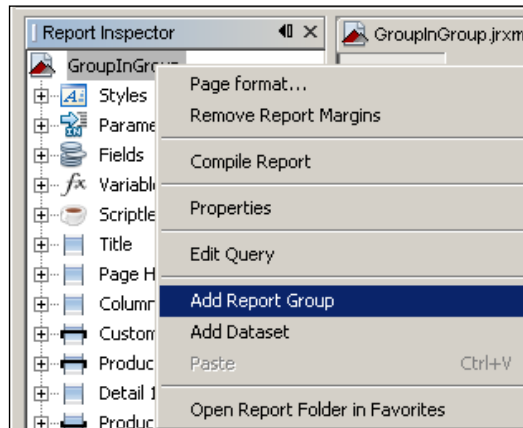
1. Open the `GroupInGroup.jrxml` file from the `Task5` folder in the source code for this chapter. (This is the final form of the recipe *Inserting a heading for a group of records* of this chapter.) The **Designer** tab of iReport shows a report with a main title **Monthly Customer Invoices**, as shown in the following screenshot:



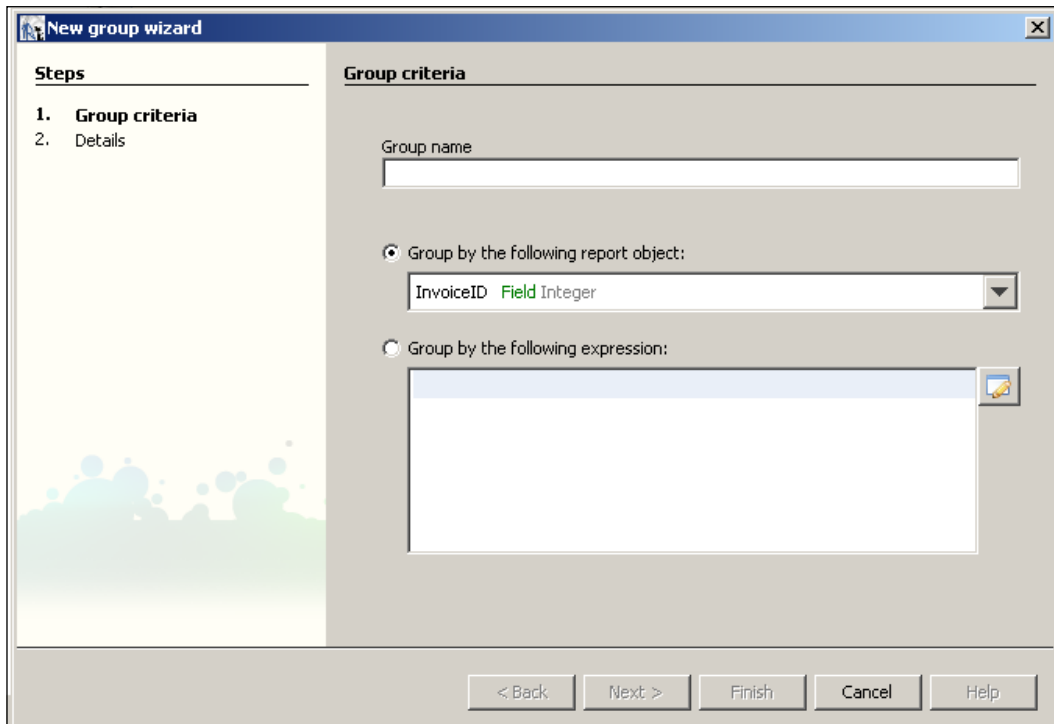
2. Switch to the **Preview** tab to see the current preview of your report. You will see a table of records, which are grouped with customer names. The heading of each customer group is displayed before the start of each group. You will also notice that the records within a group are not arranged or grouped.



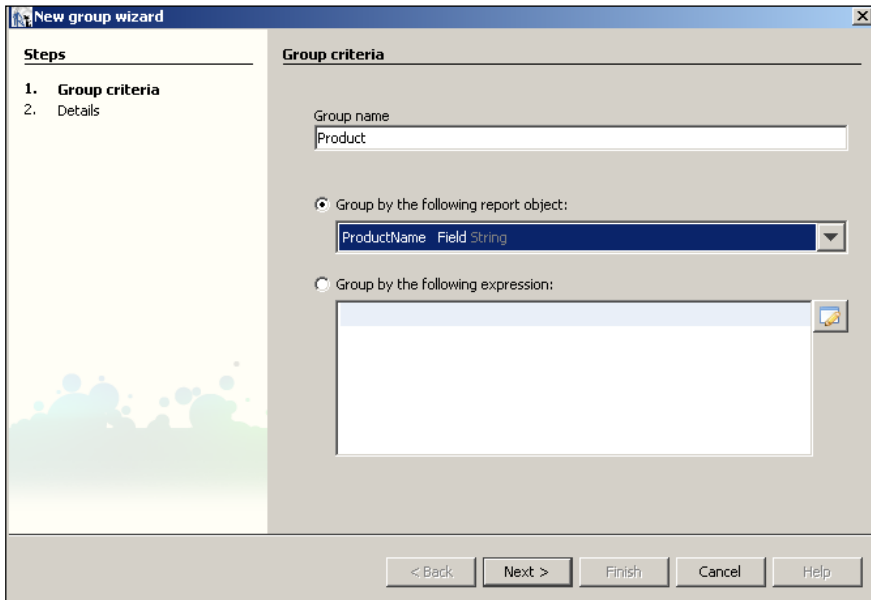
- Now we are about to arrange records within a group using a nested group. Switch to the **Designer** tab. Right-click on the top-most top element **GroupInGroup** in the **Report Inspector**. A pop-up menu will appear. Select the **Add Report Group** option from the pop-up menu, as shown in the following screenshot:



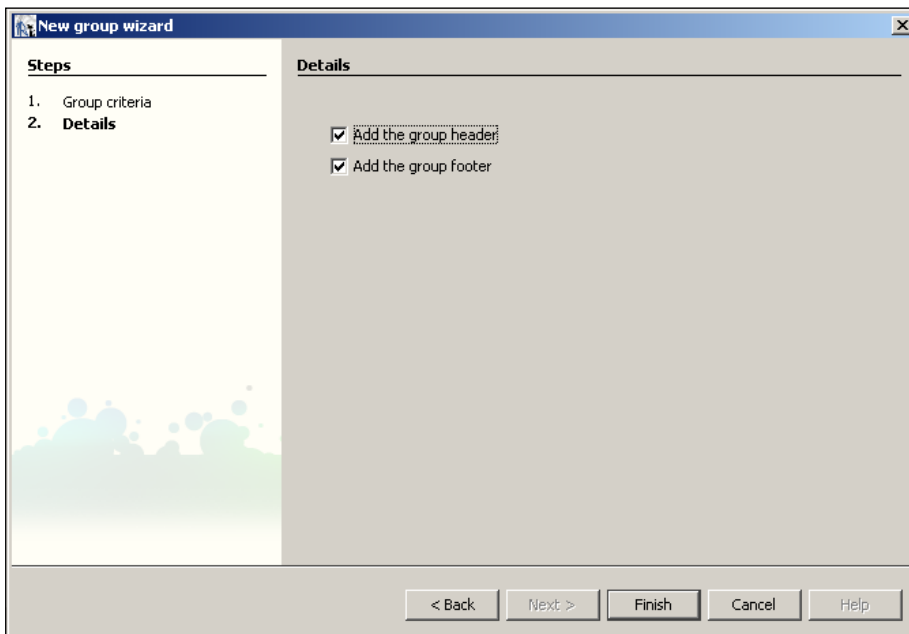
- A **New group wizard** dialog will appear, as shown in the following screenshot:



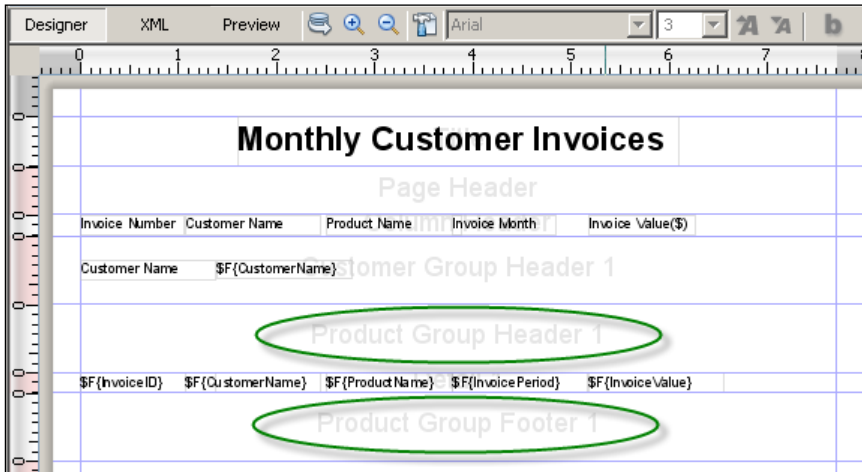
5. Type `Product` in the **Group name** and select **ProductName Field** in the **Group by the following report object:** option, as shown in the following screenshot:



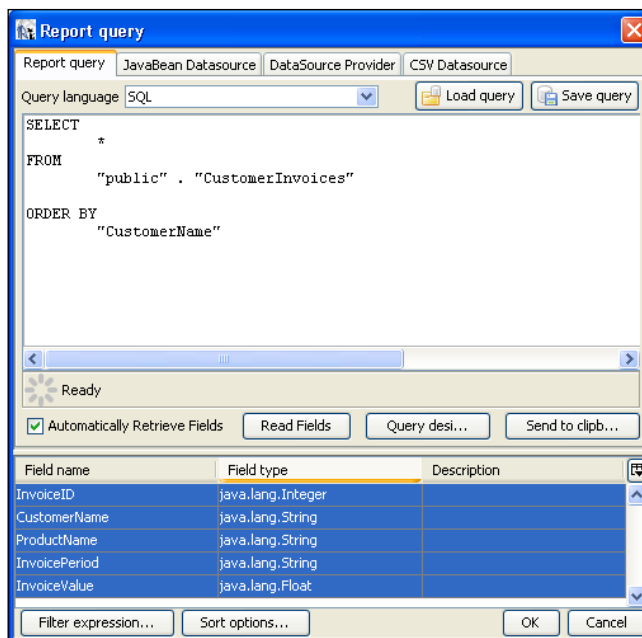
6. Click the **Next** button, and the dialog will change, as in the following screenshot:



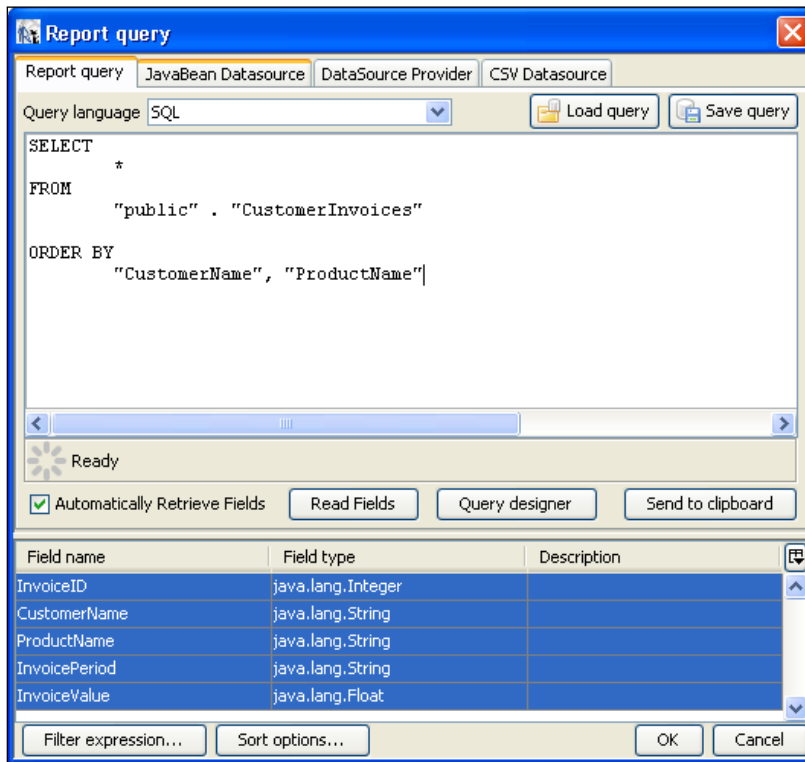
- Click the **Finish** button to dismiss the dialog. You will notice that **Product Group Header 1** and **Product Group Footer 1** sections have been added into your report, as shown in the following screenshot:



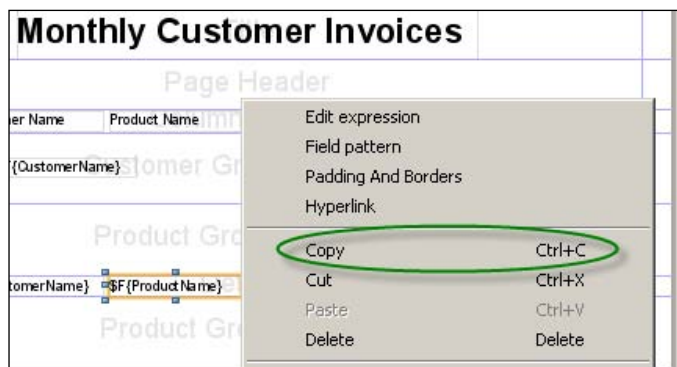
- Click the **Report query** button at the top of the report window (on the right side of the **Preview** tab). You can find it by its icon. It is similar in shape to a standard database icon. As you click on this button, a **Report query** dialog will appear, as shown in the following screenshot:



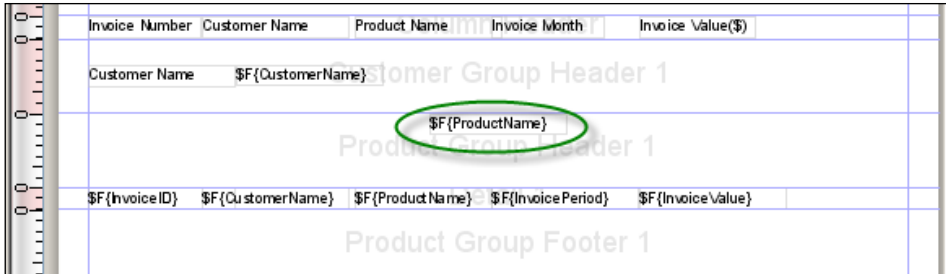
9. In the query editor bring your cursor to the end of the text ORDER BY "CustomerName" and type , "ProductName". Click the **OK** button to dismiss the dialog, as shown in the following screenshot:



10. Select the `ProductName` text field component from the Detail 1 section of your report and right-click on it and select the **Copy** option from the menu, as shown in the following screenshot:



11. Go to the **Product Group Header 1** section of your report. Right-click anywhere on the whitespace of the section and select the paste option from the menu. A copy of the ProductName text field component will be generated in the section. Align the text field component to the top-middle position in the **Product Group Header 1** section, as shown in the following screenshot:



12. Switch to the **Preview** tab to see a preview of your report. You will see a grouping of records based on customer names and inside the customer group a grouping based on product names. A heading for each product group is also used for distinction, as shown in the following screenshot:



13. Navigate through all the pages. You will notice that all product names are grouped under the main grouping of customer names, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name Packt Publishing				
Binding Material				
1021	Packt Publishing	Binding Material	Jan09	2580.0
1026	Packt Publishing	Binding Material	Sep09	4550.0
1022	Packt Publishing	Binding Material	Feb09	4808.0
1024	Packt Publishing	Binding Material	Jan09	8770.0
1043	Packt Publishing	Binding Material	Mar09	6522.0
Developer				
1050	Packt Publishing	Developer	Mar09	4896.0
1049	Packt Publishing	Developer	Mar09	1589.0
Offset Paper				
1033	Packt Publishing	Offset Paper	Jan09	3000.5
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1028	Packt Publishing	Offset Paper	May09	8660.0
1029	Packt Publishing	Offset Paper	May09	8660.0
1034	Packt Publishing	Offset Paper	Jun09	null
1035	Packt Publishing	Offset Paper	Aug09	3000.5
1036	Packt Publishing	Offset Paper	Sep09	3000.5

How it works...

The nested group that you formed in this recipe is similar to the group described in the *Inserting a heading for a group of records* recipe of this chapter. The only difference is that the nested group formed in this recipe is based on product names, whereas grouping in the original (*Inserting a heading for a group of records*) recipe is based on customer names. This recipe actually fits product name grouping as a nested group inside customer name grouping.

You can learn how the grouping works from the earlier *Inserting a heading for a group of records* recipe.

Adding a simple footer to your report

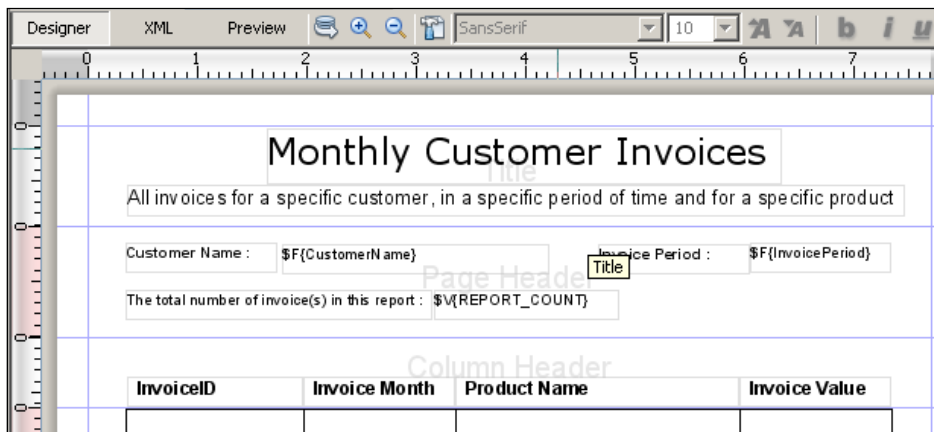
Almost every report has a footer. The simplest footer will provide information about the report such as the name of the report and the date and time when the report was generated among others.

This recipe teaches you how you will display the name of the report and the date when the report was generated.

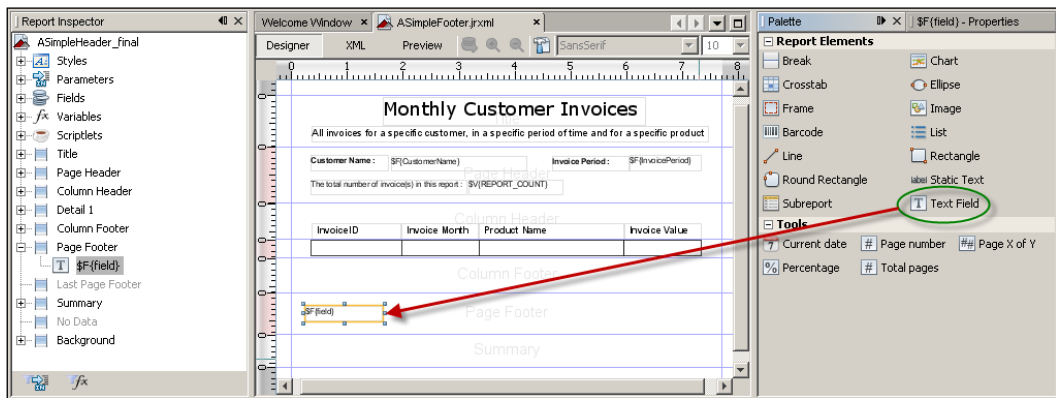
How to do it...

The steps to a simple footer design are as follows:

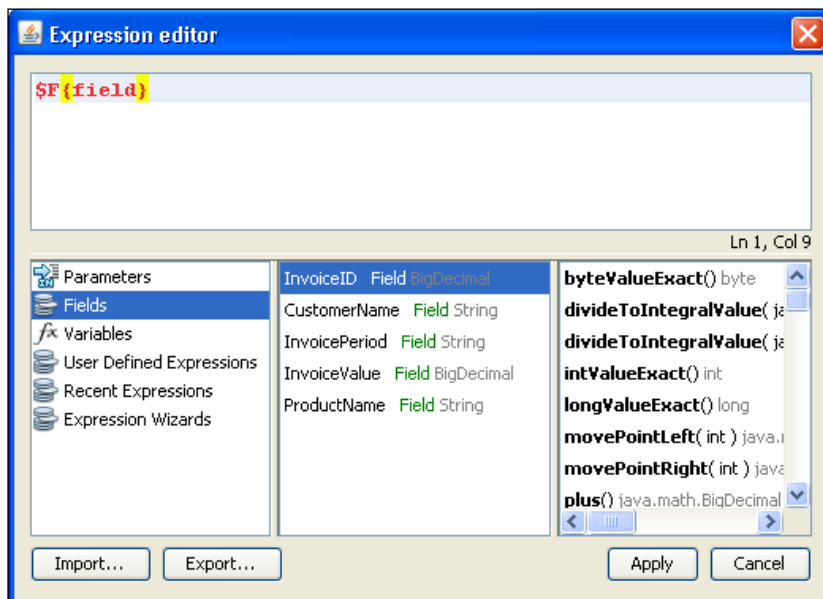
1. Open the `ASimpleFooter.jrxml` file from the `Task6` folder of the source code for this chapter. The **Designer** tab of iReport shows that the report contains a main title, a subtitle, a page header, a column header, and an empty footer, as shown in the following screenshot:



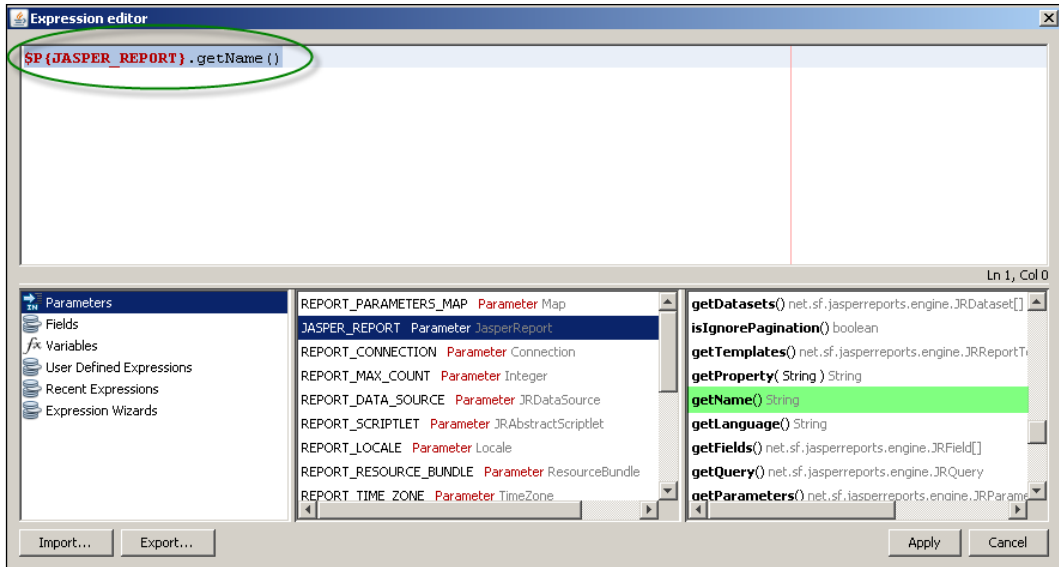
- Now you are all set to insert a simple footer into your report. A palette of components is available on the right of the main iReport window. Select the text field component from the palette and drag it into the Page Footer section in the **Designer** tab.



- Right-click on the text field component of step 2 and select the **Edit expression** option from the pop-up menu. An **Expression editor** window will open, as shown in the following screenshot:

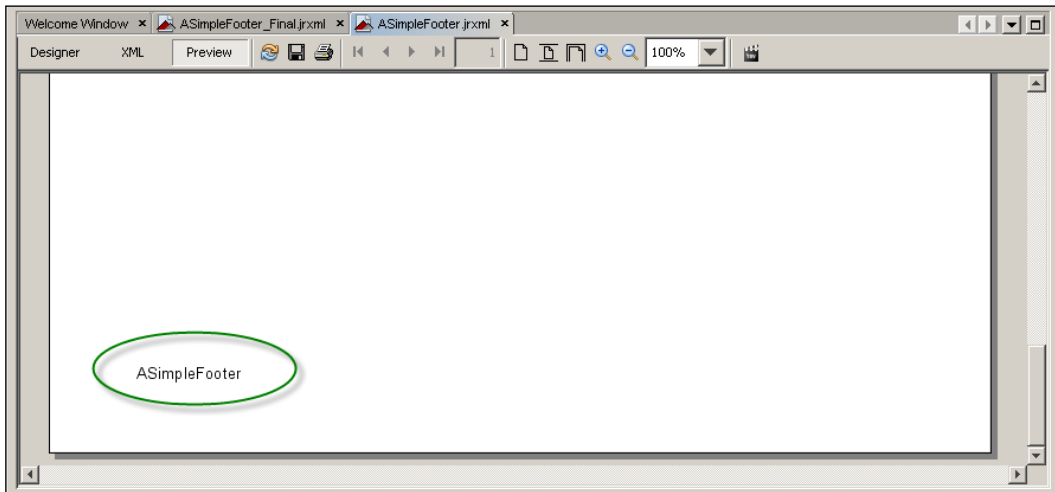


4. Delete the default expression (`$F{field}`) from the **Expression editor** window.
5. Click on **Parameters** in the first column of the lower half of the **Expression editor** window. This will list all the parameters in the adjacent second column. Scroll down the list of parameters in the second column to find the `JASPER_REPORT` parameter and select it. This will list all methods for this parameter in the adjacent third column. Scroll down the list of methods in the third column to find the `getName()` method. Double-click on it. This will set `$P{JASPER_REPORT}.getName()` in the **Expression editor**, as shown in the following screenshot:

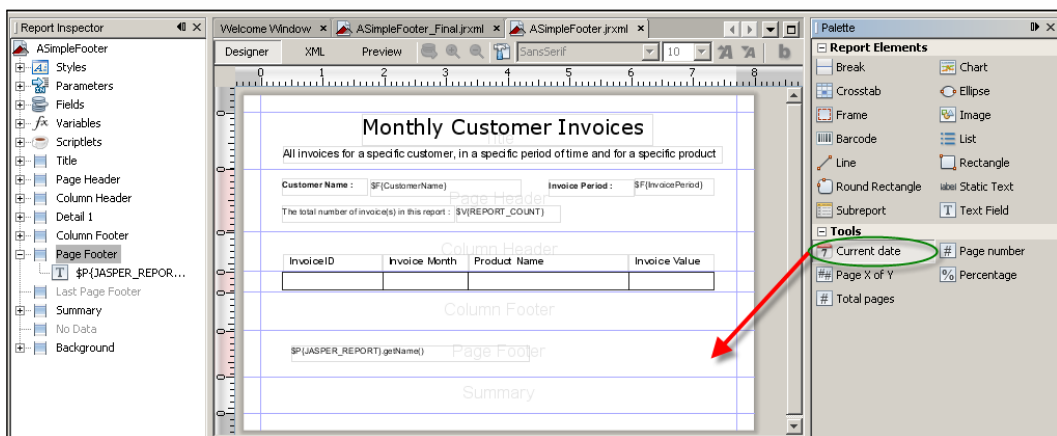


6. Click the **Apply** button. This will attach the `$P{JASPER_REPORT}.getName()` expression to your text field component of step 2.

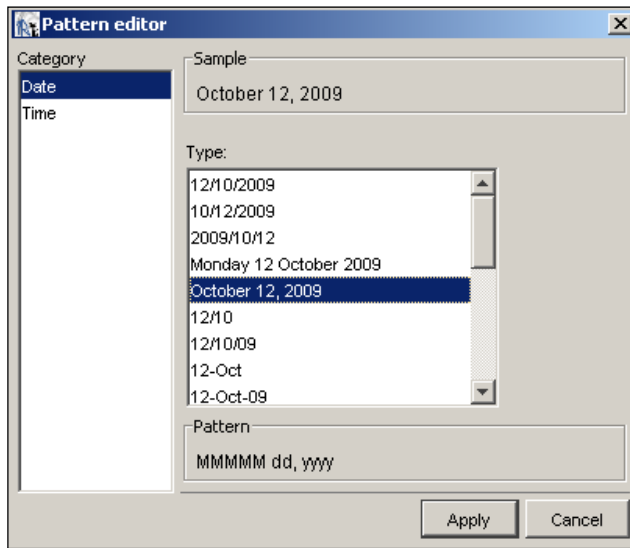
- Switch to the **Preview** tab and scroll the vertical scroll bar down until you see the footer of your report at the bottom, as shown in the following screenshot:



- Switch back to the **Designer** tab for more editing in the Page Footer section.
- Now you are about to insert the current date into your report. Drag a new **Current date** from the **Tools** section of the palette and drop it into the Page Footer section of your report, as shown in the following screenshot:



10. A **Pattern editor** dialog will appear, as shown next. Select **Date** in the **Category** column and the appropriate date format from the **Type** column and click the **Apply** button.



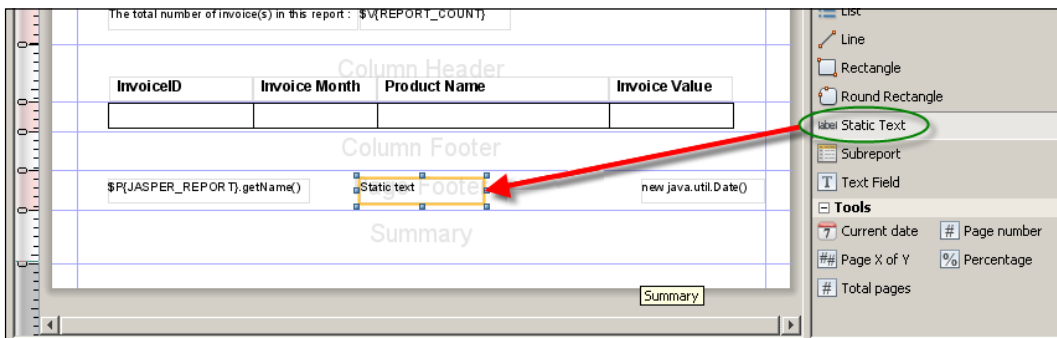
11. A text field with the new `java.util.Date()` expression will appear in the Page Footer section of your report. Select this date text field component and use the arrow keys to roughly align this text field horizontally and vertically to the right-middle position in the Page Footer section of your report, as shown in the following screenshot:

Monthly Customer Invoices			
All invoices for a specific customer, in a specific period of time and for a specific product			
Customer Name :	<code>\${CustomerName}</code>		Invoice Period : <code>\${InvoicePeriod}</code>
The total number of invoice(s) in this report : <code>#{REPORT_COUNT}</code>			
InvoiceID	Invoice Month	Product Name	Invoice Value
<div> <div><code>#{JASPER_REPOR T}.getName()</code></div> <div><code>new java.util.Date()</code></div> </div>			

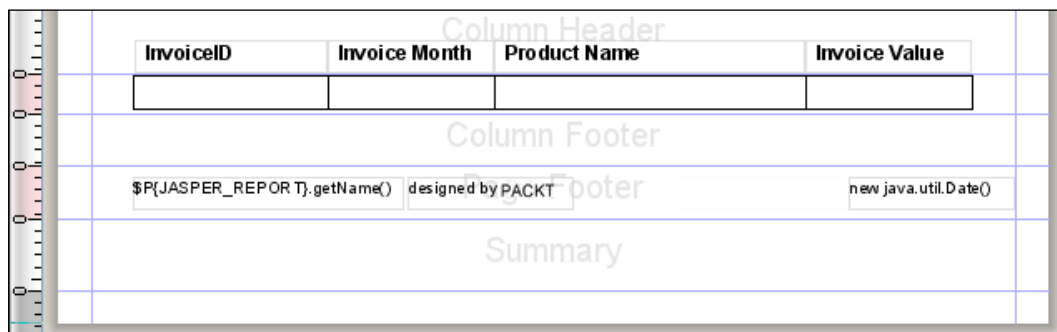
12. Switch to the **Preview** tab. You will see the current date on the bottom-right corner of your report, as shown in the following screenshot:



13. Switch back to the **Designer** tab. Select a static text component from the palette and drag it into the Page Footer section in the **Designer** tab.



14. Double-click and edit the static text component to add designed by PACKT text in it. Then select the static text component and use the arrow keys to select its position, as shown in the following screenshot:



15. Switch to the **Preview** tab, which shows what your report looks like.

Monthly Customer Invoices

All invoices for a specific customer, in a specific period of time and for a specific product

Customer Name : null

Invoice Period : null

The total number of invoice(s) in this report : 1

InvoiceID	Invoice Month	Product Name	Invoice Value

16. You have dropped three components (a text field, a date field, and a static text) into your report. Now you will combine the text field and static text components to improve the look of the Page Footer. Switch back to the **Designer** tab.
17. Double-click the static text component and copy the selected text (designed by PACKT) by pressing **Ctrl+C**. Then double-click on the text field component with the expression `$P{JASPER_REPORT}.getName()`. The text of expression will be selected. Deselect the text and move your cursor to the end of the expression. Type `+` and then a space, now press **Ctrl+V** to insert your copied static text (designed by PACKT). The expression in the text field component will change to `$P{JASPER_REPORT}.getName() + " designed by PACKT"`. Move your cursor to the end of expression and press the `"` key of your keyboard. Now your expression in the text field component will look like `$P{JASPER_REPORT}.getName() + " designed by PACKT"`.



21. You have successfully designed a simple footer for your report. Save it by selecting **Save** from the **File** menu.

How it works...

You have learned how to display the name of your report and the date when the report was generated in the footer. You have also learned the small trick of combining two components into one in step 17 of the recipe. Switch to the **XML** tab to see how all this works:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other JasperReports XML tags -->
  <pageFooter>
    <band height="32" splitType="Stretch">
      <textField>
        <reportElement x="25" y="7"
          width="279" height="20"
          isPrintInFirstWholeBand="true"/>
        <textElement/>
        <textFieldExpression class="java.lang.String">
          <![CDATA[${P{JASPER_REPORT}.getName()
            +" designed by PACKT"}]]>
        </textFieldExpression>
      </textField>
      <textField pattern="MMMM dd, yyyy">
        <reportElement x="446" y="7" width="100"
          height="20"/>
        <textElement/>
        <textFieldExpression class="java.util.Date">
          <![CDATA[new java.util.Date()]]>
        </textFieldExpression>
      </textField>
    </band>
  </pageFooter>
  <!-- other JasperReports XML tags -->
</jasperReport>
```

I have shown only a `<pageFooter>` tag (which corresponds to the Page Footer section of your report) omitting all the other tags, which are not relevant to this discussion.

The `<pageFooter>` tag has a `<band>` child, with two children, both named `<textField>`. The first `<textField>` tag corresponds to the name of the report that you dropped in step 2 of the recipe and later edited in step 17. Notice that the expression and text combination that you authored in step 17 (`$P{JASPER_REPORT}.getName() + " designed by PACKT"`) is part of the content inside the first `<textField>` tag. This means when you followed step 17, iReport copied the expression and text combination into the first `<textField>` tag.

The second `<textField>` tag simply corresponds to the Current date component that you dropped in step 9 of the recipe.

Also look at the `$P{JASPER_REPORT}.getName()` expression of step 5. JasperReports provides many other useful parameters like this `JASPER_REPORT` parameter, which holds information about your report (for example, the name of the report, the SQL query used in your report, and so on). You can follow steps 3, 4, and 5 to play with different available parameters and their methods to discover the different bits of information you can display in any text field.

Displaying general information or summary at the end of your report

You will often need to provide a brief summary or general information about your report on the last page. JasperReports provides you with a section named `Summary`, which is displayed before the footer on the last page of your report. For example, if you have a large report of customer invoices, you may display a product-wise summary of invoice totals in the `Summary` section.

This recipe shows how to display some general information such as the date and time of your report generation as well as the total value of all invoices at the end of your report.

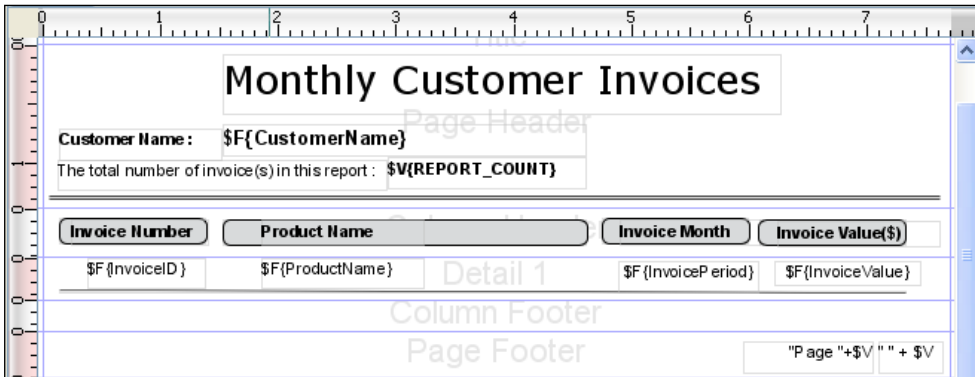
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb2` and copy sample data for this recipe into the database.

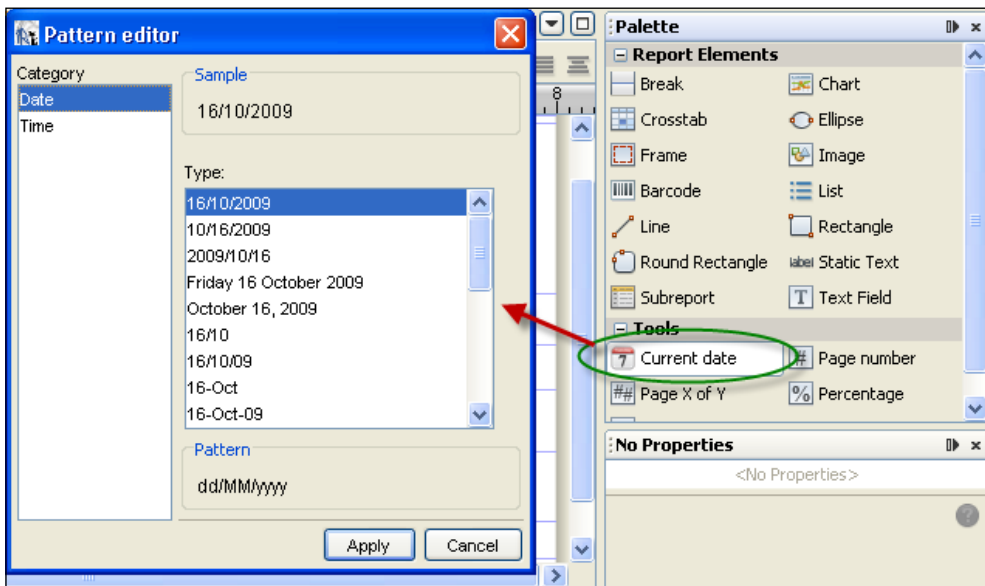
How to do it...

These steps show how to include a summary at the end of your report:

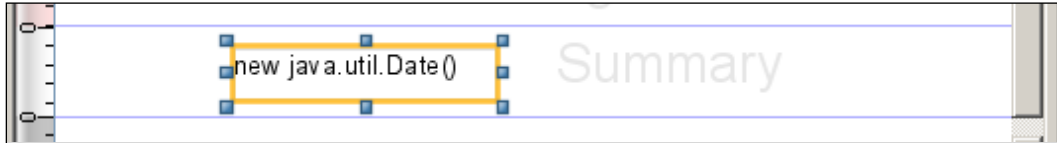
1. Open the `General_Summary.jrxml` file from the `Task7` folder in the source code for this chapter. The **Designer** tab of iReport shows a simple report, as shown in the following screenshot:



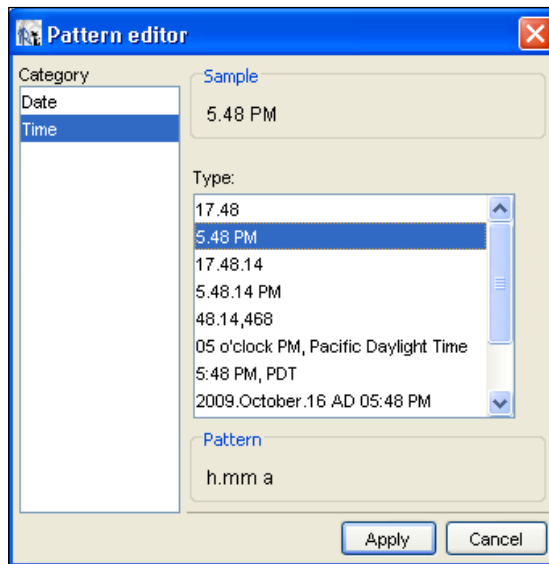
2. Drag the current date component from the Tools section of palette of the components and drop it into the Summary section of your report. A **Pattern editor** dialog will open, as shown in the following screenshot:



- The **Pattern editor** dialog has two sections. The section on the left is named **Category** and the one on the right is named **Type**. The **Category** section has two values, **Date** and **Time**. **Date** is selected by default. You can see different patterns for date in the **Type** section. Select a pattern for the date and press **Apply** button. This will insert a text field component with an expression `new java.util.Date()` into the Summary section of your report, as shown in the following screenshot:

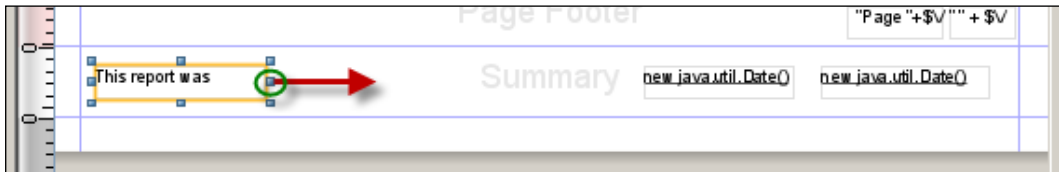


- Repeat steps 2 and 3 to insert another current date component. This time select **Time** from the **Category** section of the **Pattern editor** dialog and choose an appropriate pattern for the time from the **Type** section. This will insert another text field component with the same `new java.util.Date()` expression.

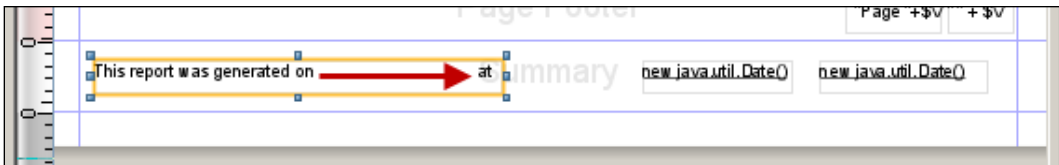


- Select a static text component from the palette; drag-and-drop it into the summary section of your report.

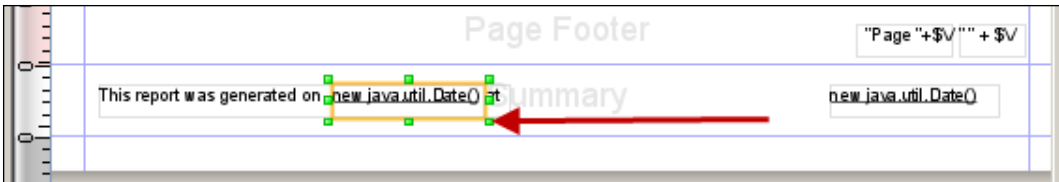
6. Double-click and edit the static text component to enter This report was generated on at as its value. The text will disappear from the view area, as it is longer than the length of the static text component. Enlarge the rectangular boundary of the static text component by dragging one of its corners, as shown in the following screenshot:



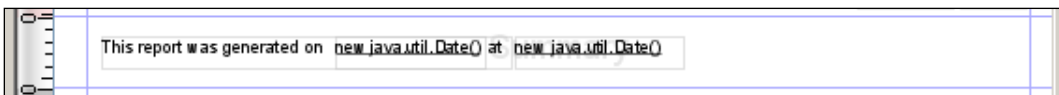
7. Insert whitespace between the words on and at of the static text component by pressing the space bar 35 times, as shown in the following screenshot:



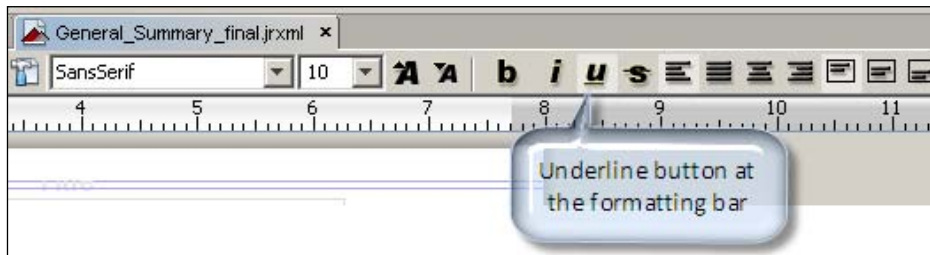
8. Select the date text field component of step 3. Use the arrow keys to move the date text field component until it sits in the space between the words on and at, as shown in the following screenshot:



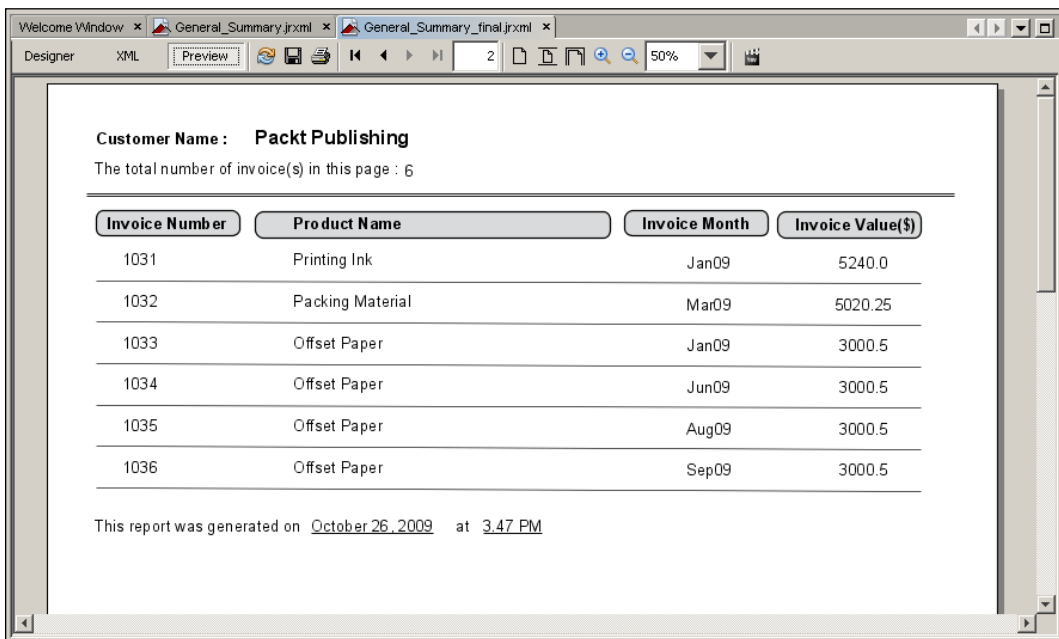
9. Now select the time text field component of step 4. Move the time text field component until it sits just next to the static text component, as shown in the following screenshot:



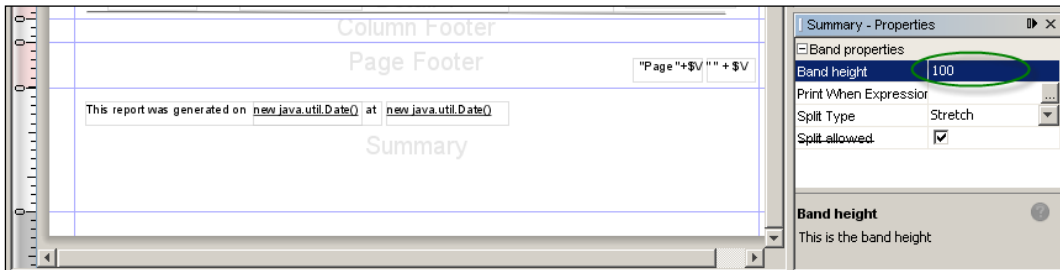
10. Underline the date and time text field components by selecting each component while holding down the **Ctrl** key and then (after releasing the **Ctrl** key) pressing the **Underline** button from the formatting bar of the **Designer** tab in your report window.



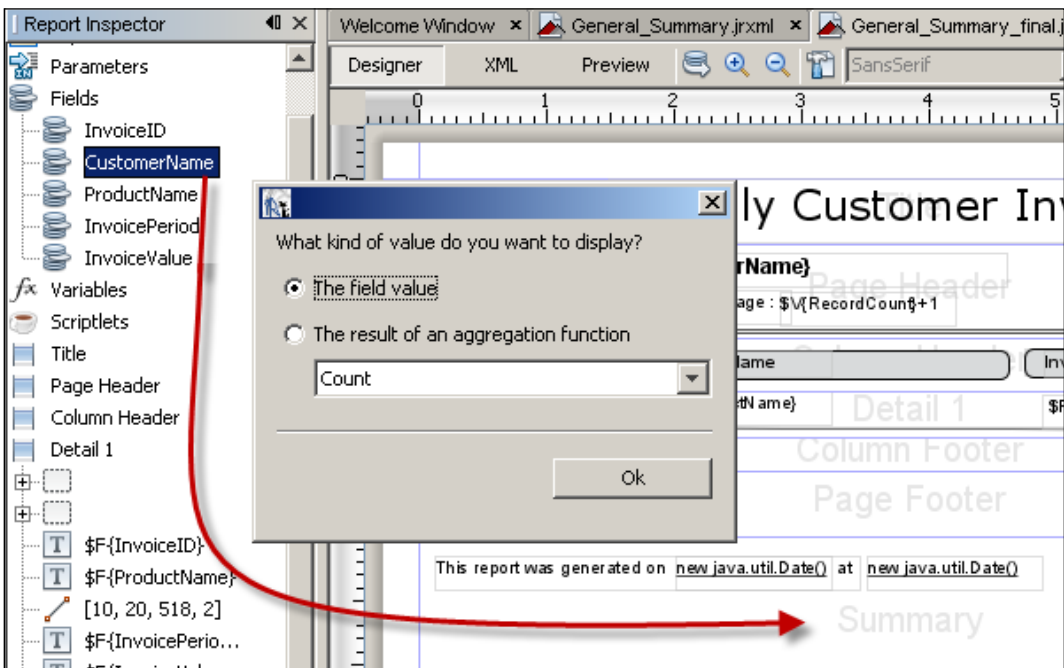
11. Switch to the **Preview** tab and go to the last page of your report. You will find a summary of your report at the end of this page, as shown in the following screenshot:



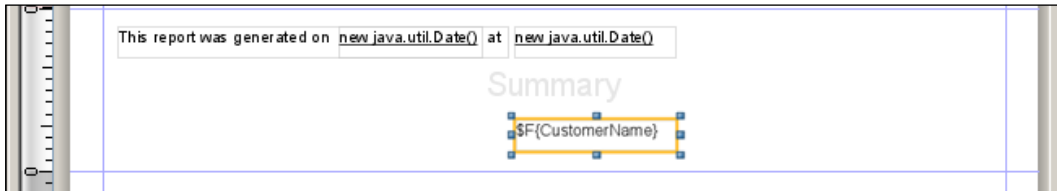
12. Switch to the **Designer** tab. Click on the Summary section of your report and go to the **Band properties** section of **Properties** below the palette and select the **Band height** property. Change its value to **100**. This will increase the height of the Summary section in your report, as shown in the following screenshot:



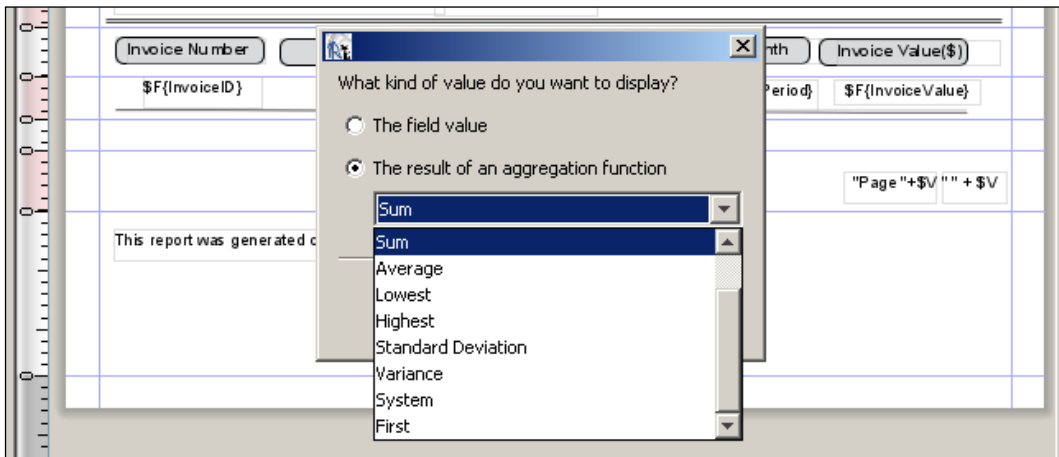
13. The left-most part of the main **iReport** window is a **Report Inspector** window, which shows a tree with many nodes. Double-click on the **Fields** node; it will list all its child fields.
14. Drag-and-drop **CustomerName** field from the **Fields** node into the Summary section of your report. A dialog window will appear, as shown in the following screenshot:



15. Don't change anything; just click the **Ok** button to dismiss it. A text field component with the expression `#{ProductName}` will appear in the Summary section of your report, as shown in the following screenshot:

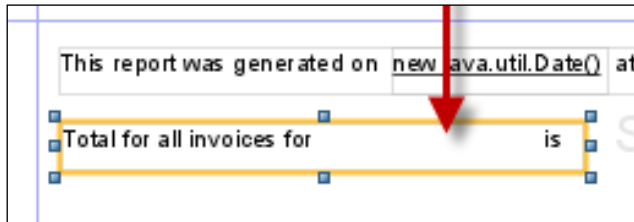


16. Similarly, drag-and-drop **InvoiceValue** field from the **Fields** node into the Summary section of your report. A dialog window will appear; this time select **The result of an aggregation function** from the **Radio buttons** option and **Sum** from the **Combo box** option, as shown in the following screenshot:



17. Click **Ok** to dismiss the dialog window.
18. Drag-and-drop a Static Text component from the palette of components into the Summary section just below the already dropped Static Text component.
19. Double-click and edit the Static Text component and type **Total for all invoices for is** as the summary text in the Summary section of your report. Part of the summary text will disappear from view in the static text component. So, enlarge the rectangular boundary of the static text component by dragging one of its corners, just like in step 6 of this recipe.

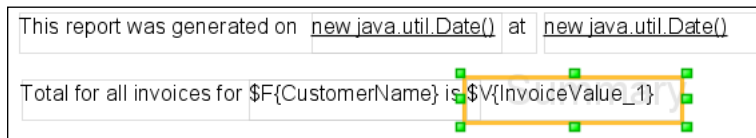
20. Insert whitespace of 30 characters between the words `Invoices` `for` and `is` of the static text by pressing the spacebar 30 times, as shown in the following screenshot:



21. Now use the arrow keys to move the `CustomerName` text field component until it sits adjacent to the word `Invoices` `for` of the summary text, as shown in the following screenshot:



22. Similarly, move the `InvoiceValue` text field component until it touches the boundary of the static text component containing the summary text, as shown in the following screenshot:



23. Select the **CustomerName** and **InvoiceValue** text field components while keeping the **Ctrl** key pressed. Release the **Ctrl** key and click the **Underline** button on the formatting bar of the **Designer** tab in your report window.

24. Switch to the **Preview** tab and go to the last page of your report; your report summary will look as shown in the following screenshot:

Customer Name : Packt Publishing

The total number of invoice(s) in this page : 6

Invoice Number	Product Name	Invoice Month	Invoice Value(\$)
1031	Printing Ink	Jan09	5240.0
1032	Packing Material	Mar09	5020.25
1033	Offset Paper	Jan09	3000.5
1034	Offset Paper	Jun09	3000.5
1035	Offset Paper	Aug09	3000.5
1036	Offset Paper	Sep09	3000.5

This report was generated on October 26, 2009 at 7:21 PM

Total for all invoices for Packt Publishing is 156167.05

How it works...

You dragged-and-dropped a **Current date** component into the Summary section and configured it to display the date in the required format in steps 2 and 3 of the recipe. Similarly, you dragged-and-dropped the same **Current date** component into the Summary section and configured it to display the time in step 4 of the recipe.

Then, in steps 5 to 10, you transformed this information into a presentable style. In step 14 of the recipe you dragged-and-dropped the **CustomerName** field element into the Summary section of your report. iReport intelligently identified that there are a number of records which cannot be displayed in a single text field and therefore presented a dialog window asking your permission for the aggregation function. In step 15, you kept the default selection which was **The field value**. This option tells JasperReports to use the first occurring value of the **CustomerName** field.

Similarly, in step 16, you dragged-and-dropped an **InvoiceValue** field element into the Summary section. iReport identified that there is more than one entry beside it. Therefore, iReport asked your permission for the aggregation function and you told iReport to perform a sum operation on the values by selecting **The result of an aggregation function** from the **Radio buttons** option and **Sum** from the **Combo box** option.

3

Enhancing the Look and Feel of your Report

In this chapter, you will learn:

- ▶ Deploying and reusing styles in your report
- ▶ Setting background color for data
- ▶ Using HTML tags and bullet lists
- ▶ Expanding a field vertically to accommodate large text
- ▶ Applying a formatting pattern to the value of a data field
- ▶ Using background images and watermarks in your report

Introduction

Graphical and textual effects greatly improve the readability, look, and feel of your reports. For example, a big, bold title, a highlighted heading, a background image, and the use of appropriate fonts and styles all make your report reader friendly.

This chapter contains recipes to produce several types of enhanced visual effects like:

- ▶ Using different fonts, sizes, colors, and styles
- ▶ Building reusable styles
- ▶ Applying formatting to fields and groups
- ▶ Using background colors and images
- ▶ Expanding fields to accommodate large text

Deploying and reusing styles in your report

Styles such as borders, frames, and fonts make your report attractive and user friendly. Companies normally design style templates and apply them to different business documents such as reports. This provides consistency in the look and feel of variety of documents produced by the same company.

If you have read any Packt Publishing Cookbooks, you will find this consistency in the style (heading styles, fonts used for different types of text, and so on) across different cookbooks.

JasperReports allows you to design a template of styles once and reuse it to produce a variety of reports. This recipe shows you how to create a style template with a particular font size, color, and line width. The recipe will show you how to use the template in your report. You will also learn how to apply textual styles directly to your report without making a template. The recipe also shows how to use rectangles and frames to design borders in your report.

Getting ready

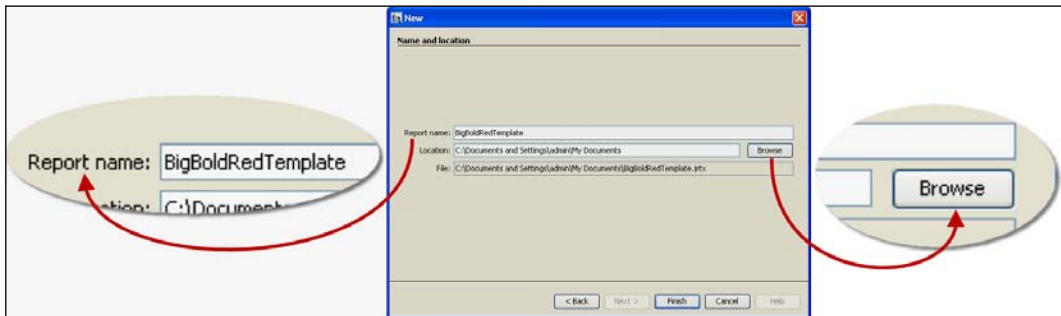
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

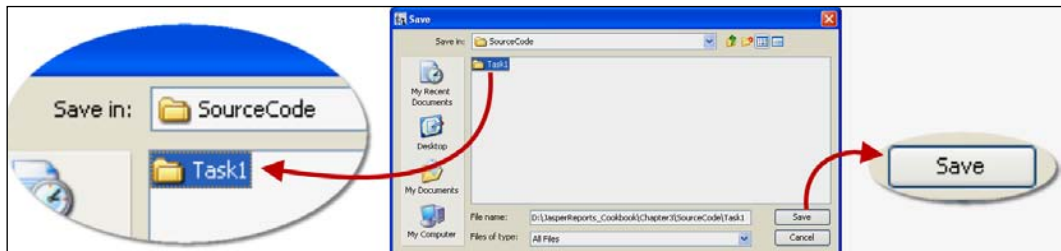
1. Click **File** and select **New** from the **File** menu. A **New file** dialog will appear. Select **Style** from the **New file** dialog window and click the **Finish** button.



- The next screen (a **Name and location** window) will ask you for the name of the new style template file and the location where you want to save this file. Type `BigBoldRedTemplate` in the **Report name** field and click on the **Browse** button to locate the folder where you want to save the style template file.

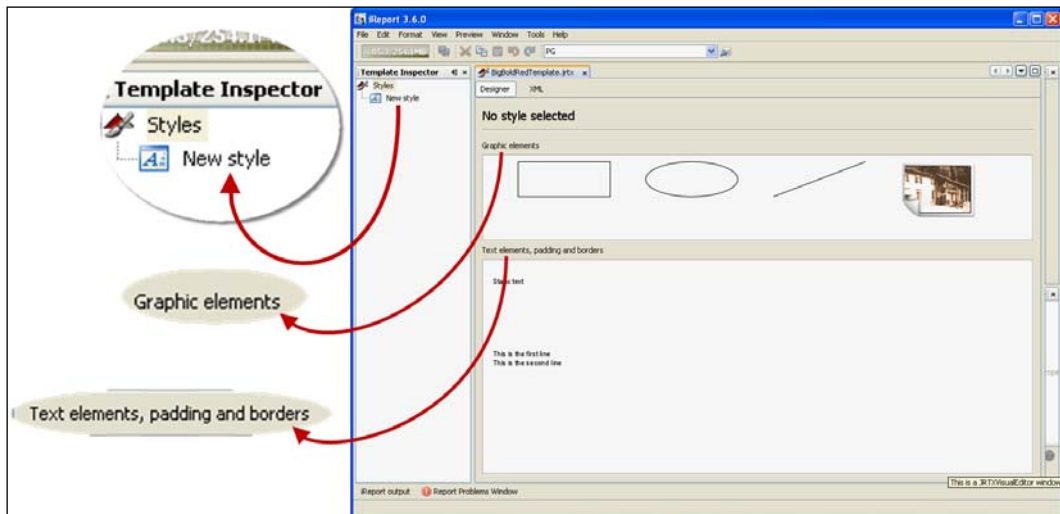


- A **Save** dialog will appear, which will help you locate the folder where you want to save the style template file. Select the `Task1` folder from the source code for this chapter and click on the **Save** button. The **Save** dialog will disappear and you will be back to the **Name and location** window.

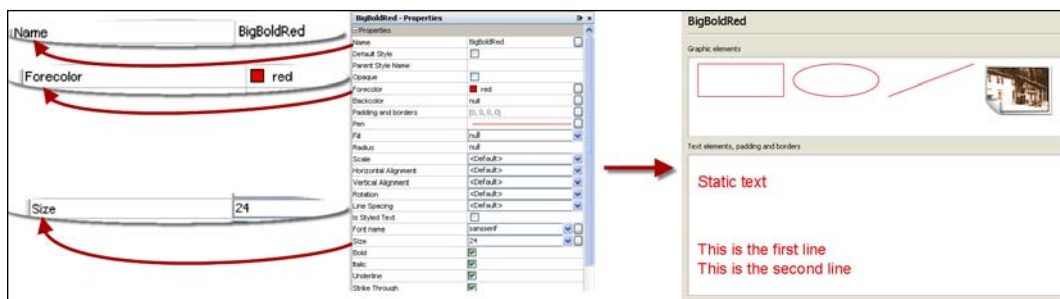


- Click the **Finish** button at the bottom of the **Name and location** window; this will create an empty `BigBoldRedTemplate.jrtx` file in the `Task1` folder of the source code for this chapter and will also dismiss this **Name and location** window.

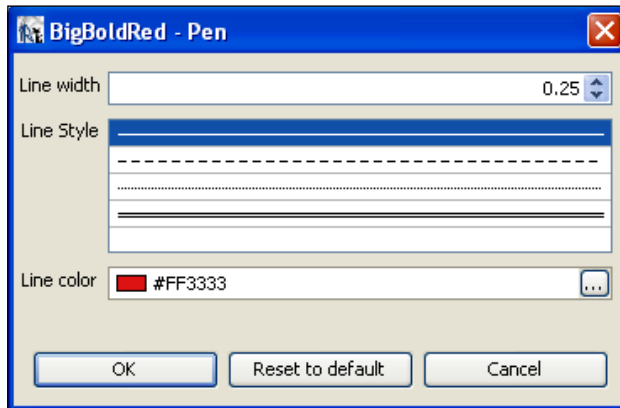
5. Your new style template will open in the **Designer** tab, which has two sections. The upper section shows the look of graphical components (like rectangle, eclipse, and so on) in your style, while the lower portion shows the appearance of text in this style. The left-most part of the main iReport screen is a **Template Inspector** window, which has a tree of styles showing all the style templates that you may have previously created. If you have not created any style before, your template inspector window will show only one node named **New Style**.



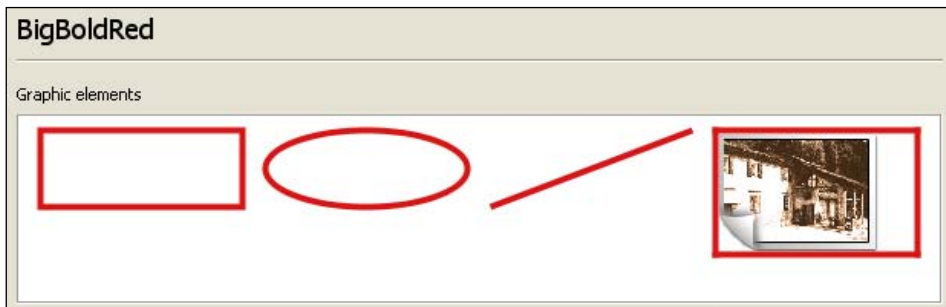
6. Click on the **New Style** node; its properties window will appear below the palette of components on the right of your **Designer** tab. Properties are actually associated with the style template you are creating.
7. Switch to the properties window of your style; select **Name** property and type **BigBoldRed** as the name of the style you are creating. Now select the **Size** property and change its value to 24. Then click on the **Forecolor** property and click the selection button in its value field. A color selector window will open; select **red** as the foreground color and click **OK**. Now the **Designer** tab will look like the following:



8. Again switch to the properties window and select its **Pen** property, which allows you to specify the width, style, and color of the line used in drawing graphical components. Click the selection button in its value field. A **Line Style** selection window named **BigBoldRed - Pen** will appear, as shown in the following screenshot:



9. Type 4 as the value of the **Line width** field and click **OK** button. This line width affects only the graphical components, so the upper graphical section of the **Designer** tab will look as follows. You will notice that graphical components (rectangle, ellipse, line, and so on) are now shown as bold red.



10. You have created your style template. Save it by selecting **Save** from the **File** menu. You can now switch to the **XML** tab to see the XML view of the style template.

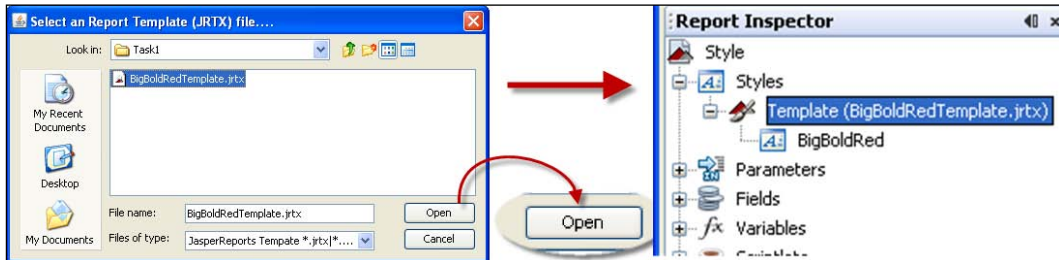
11. Now you will use the style template in the title of your business report. Open `Style.jrxml` file from the `Task1` folder in the source code for this chapter. The **Designer** tab of iReport shows a report with a small, dull, black title, **Monthly Customer Invoices**, as shown in the following screenshot:

Monthly Customer Invoices				
Customer Name: \$F{CustomerName}				
Product Name: \$F{ProductName}				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
Total Amount:				\$V{InvoiceValue_1}

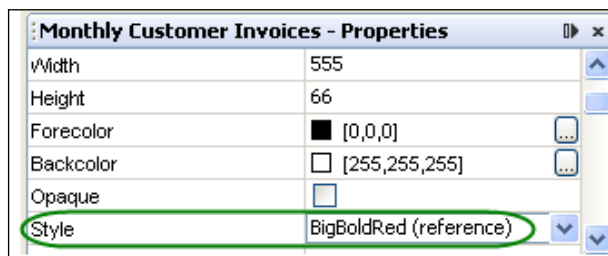
12. The left-most part of the main iReport screen is a **Report Inspector** window, which shows a tree with many nodes. The first node is named **Styles** as shown in the following screenshot:



13. Right click on the **Styles** node, and a pop-up menu will appear. Select the **Add** option from the pop-up menu; another pop-up menu will appear. Select the **Style Reference** option from the new pop-up menu. A **Select an Report Template (JRTX) file** dialog box will appear from where you will browse to the style template you created in steps 1 to 10 of this recipe. Select the `BigBoldRedTemplate.jrtx` file and click on the **Open** button. You will see that the **Template (BigBoldRedTemplate.jrtx)** sub-node has now been added as a child node of the **Styles** node in the **Report Inspector** window.



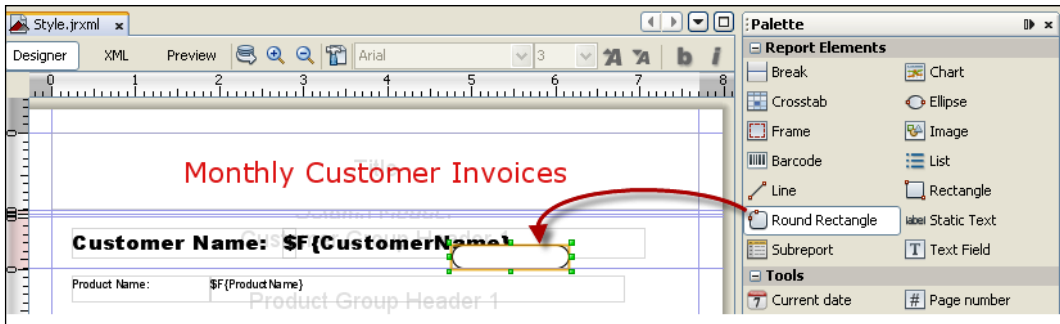
14. Now you will apply the `BigBoldRed` style to the title of your report. Click on the title of your report, **Monthly Customer Invoices**. Switch to the **Properties** window, which is now showing the properties of the title. Look for the **Style** property; its value is a drop-down list, in which you will find your `BigBoldRed` (reference) style that you created earlier in steps 1 to 10. Select the `BigBoldRed` (reference) style from the drop-down list:



15. Click the **Preview** button and you will see that your small, dull, black title has changed into a big, bold red title.

Monthly Customer Invoices

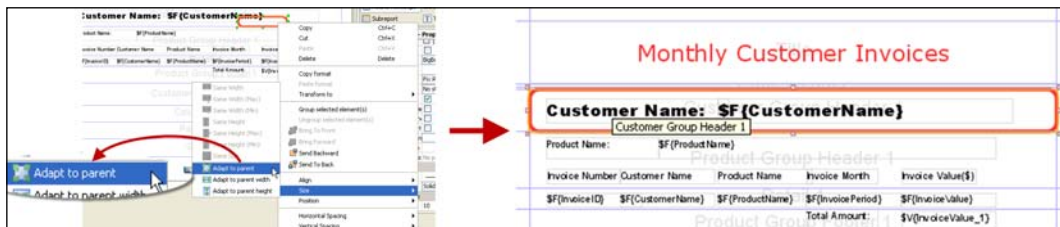
- Switch back to the **Designer** tab. Drag-and-drop a **Round Rectangle** component from the **Palette** into the **Customer Group Header 1** section of your report.



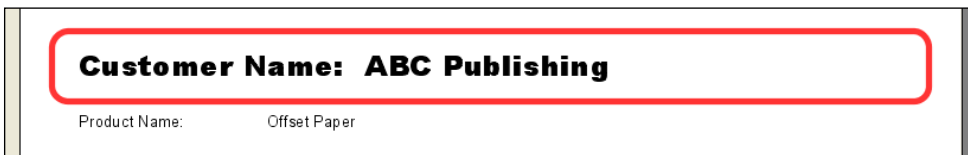
- Select the **Round Rectangle** component you just dropped. The **Properties** window, below the **Palette** of components, will show the properties of the **Round Rectangle** component you just dropped. Find the **Opaque** property and uncheck the checkbox beside it. This will make the **Round Rectangle** transparent, as shown in the following screenshot:



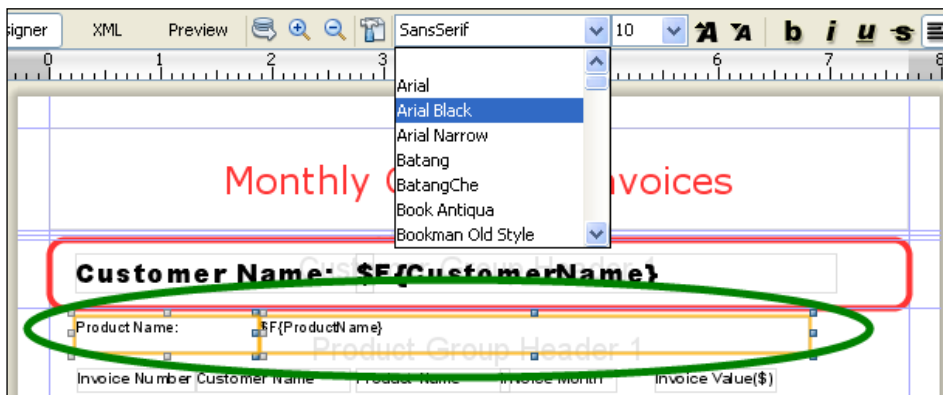
- While the **Round Rectangle** component of the **Customer Group Header 1** section is selected, find the **Style** property; its value is a drop-down list, in which you will find your **BigBoldRed** (reference) style that you created earlier in steps 1 to 10. Select the **BigBoldRed** (reference) style from the drop-down list.
- Right-click the **Round Rectangle** component, and a pop-up menu will appear. Select the **Size** option, and another pop-up menu will appear. Choose the **Adapt to Parent** option from the sub pop-up menu. This will size your border to equal the size of your **Customer Group Header 1** section.



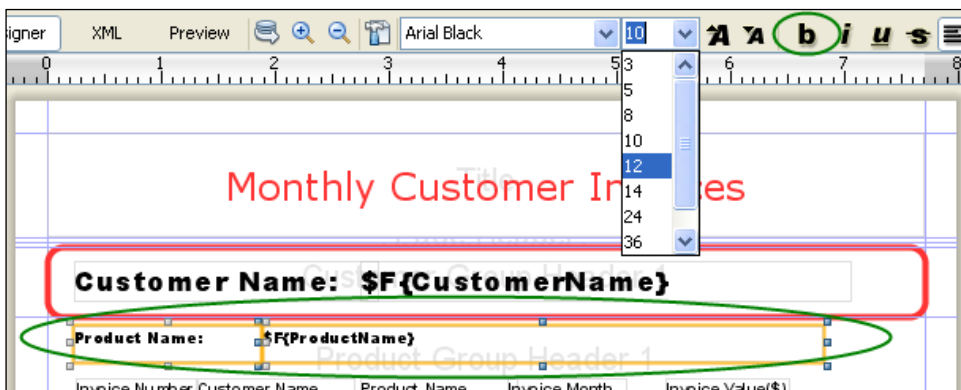
20. Switch to the **Preview** tab of your report, which shows the view of your report:



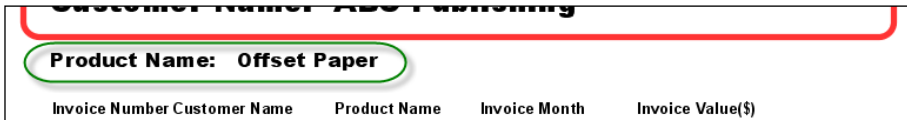
21. Switch back to the **Designer** tab. Select the **Product Name: Static Text** component and the **Text Field** component with the expression `$F{ProductName}`, both present in the **Product Group Header 1** section. While the text field and static text components of **Product Group Header 1** section are selected, expand the font selection combobox on the formatting toolbar and select the **Arial Black** option.



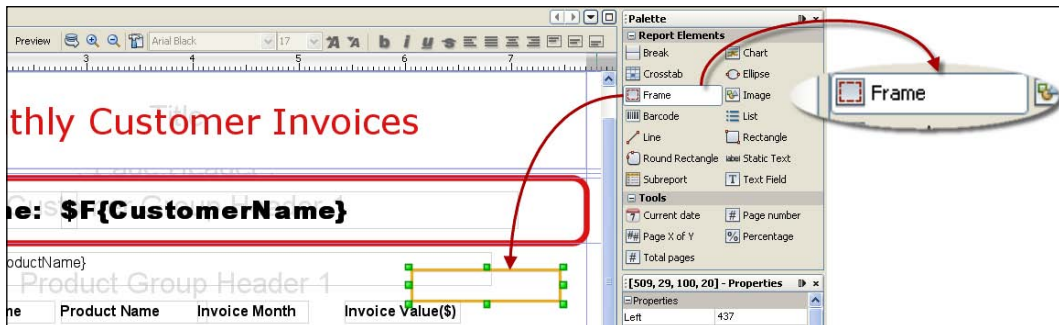
22. While the text field and static text components of the **Product Group Header 1** section are selected, expand the font size selection combobox on the formatting toolbar and select the value of 12. Also click on the **bold** button on the formatting toolbar.



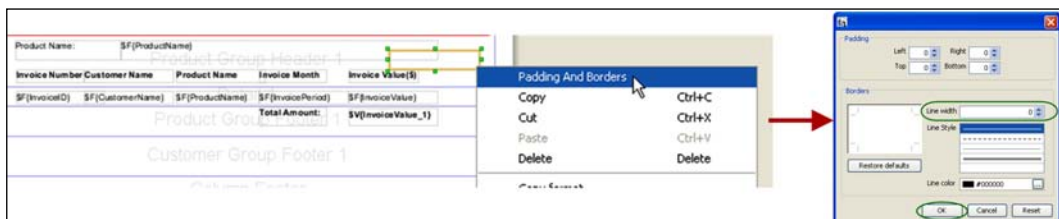
23. Switch to the **Preview** tab; your report will show prominent product names, as shown in the following screenshot:



24. Switch back to the **Designer** tab. Drag-and-drop a **Frame** component from the **Palette** (on the right side of your iReport main window) into the **Product Group Header 1** section.

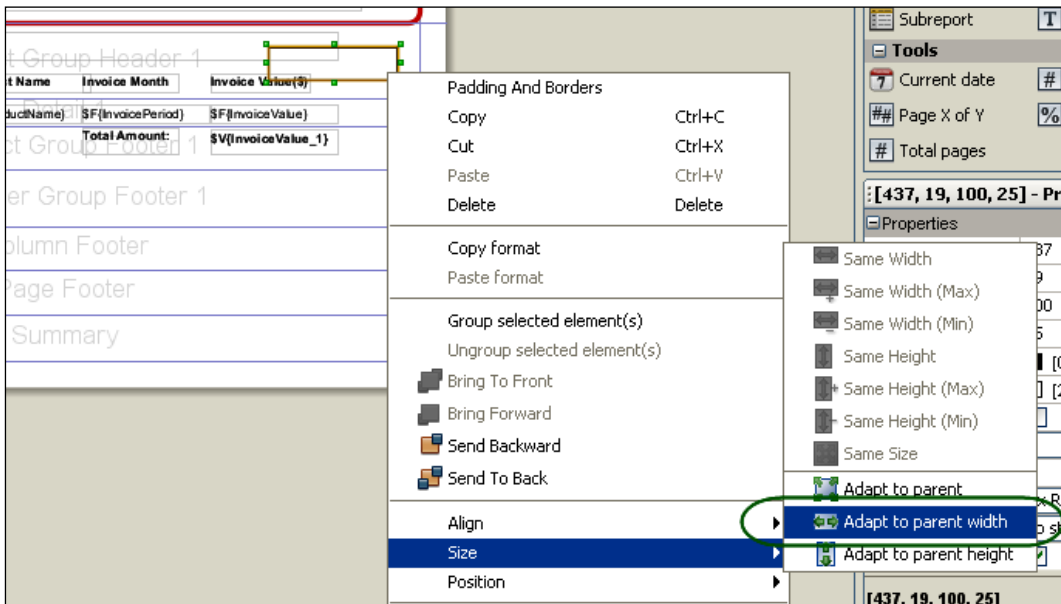


25. Right-click the **Frame** component, select the **Padding and Borders** option from the pop-up menu, which will open a window containing options for padding and border styles (padding, width, style, color of border, and so on) as shown in the following screenshot. You will notice that the dialog allows you to adjust the width, style, and color of the line to be used as the border of the frame. Set the value of the **Line width** field to 1. Leave the rest of the properties to their default values. Click the **OK** button to dismiss the window.

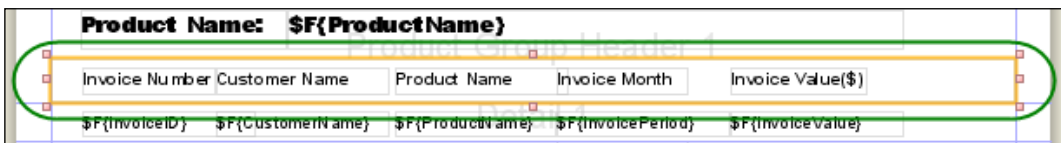


26. While the **Frame** component you just dropped is selected, switch to the **Properties** window, which is now showing the properties of the **Frame** component. Look for the **Height** property, and set its value to 25.

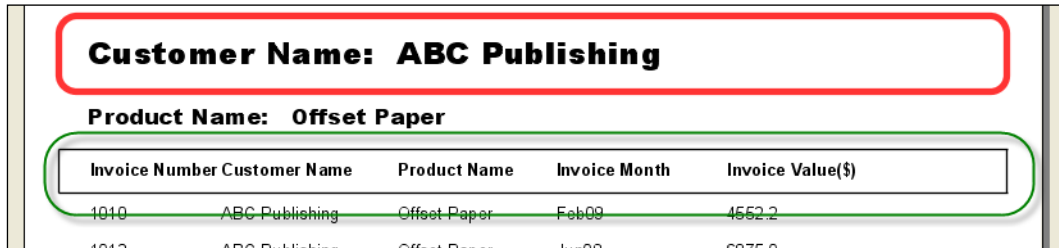
27. Right-click the **Frame** component; select the **Size** option from the pop-up menu. Choose the **Adapt to Parent width** option from it. This will set the width of your frame equal to the width of your **Product Group Header 1** section.



28. Again, right-click the **Frame** component. This time, select the **Align** option from the pop-up menu, and a sub pop-up menu will appear. Choose the **Align To Bottom Margin** option from it. This will move the frame downwards to touch the bottom of the **Product Group Header 1** section so that the frame will wrap all the headings of your table columns, as shown in the following screenshot:



29. Switch to the **Preview** tab and you will see an outline around the headings of your table columns in your report body, as shown in the following screenshot:

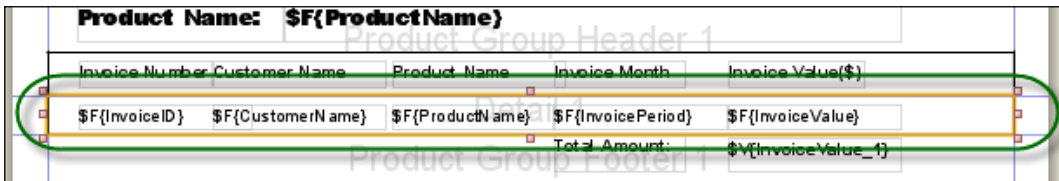


Customer Name: ABC Publishing

Product Name: Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0

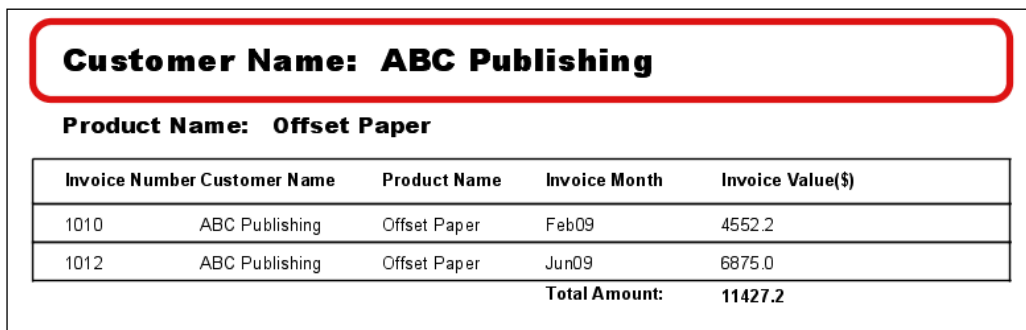
30. Switch back to the **Designer** tab. Drag-and-drop another **Frame** from the **Palette** into the **Detail 1** section.
31. Right-click the **Frame** component of the **Detail 1** section, and a pop-up menu will appear. Select the **Padding and Borders** option from it, which will open a window containing options for padding and border styles. Set the value of the **Line width** field to 1. Leave the rest of the properties at their default values and click on **OK**.
32. Right-click the **Frame** component of the **Detail 1** section; select the **Size** option from the pop-up menu. Choose the **Adapt to Parent** option from the new pop-up menu. This will set the size of your frame equal to the size of your **Detail 1** section.



Product Name: \$F{ProductName}

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
Total Amount:				\$V{InvoiceValue_1}

33. Switch to the **Preview** tab and you will see an outline around each record in your report body.

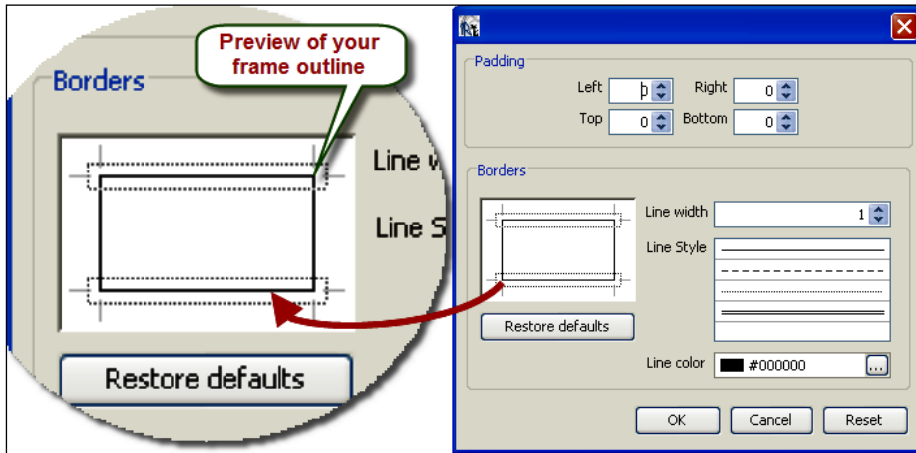


Customer Name: ABC Publishing

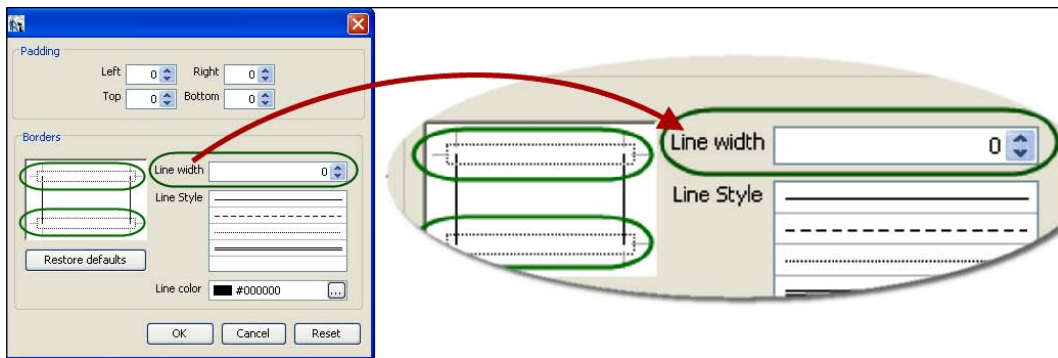
Product Name: Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Total Amount:				11427.2

34. Switch back to the **Designer** tab. Right-click the **Frame** component; select the **Padding and Borders** option, which will open a border and padding options window.



35. Look at the left half of the border and padding options window in the **Borders** section. You will see a preview window showing your frame border. Click the top and bottom lines of your frame in the preview window. Set 0 as the value of the **Line width** field. This will make the top and bottom lines of the frame invisible, as shown in the following screenshot. Click the **OK** button.



36. Switch to the **Preview** tab and compare the result with the preview of step 33. You will notice this time that the top and bottom lines of the frame, which were wrapping each individual record, have been removed.

Product Name: Offset Paper				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Total Amount:				11427.2
Product Name: Packing Material				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Total Amount:				18597.5

37. Switch back to the **Designer** tab. Drag-and-drop another **Frame** component from the **Palette** into the **Product Group Footer 1** section.
38. Right-click the newly dropped **Frame** component; select the **Size** option, and a sub pop-up menu will appear. Choose the **Adapt to Parent** option from the sub pop-up menu. This will size your frame equal to the size of your **Product Group Footer 1** section.
39. Right-click the **Frame** component; select the **Padding and Borders** option, which will open the border and padding options window. Set the value of the **Line width** field to 1 and click the **OK** button.

40. Switch to the **Preview** tab and you will see an outline around the set of records for each product in the body of your report, as shown in the following screenshot:

Monthly Customer Invoices				
Customer Name: ABC Publishing				
Product Name: Offset Paper				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Total Amount:				11427.2
Product Name: Packing Material				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Total Amount:				18597.5

How it works...

Reusable styles are defined and saved inside a JRTX template file. These predefined styles can be used in any JRXML file after linking it to the JRTX file, which you have learned in this recipe.

You have done four things here:

1. First you created and saved a reusable style named `BigBoldRed` inside a JRTX file named `BigBoldRedTemplate.jrtx` in steps 1 to 10 of the recipe.
2. Next you linked `BigBoldRedTemplate.jrtx` file to the `Style.jrxml` file in step 13 of the recipe and applied the reusable style to your report in steps 14 to 20.
3. Then you learned how to apply some non-reusable styles (Arial Black font of size 12 with bold style) in steps 21 to 23.
4. Finally, in the last 16 steps, you used frames as borders to wrap the headings and records of your report.

All these four things help you understand that you can either keep your styles separate from your report (like you did in steps 1 to 20) or you can build the style directly into your report (like you did in steps 21 to 23).

Setting background color for data

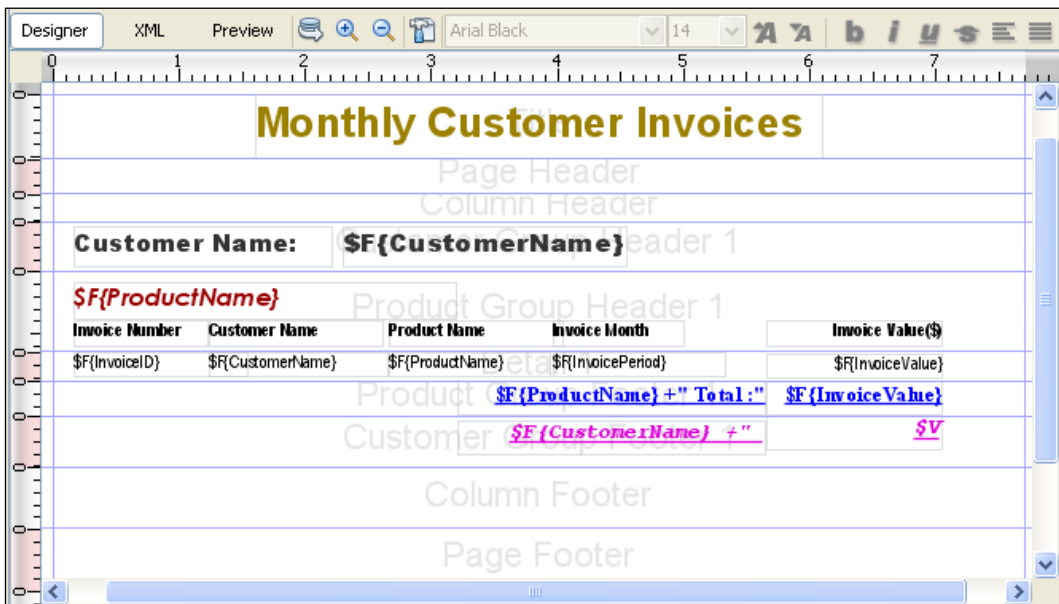
Using different background colors in different sections of a report produces a very good visual effect to distinguish between different sections. This recipe demonstrates how to use different background colors in your report.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

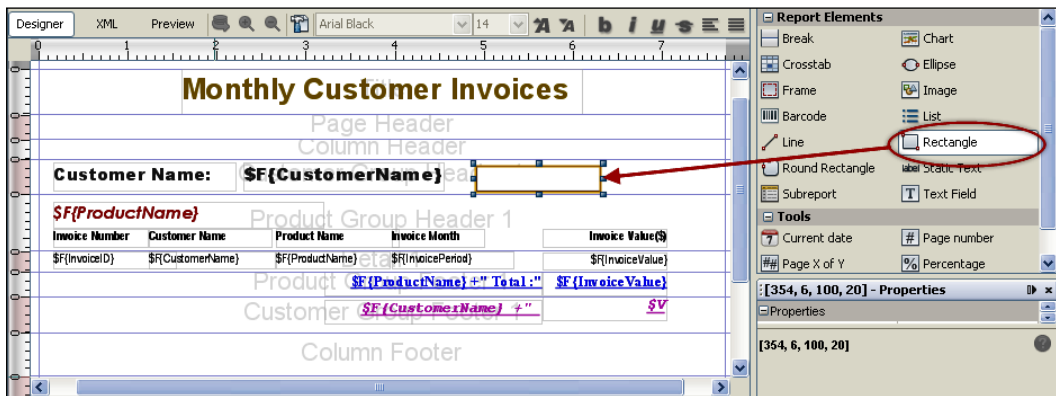
1. Open the `BackgroundColor.jrxml` file from the `Task2` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, **Product Group Footer 1**, and **Customer Group Footer 1** sections, as shown in the following screenshot:



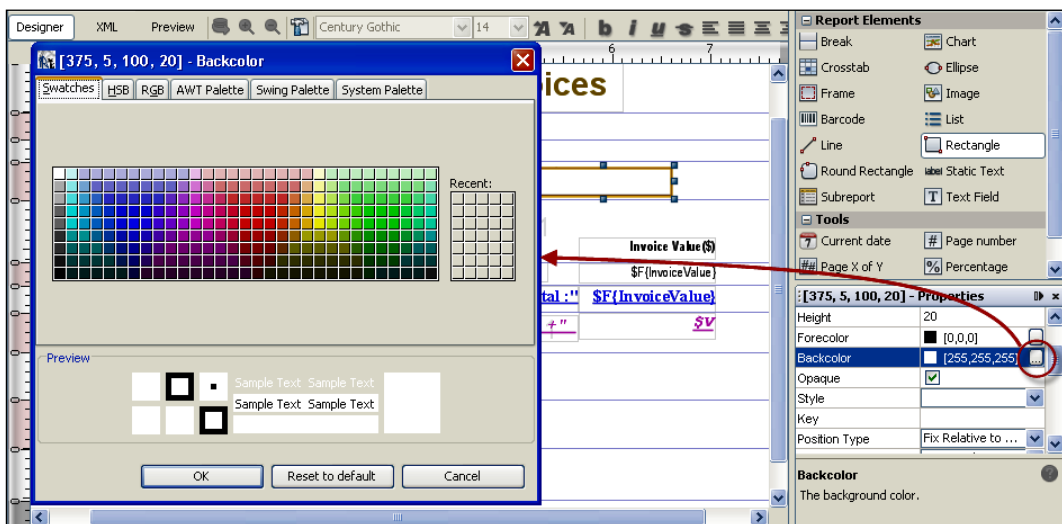
- Switch to the **Preview** tab to see that the report shows invoices for each customer categorized by products sold to the customer.

Monthly Customer Invoices				
Customer Name: ABC Publishing				
<i>Offset Paper</i>				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4952.2
<u>Offset Paper Total:</u>				<u>4552.2</u>
<i>Packing Material</i>				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
<u>Packing Material Total:</u>				<u>9852.0</u>
<i>Printing Ink</i>				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
<u>Printing Ink Total:</u>				<u>2440.0</u>
<u>ABC Publishing Total:</u>				<u>35914.7</u>
Customer Name: Alice				
<i>JasperReport Cookbook</i>				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1038	Alice	JasperReport Cookbook	Mar09	49.5
1007	Alice	JasperReport Cookbook	Apr09	49.5
<u>JasperReport Cookbook Total:</u>				<u>49.5</u>
<i>WordPress Cookbook</i>				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1014	Alice	WordPress Cookbook	Jan09	45.0
<u>WordPress Cookbook Total:</u>				<u>45.0</u>
<u>Alice Total:</u>				<u>144.0</u>
Customer Name: Bob				

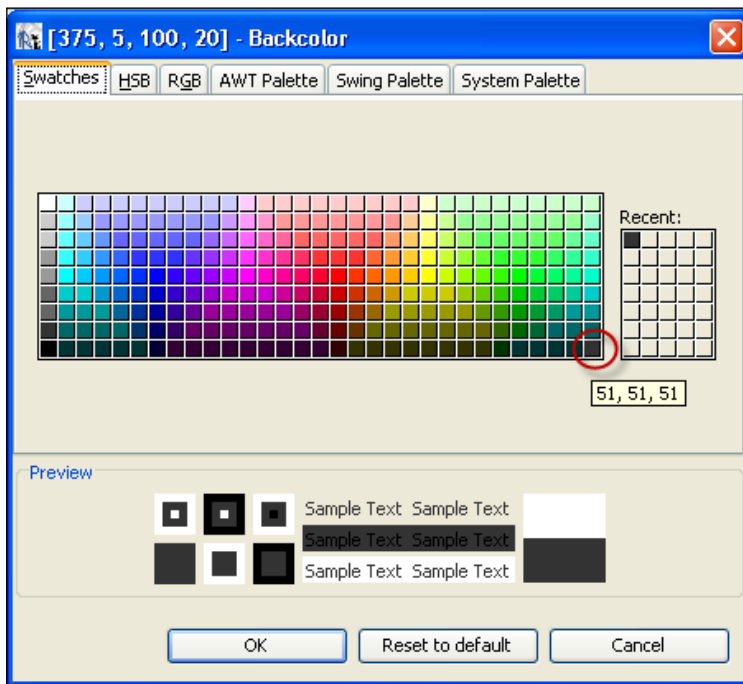
- Switch to the **Designer** tab. Drag-and-drop a **Rectangle** component from the **Palette** on the right of the iReport main window into the **Customer Group Header 1** section of your report.



- Click the **Rectangle** component you just dropped; its properties will appear in the **Properties** window below the **Palette** of components. Find the **Backcolor** property and click the button next to it, as shown encircled in following screenshot. A color selector window will open.



5. Select the color black and click **OK**.



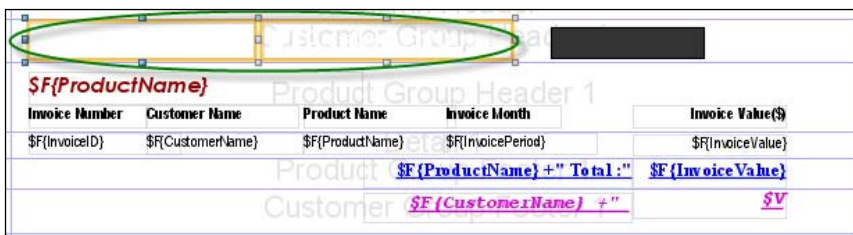
6. Now the **Rectangle** will look like the following:

Customer Name:		<code>\$F{CustomerName}</code>			
<code>\$F{ProductName}</code>					
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)	
<code>\$F{InvoiceID}</code>	<code>\$F{CustomerName}</code>	<code>\$F{ProductName}</code>	<code>\$F{InvoicePeriod}</code>	<code>\$F{InvoiceValue}</code>	
Product Group Header 1					
<code>\$F{ProductName} + " Total :"</code>				<code>\$F{InvoiceValue}</code>	
Customer				<code>\$F{CustomerName} + "</code>	
				<code>\$V</code>	

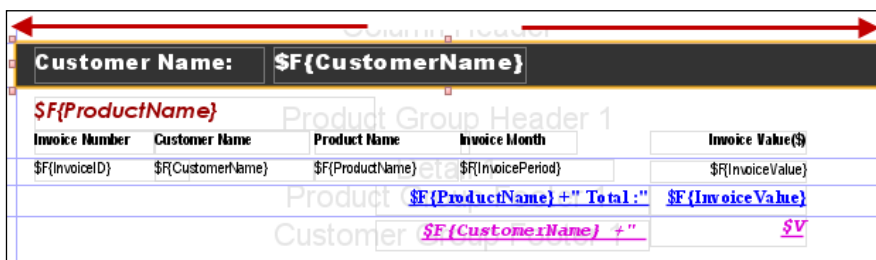
- Click the **Static Text** component with **Customer Name:** as its text, in the **Customer Group Header 1** section. Press the **Ctrl** key and click the **Text Field** component with the expression `$F{CustomerName}` in the **Customer Group Header 1** section. You will see that both components are selected.



- Right-click on any of the selected components, and a pop-up menu will appear; select the **Bring to Front** option.
- Now, while keeping both components selected, find the **Forecolor** property in the **Properties** window below the **Palette** of components. A color selector window will open select the color white and click **OK**. The text inside the components will disappear from view, as shown encircled in the following screenshot:

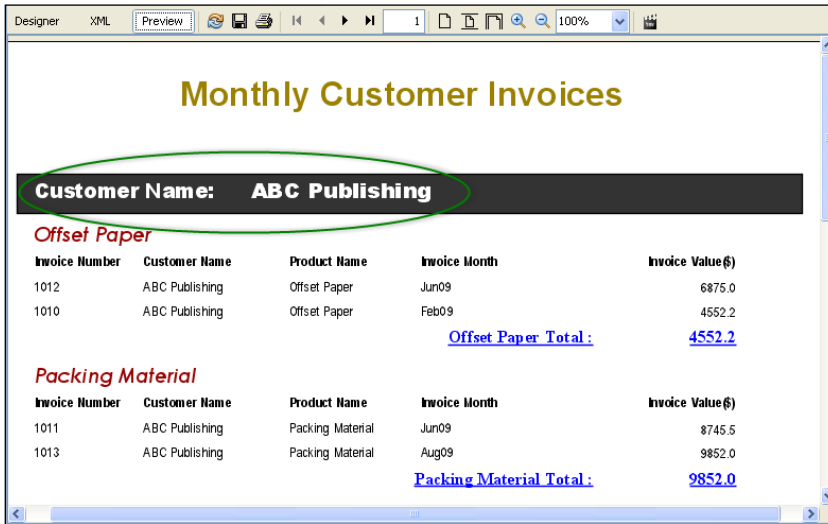


- Right-click the **Rectangle** component inside the **Customer Group Header 1** section, and a pop-up menu will appear. Select the **Size** option, and a sub-menu will open. Choose the **Adapt to Parent** option from the sub-menu. The size of the rectangle will become equal to the size of the **Customer Group Header 1** section.



Now you can understand the importance of step 8. If you had not brought the two components to the front in step 8, they would have been hidden behind the rectangle and become invisible.

11. Switch to the **Preview** tab and you will see that the customer name heading in your report body is shown in reverse color as follows:



12. Switch to the **Design** tab, and drag-and-drop another **Rectangle** component from the **Palette** into the **Detail 1** section. While dropping the rectangle, make sure that its upper side touches the blue line that separates the **Detail 1** and **Product Group Header 1** sections, as shown in the following screenshot:



13. Click the newly dropped **Rectangle** component inside the **Detail 1** section; its properties will appear in the **Properties** window below the **Palette** of components. Find the **Backcolor** property and click the button next to it. A color-selector window will open; set the background to the color gray and click **OK**. Now the **Rectangle** will look like the following screenshot:



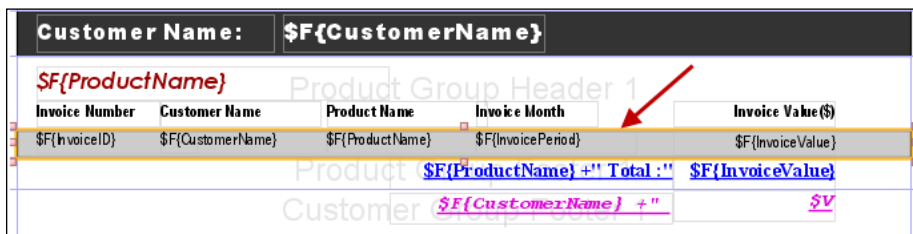
Customer Name:		\$F{CustomerName}		
\$F{ProductName}				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
\$F{ProductName} +" Total :"				\$F{InvoiceValue}
\$F{CustomerName} +"				\$V

14. Right-click on the selected **Rectangle** component in the **Detail 1** section, and a pop-up menu will appear; select **Send to Back** option. You will see that the rectangle has become the background so that the other components placed in the **Detail 1** section are now visible.



Customer Name:		\$F{CustomerName}		
\$F{ProductName}				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
\$F{ProductName} +" Total :"				\$F{InvoiceValue}
\$F{CustomerName} +"				\$V

15. Select the **Rectangle** component inside the **Detail 1** section. Right-click on it, and a pop-up menu will appear. Select the **Size** option, and a sub-menu will open. Choose the **Adapt to Parent** option from the sub-menu. The size of the rectangle will become equal to its parent **Detail 1** section.



Customer Name:		\$F{CustomerName}		
\$F{ProductName}				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
\$F{ProductName} +" Total :"				\$F{InvoiceValue}
\$F{CustomerName} +"				\$V

16. Switch to the **Preview** tab and you will see gray background for each product invoice. However, the look and feel of the report is not good, perhaps because data for each invoice is outlined with a black border.

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
Offset Paper Total :				4552.2

Packing Material

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				9852.0

17. Switch to the **Designer** tab; select the **Rectangle** component in the **Detail 1** section, and its properties will appear in the **Properties** window. Find the **Forecolor** property and click the button next to it. A color-selector window will open; set the border color as white and click **OK**. Now switch to the **Preview** tab. Your report will look as follows:

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Offset Paper Total :				6875.0

Packing Material

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				9852.0

Printing Ink

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
Printing Ink Total :				3450.0

ABC Publishing Total : **35914.7**

How it works...

Switch to the **XML** tab and you will see the following JRXML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other JRXML tags -->
  <group name="Customer">
    <groupHeader>
      <band height="28">
        <rectangle>
          <reportElement x="0" y="0" width="555"
            height="28" backcolor="#333333"/>
        </rectangle>
      </band>
    </groupHeader>
  </group>
  <detail>
    <band height="17" splitType="Stretch">
      <rectangle>
        <reportElement x="0" y="0" width="555" height="17"
          forecolor="#FFFFFF" backcolor="#CCCCC"/>
      </rectangle>
    </band>
  </detail>
  <!-- other JRXML tags -->
</jasperReport>
```

I have shown only a `<group>` and a `<detail>` tag, omitting all the other JRXML tags. That's because you have worked only in the **Customer Group Header 1** and **Detail 1** sections in this recipe. The `<group>` and `<detail>` tags correspond to these two sections, respectively.

You can see two highlighted `<rectangle>` tags. iReport authored the first `<rectangle>` tag when you set the background color of the **Customer Group Header 1** section in steps 3 to 10 of the recipe.

Similarly, when you set the background color of the **Detail 1** section in steps 12 to 17, iReport authored the second `<rectangle>` tag.

Each `<rectangle>` tag has a `<reportElement>` child whose `backcolor` attribute specifies the background color of the rectangle.

Notice that the `<reportElement>` child of the first `<rectangle>` tag has a `height` attribute whose value is 28. This value is equal to the value of the `height` attribute of the rectangle's parent `<band>` tag. This means the child `<rectangle>` and the parent `<band>` have the same height. So the child occupies the parent completely. This is a result of the **Adapt to Parent** option you selected in step 10.

Using HTML tags and bullet lists

Sometimes you may want to present the data of your report as a bulleted list of self-explanatory sentences instead of presenting it in tabular form. Similarly, you may also need to use HTML tags in your bulleted list.

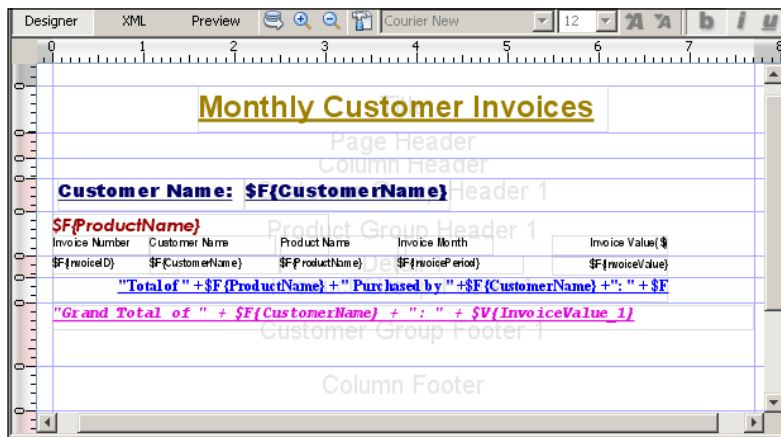
This recipe teaches you how to combine data fields coming from your database into a sentence. This way one sentence will represent one record from the database. Then you will display the self-explanatory sentences as a bulleted list using HTML tags. You will also use HTML style tags within the bulleted list.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

1. Open the `BulletedList.jrxml` file from the `Task3` folder in the source code for this chapter. The **Designer** tab of iReport shows a report, as shown in the following screenshot:



- Switch to the **Preview** tab. You will see a report with invoices grouped based upon customer names. For each customer, you will notice that the invoices are grouped based upon product names, forming a nested hierarchy. In this way, the report categorizes all invoices by product for each customer.

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
Total of Offset Paper Purchased by ABC Publishing: 4552.2				

Packing Material

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Total of Packing Material Purchased by ABC Publishing: 9852.0				

Printing Ink

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Total of Printing Ink Purchased by ABC Publishing: 2440.0				

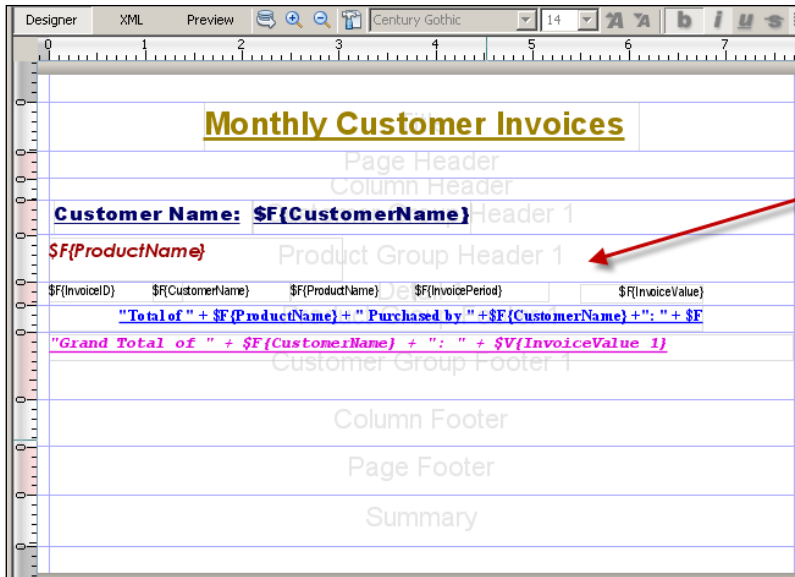
Grand Total of ABC Publishing: 35914.7

Customer Name: Alice

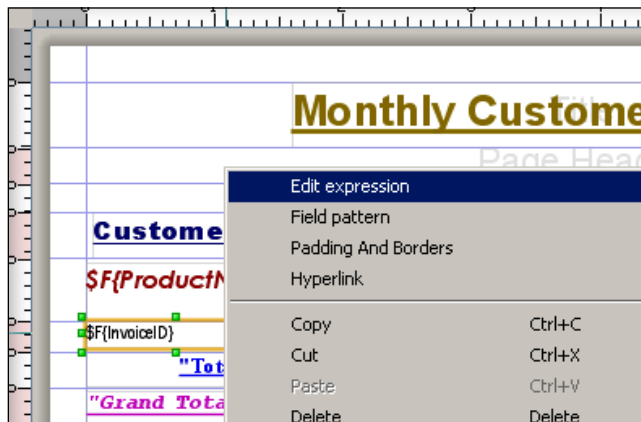
JasperReport Cookbook

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
----------------	---------------	--------------	---------------	-------------------

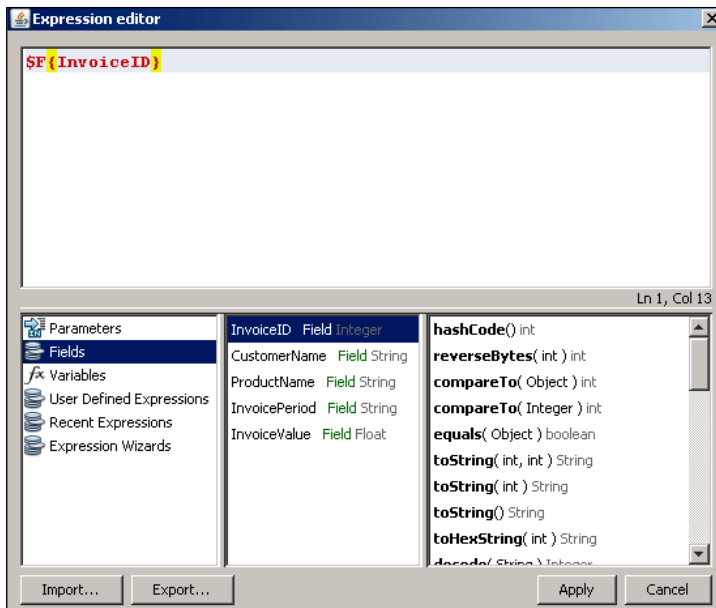
- Now you will convert the records of the nested group into a bulleted list. Switch back to the **Designer** tab.
- Select and delete the five **Static Text** components (having the text **Invoice Number**, **Customer Name**, **Product Name**, **Invoice Month**, and **Invoice Value(\$)**), which are in the **Product Group Header 1** section of your report. Your report will now look as shown in the next screenshot:



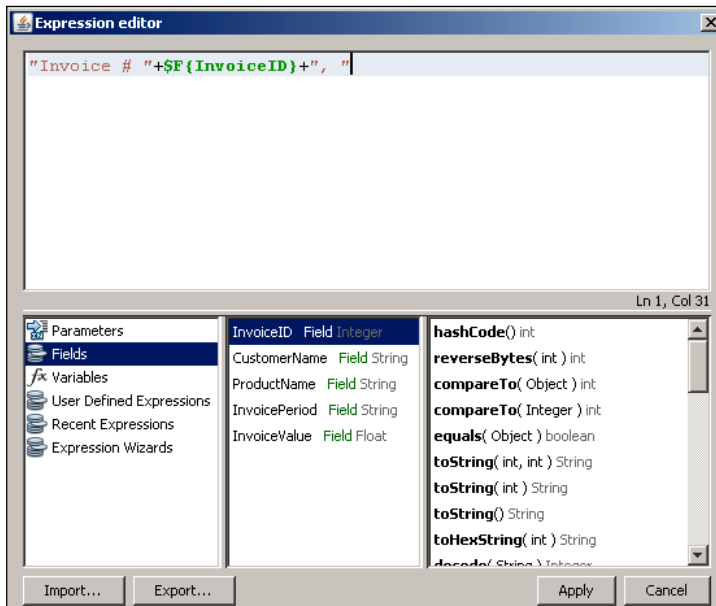
5. Select the first **Text Field** component in the **Detail 1** section (having the expression `$F{InvoiceID}`). Right-click on it, and a pop-up menu will appear. Select **Edit expression** from the pop-up menu.



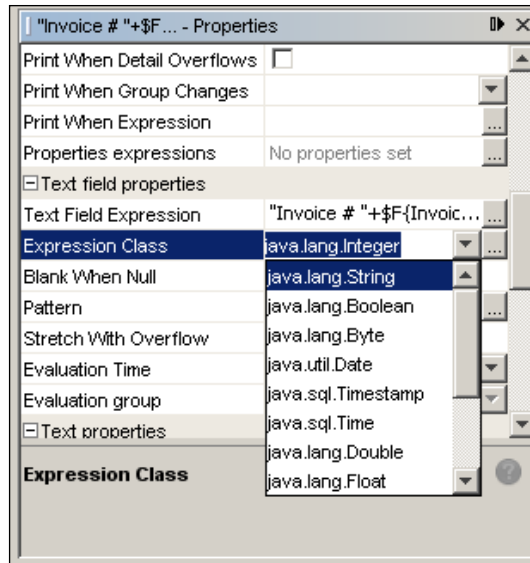
6. An **Expression editor** dialog will appear, as shown in the following screenshot:



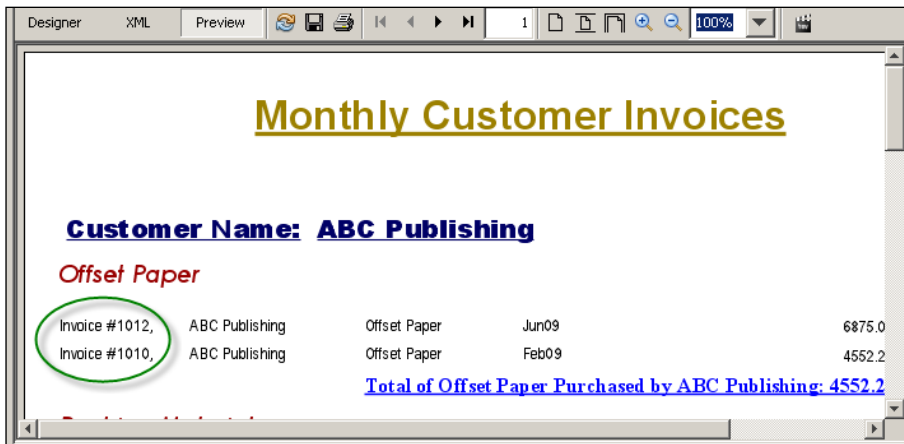
7. Take your cursor to the beginning of the expression text by using the arrow keys. Type "Invoice # + at this place. Now move the cursor to the end of the expression text and type +", . The text field expression will now become "Invoice # "+\$F{InvoiceID}+", ".



8. Click the **Apply** button, and the dialog will disappear.
9. While the first **Text Field** component of the **Detail 1** section is still selected, select the **Expression Class** property from the **Text field properties** section of the **Properties** window below the **Palette**. Change its value to `java.lang.String`.



10. Switch to the **Preview** tab. The report will look as shown in the next screenshot:



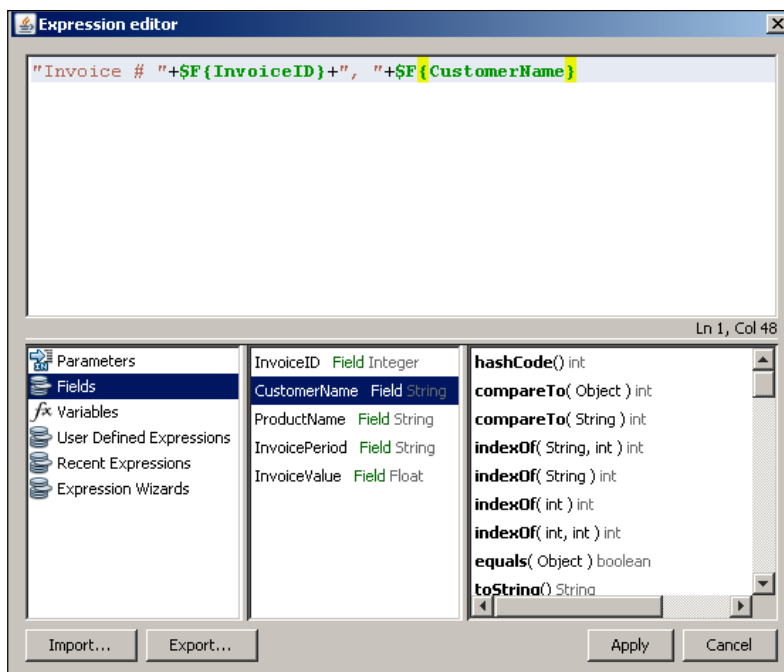
11. Switch back to **Designer** tab. Delete all the other **Text Field** components from the **Detail 1** section. Now it will have only one **Text Field** component left.



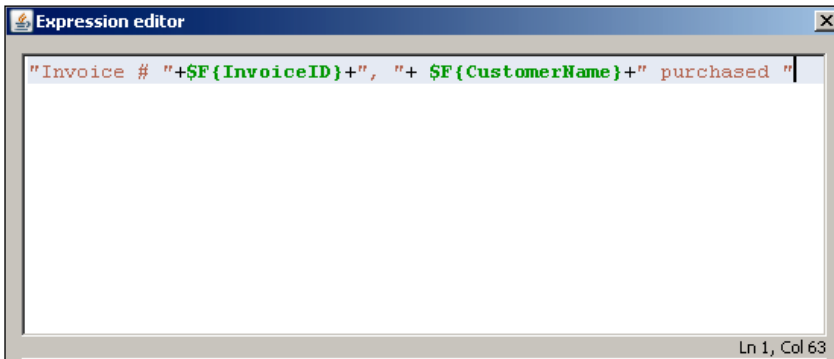
12. Right-click on the **Text Field** component of **Detail 1** section; a pop-up menu will appear. Select **Edit expression** from the pop-up menu. The **Expression editor** dialog will appear.
13. Type + at the end of the expression text. The expression will now become "Invoice # " + \$F{InvoiceID} + ", " + , as shown in the following screenshot:



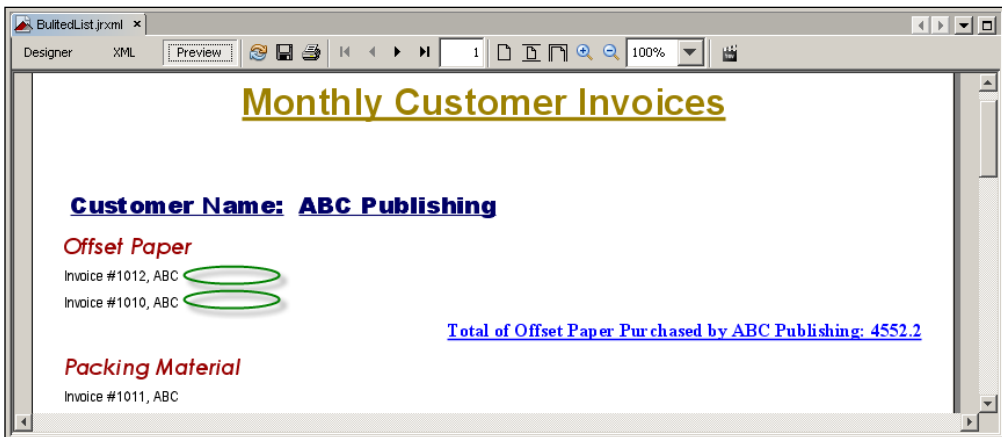
14. Select **Fields** from the first column of the lower half of the **Expression editor** window. This will list all the fields in the adjacent second column. Select **CustomerName** from the second column, and double-click on it. The expression will become "Invoice # " + \$F{InvoiceID} + ", " + \$F{CustomerName}.



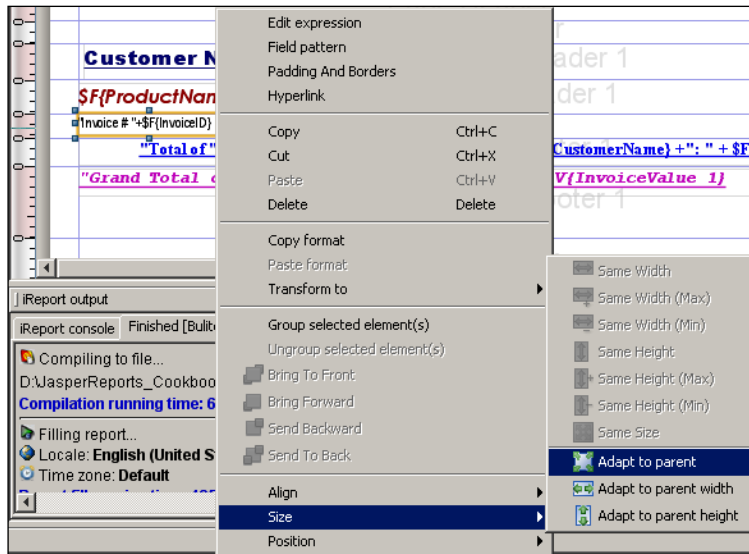
15. Click on the upper-half portion (a window where the expression text is visible) to edit the expression text. At the end of the expression text type "+
purchased". The expression will become "Invoice # "+\${InvoiceID}+", "
"+ \${CustomerName}+" purchased ".



16. Click the **Apply** button, the dialog will disappear.
17. Switch to the **Preview** tab. You will see that the result is not what you expect, that is, the text that you just typed in the **Expression editor** is not appearing in the preview.



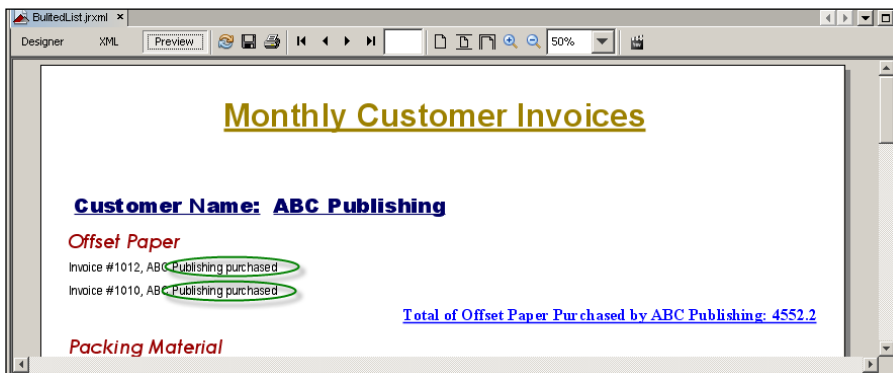
18. This is because the size of the **Text Field** component is not enough to accommodate the complete text. To increase the size, right-click on the **Text Field** component. A pop-up menu will appear; select **Size** from the pop-up menu. A sub-popup menu will appear; select the **Adapt to parent** option from the new pop-up menu.



19. The **Text Field** component will take all the available space of the **Detail 1** section.



20. Switch to the **Preview** tab. You will see this time that the result is as; that is, the text that you had typed in the **Expression editor** is now appearing in the preview, as shown in the following screenshot:



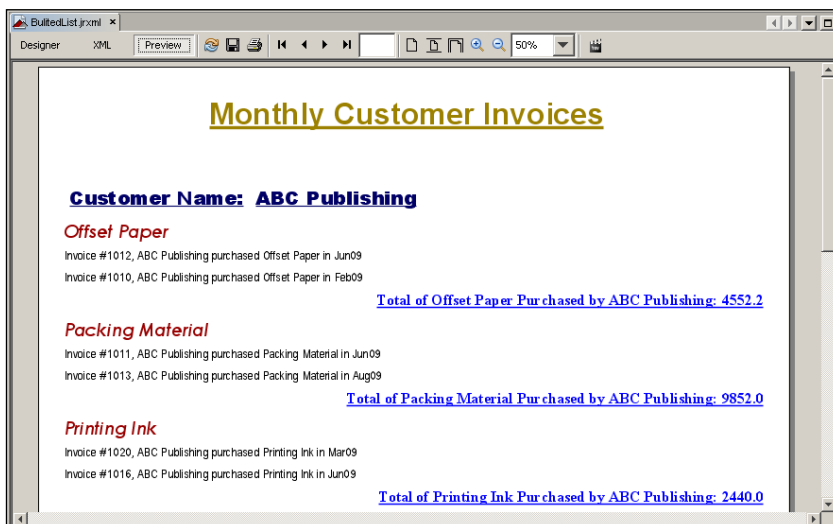
21. Switch back to the **Designer** tab; select the **Text Field** of the **Detail 1** section. Right-click on it and select **Edit expression** from the pop-up menu. The **Expression editor** dialog will appear.
22. While your cursor is in the **Expression editor**, type `+`. The expression will now become `"Invoice # "+$F{InvoiceID}+", "+$F{CustomerName}+" purchased "+`.
23. Select **Fields** from the first column of the lower-half of the **Expression editor** window. This will list all the fields in the adjacent second column. Select **ProductName** from the second column. Double-click on **ProductName**. The `$F{ProductName}` expression will be inserted into the expression text of the **Expression editor**.
24. Take your cursor at the end to the text in the **Expression editor** and type `+` in `+`. The expression in the **Expression editor** will change to `"Invoice # "+$F{InvoiceID}+", "+$F{CustomerName}+" purchased "+$F{ProductName}+" in "+` as shown in the following screenshot:



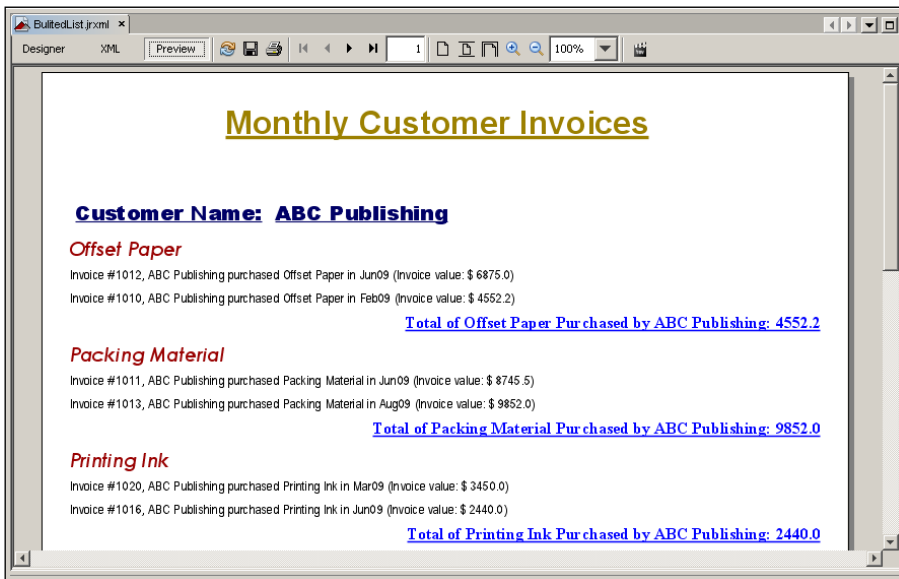
25. Select **Fields** from the first column of the lower-half of the **Expression editor** window. Select **InvoicePeriod** from the second column. Double-click on the **InvoicePeriod**; the `$F{InvoicePeriod}` expression will be inserted into the expression text of the **Expression editor**.



26. Click the **Apply** button. The **Expression editor** will disappear. Switch to **Preview** tab. The preview will be shown, as follows:

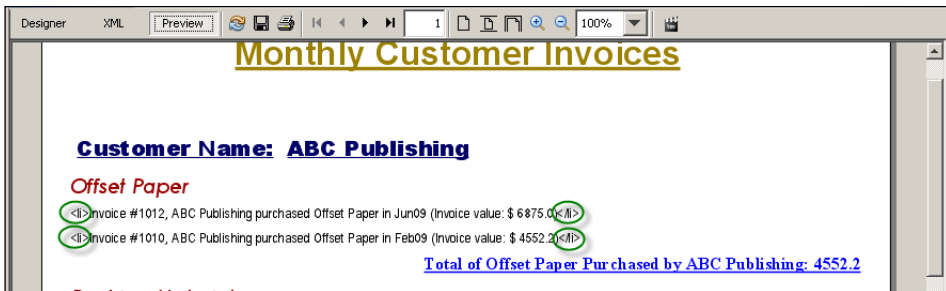


27. Switch back to the **Designer** tab. Select the **Text Field** of the **Detail 1** section. Right-click on it and select **Edit expression** from the pop-up menu. The **Expression editor** dialog will appear.
28. Type "+" (Invoice value: \\$ "+" at the end of the expression text.
29. Select **Fields** from the first column of the lower-half of the **Expression editor** window; select **InvoiceValue** from the second column. Double-click on **InvoiceValue**, and the `{InvoiceValue}` expression will be inserted into the expression text of the **Expression editor**.
30. Bring your cursor to the end of the text expression in the **Expression editor** and type `") "`. Now the final expression in the **Expression editor** will become `"Invoice # "+${InvoiceID}+", "+ ${CustomerName}+" purchased "+${ProductName}+" in "+${InvoicePeriod}+" (Invoice value: \$ "+${InvoiceValue}+") "`.
31. Click the **Apply** button, and the **Expression editor** will disappear. Switch to the **Preview** tab. The preview will be shown, as follows:

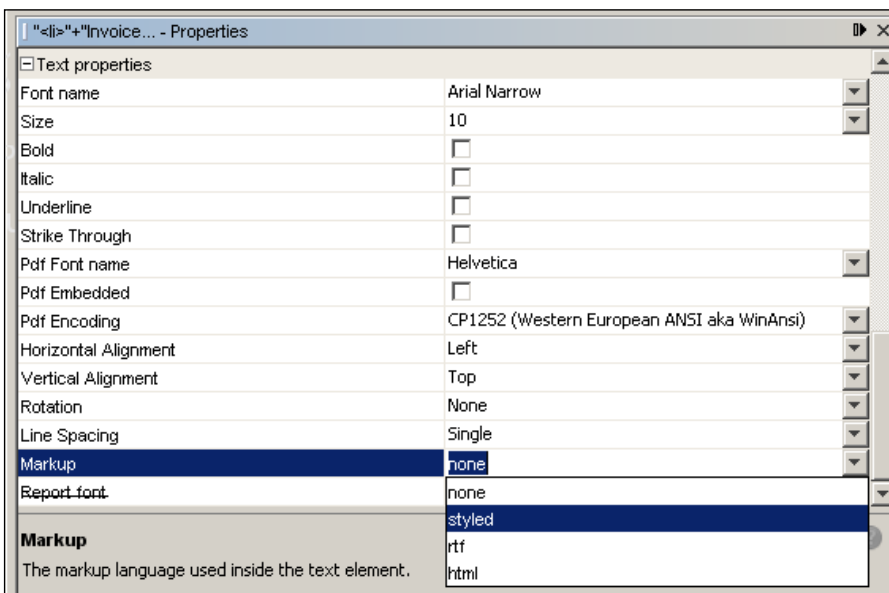


32. Now you are about to insert an HTML tag to convert this text into a bulleted list. To do so, switch back to the **Designer** tab. Again, right-click on the **Text Field** of the **Detail 1** section and select **Expression editor** from the pop-up menu.
33. An **Expression editor** dialog will appear. Bring your cursor to the beginning of the text and type `""+`.

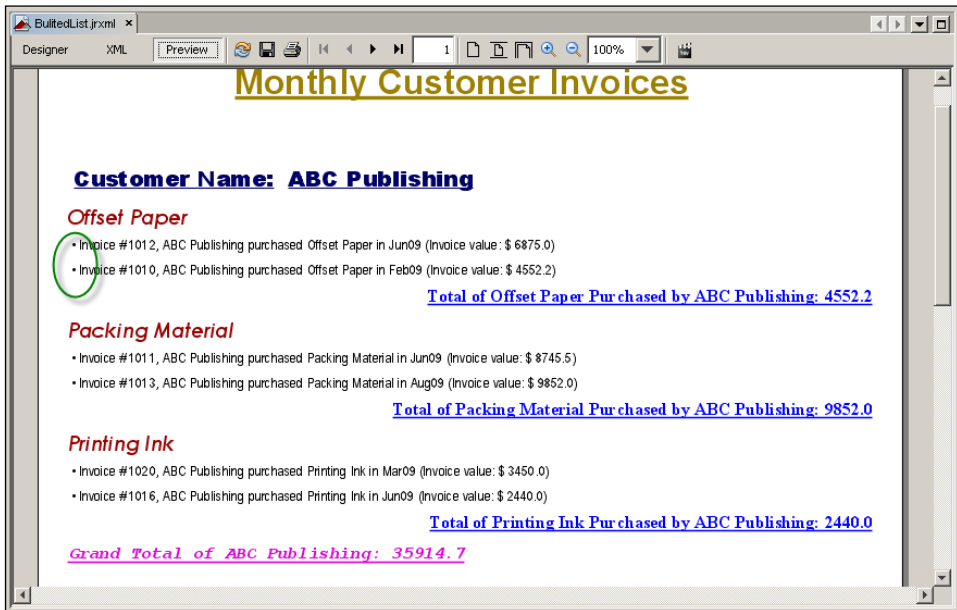
34. Take your cursor to the end of the expression and type ``. Now the complete expression will become `"+Invoice # "+${F{InvoiceID}}+",
"+ ${F{CustomerName}}+" purchased "+${F{ProductName}}+" in
"+${F{InvoicePeriod}}+" (Invoice value: \${ "+${F{InvoiceValue}}+")"+
`.
35. Click the **Apply** button, and the dialog will disappear. Switch to the **Preview** tab to see the resulting preview.



36. You will have noticed that the text is not yet bulleted. The HTML tags are being treated as text. To solve this problem, switch back to the **Designer** tab.
37. Select the **Text Field** component from the **Detail 1** section of your report and select the **Markup** property from the **Text properties** section of the **Properties** window below the **Palette**. Change its value to `styled`, as shown in the following screenshot:



38. Switch to the **Preview** tab. This time, you will notice that the HTML tags are working fine and the text is now bulleted.



39. Besides the `` tag, you can use many other HTML tags (for example, ``, `<i>`, `<u>`, and so on) on specific portions of text within a single text field to improve presentation. Now you will use the `` tag to show invoice numbers in bold style. Switch back to the **Designer** tab; right-click on the **Text Field** of the **Detail 1** section, and a pop-up menu will appear. Select the **Expression editor** option from the pop-up menu.
40. An **Expression editor** dialog will appear. Bring your cursor to the start of the text and press the right-arrow key of your keyboard seven times to bring your cursor just after "``" and just before "Invoice # and type "``". Press the right-arrow key of your keyboard 13 times to bring your cursor just after "Invoice # " and just before `$F{InvoiceID}` and type "``". Now the complete expression will become "``" + "``" + "Invoice # " + "``" + `$F{InvoiceID}` + ", " + `$F{CustomerName}` + " purchased " + `$F{ProductName}` + " in " + `$F{InvoicePeriod}` + " (Invoice value: \ \$ " + `$F{InvoiceValue}` + ") " + "``".

41. Click the **Apply** button, and the dialog will disappear. Switch to the **Preview** tab. You will notice, now that **Invoice #** is appearing in bold style, as shown in the next screenshot:

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

- **Invoice #** 010, ABC Publishing purchased Offset Paper in Feb09 (Invoice value: \$ 4552.2)
- **Invoice #** 1012, ABC Publishing purchased Offset Paper in Jun09 (Invoice value: \$ 6875.0)

Total of Offset Paper Purchased by ABC Publishing: 6875.0

Packing Material

How it works...

You have done three things in this recipe:

1. In steps 5 to 31, you converted the rows of a table into sentences. This is accomplished by concatenating the expressions of several text fields in the **Detail 1** section and inserting appropriate text to fill the gaps. While accomplishing this concatenation, you learned how to combine manual editing with graphical coding in an **Expression editor**.
2. Then, in steps 32 to 38 of the recipe, you used a HTML `` tag to display the sentence into a bulleted list. Here you also learned that HTML tags don't work unless you change the value of the **Markup** property to `styled`.
3. Later, in steps 39 to 41, you learned how to use some other HTML tags (such as a `` tag) to beautify your report text.

Expanding a field vertically to accommodate large text

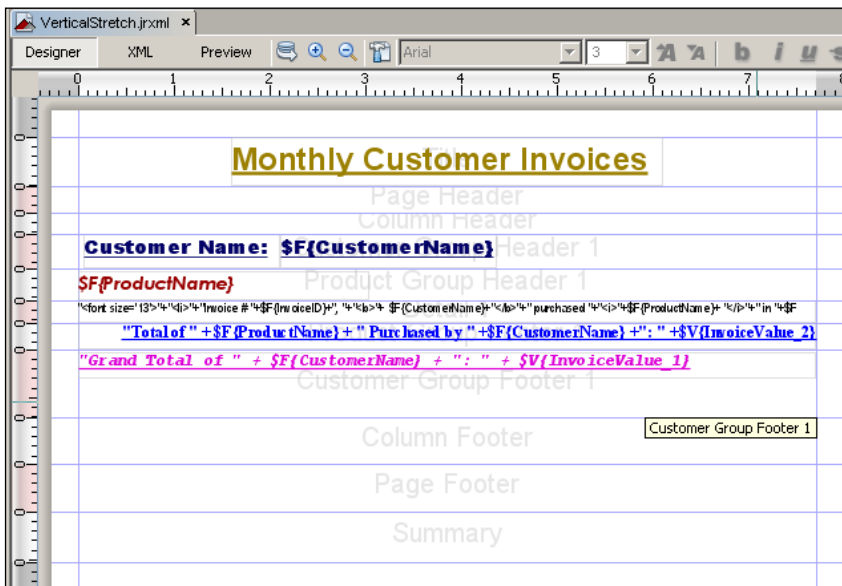
This recipe teaches you how to handle the presentation of dynamic data in text fields, which can contain large amounts of data. Large dynamic text scenarios occur when your report contains sentences, the exact word count of which is not known while designing a report and will be determined during report processing (for example, a sentence that contains the name and address of a company). In such situations, the large text size can overflow the size of the field and, therefore, wrapping to the next line may become essential.

Getting ready

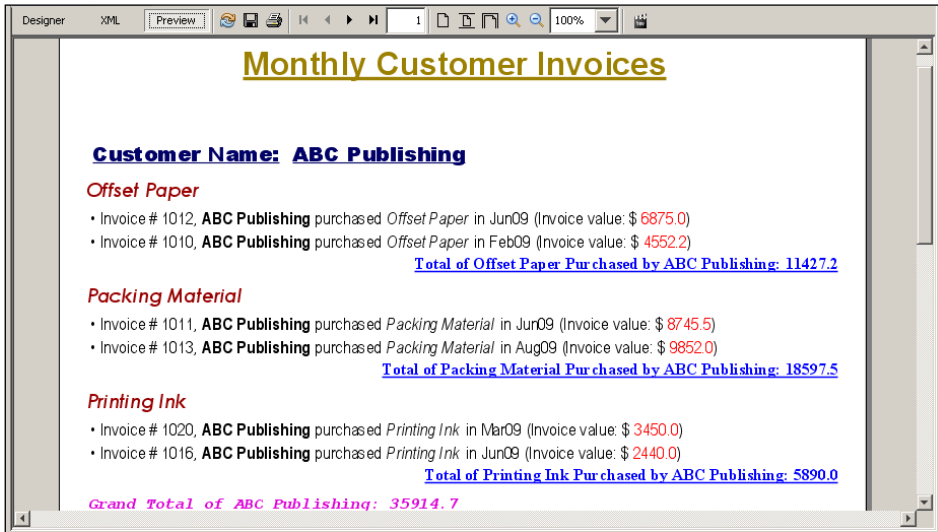
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

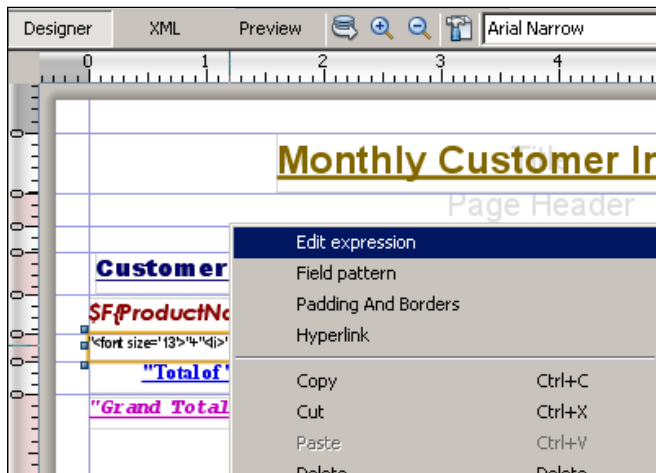
1. Open the `VerticalStretch.jrxml` file from the `Task4` folder in the source code for this chapter. The **Designer** tab of iReport shows a report, as follows:



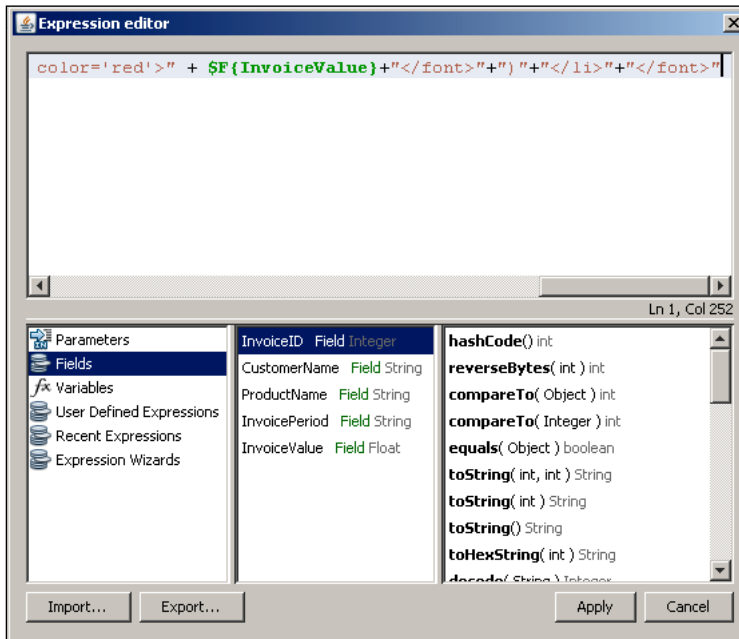
- Switch to the **Preview** tab to see a current preview of your report. You will see a report with bulleted lists of records grouped based upon customer names and product names, forming a nested hierarchy.



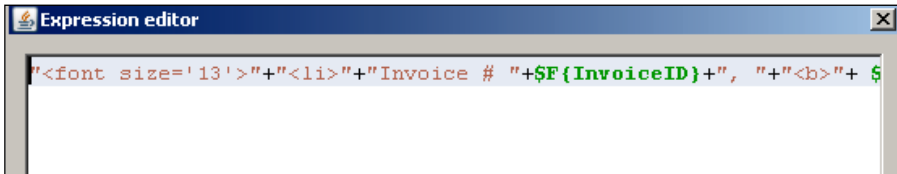
- Switch back to the **Designer** tab. Select the **Text Field** component in the **Detail 1** section. Right-click on it, and a pop-up menu will appear. Select **Edit expression** from the pop-up menu.



4. An **Expression editor** dialog will appear, as shown in the following screenshot:

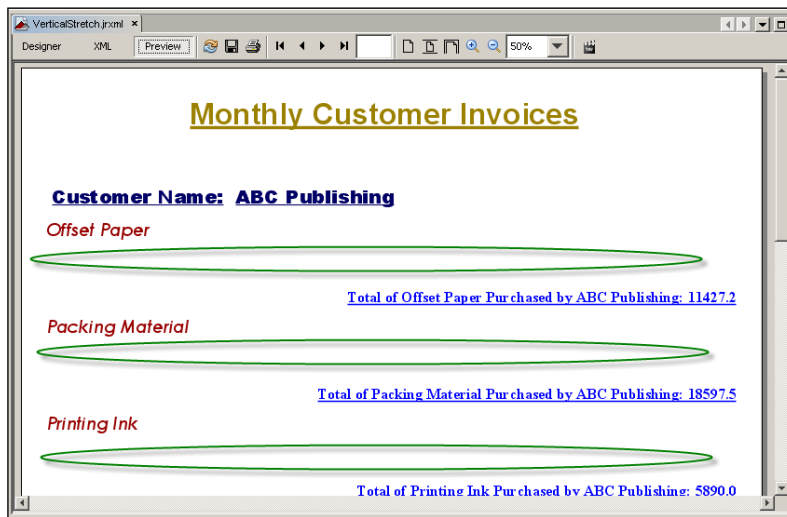


5. Take your cursor to the beginning of the expression by pressing the *Home* key on your keyboard. You will notice a `` tag at the beginning of the expression.

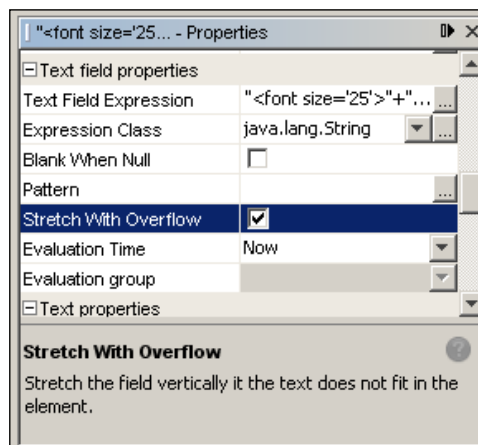


6. Change the value of the `size` parameter of the `` tag from 13 to 25. Click on the **Apply** button. The **Expression editor** will disappear.

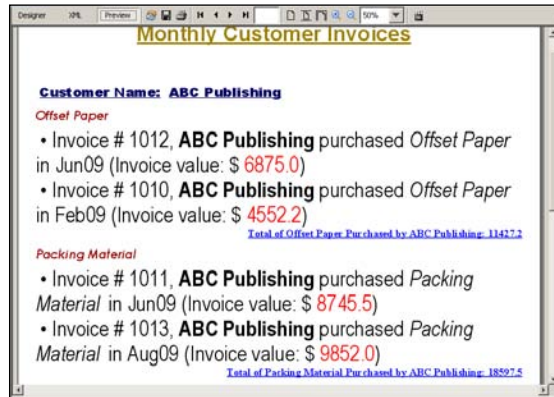
7. Switch to the **Preview** tab. The report will look as follows:



8. Notice that the preview output is not as we expected. The text of the **Detail 1** section **Text Field** is missing in the preview, (We were expecting that the text would appear larger than before.)
9. This problem is usually faced when we use a smaller text field to display a larger text. To solve this problem JasperReports offers a feature of dynamically stretching a text field with overflow of text. To use this feature, switch back to the **Designer** tab.
10. Select the **Text Field** in the **Detail 1** section. Select the **Stretch With Overflow** property from the **Text field properties** section of the **Properties** window below the **Palette**. Tick the checkbox beside it, as shown in the following screenshot:



11. Switch to the **Preview** tab. The report will look as shown below. Notice that the preview output is now as we expected. The text of the **Detail 1** section **Text Field** is appearing in a very large size in the preview.



How it works...

This recipe works because JasperReports allows us to use stretchable fields, which it automatically stretches in a vertical fashion when it sees that data in the field cannot be accommodated within one line. If you switch to the XML tab, you will see the following JRXML code for the text field that you made stretchable in this recipe:

```
<textField isStretchWithOverflow="true">
```

You can see the `isStretchWithOverflow="true"` attribute in the `<textField>` tag. This attribute makes your text fields stretchable. iReport authored this attribute in response to step 9 of this recipe.

Applying formatting pattern to the value of a data field

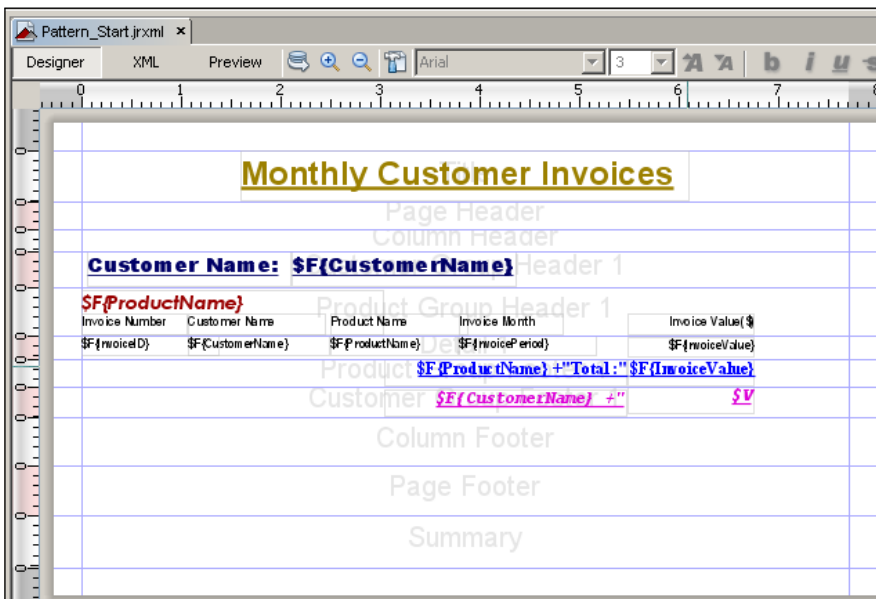
Several components of a report may require some sort of formatting of data coming from the database. For example, companies normally have specific formats for identifiers or numeric values used in their reports. JasperReports allows you to incorporate such formatting and this recipe teaches you how to format identifiers and numeric values.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

1. Open the `Pattern.jrxml` file from the `Task5` folder in the source code for this chapter. The **Designer** tab of iReport shows a report, as follows:



- Switch to the **Preview** tab to see a current preview of your report. You will see a report with invoices grouped based upon customer names. For each customer, you will notice that invoices are grouped based upon product names, forming a nested hierarchy. In this way, the report categorizes all invoices by product for each customer. Also notice that the **Invoice Value** and **Invoice Number** fields, which show simple data (without any specific pattern).

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value (\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
Offset PaperTotal:				4552.2

Packing Material

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value (\$)
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing MaterialTotal:				9852.0

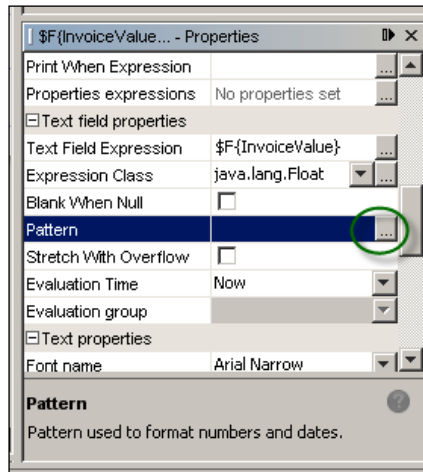
Printing Ink

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value (\$)
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Printing InkTotal:				2440.0
ABC PublishingTotal:				35914.7

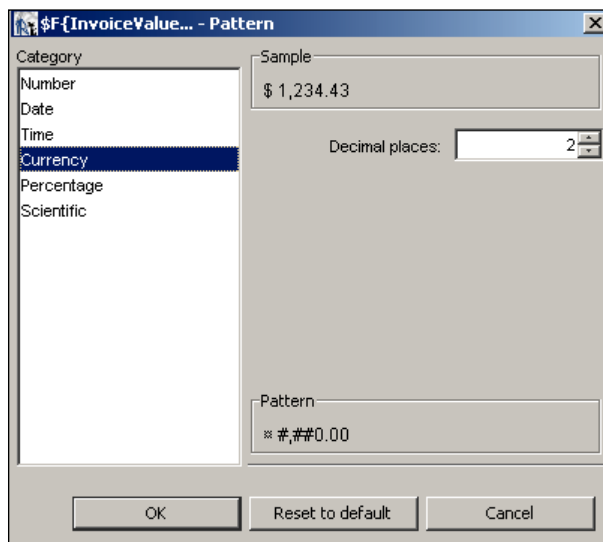
Customer Name: Alice

- Now you will apply a currency formatting pattern to the invoice value of the records from the nested group. Switch back to the **Designer** tab.
- Select the **Text Field** component in the **Detail 1** section, having the expression `$F{InvoiceValue}`.

- While the **Text Field** component from step 4 is still selected, select the **Pattern** property from the **Text field properties** section of the **\$F{InvoiceValue...-Properties}** window below the **Palette**. Click the button beside this property, as shown in the next screenshot:



- A **\$F{InvoiceValue...-Pattern}** dialog will appear. Select **Currency** from the **Category** column and click the **OK** button.



- Switch to the **Preview** tab to see the effect of steps 4, 5, and 6.

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	\$ 6,875.00
1010	ABC Publishing	Offset Paper	Feb09	\$ 4,552.20
Offset Paper Total :				4552.2

Packing Material

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	\$ 8,745.50
1013	ABC Publishing	Packing Material	Aug09	\$ 9,852.00
Packing Material Total :				9852.0

Printing Ink

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1020	ABC Publishing	Printing Ink	Mar09	\$ 3,450.00
1016	ABC Publishing	Printing Ink	Jun09	\$ 2,440.00
Printing Ink Total :				2440.0
ABC Publishing Total :				35914.7

Customer Name: Alice

- Similarly, after switching back to the **Designer** tab, select the **Text Field** component in the **Product Group Footer 1** section having the expression `$V{InvoiceValue_2}`, and repeat steps 5 and 6. Then select the **Text Field** component in the **Customer Group Footer 1** section having the expression `$V{InvoiceValue_1}`, and repeat steps 5 and 6.
- Switch to the **Preview** tab. Now the report will look as shown in the following screenshot:

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	\$ 6,875.00
1010	ABC Publishing	Offset Paper	Feb09	\$ 4,552.20
Offset Paper Total :				\$ 11,427.20

Packing Material

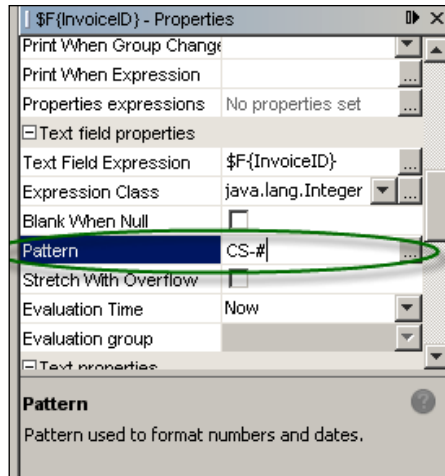
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1011	ABC Publishing	Packing Material	Jun09	\$ 8,745.50
1013	ABC Publishing	Packing Material	Aug09	\$ 9,852.00
Packing Material Total :				\$ 18,597.50

Printing Ink

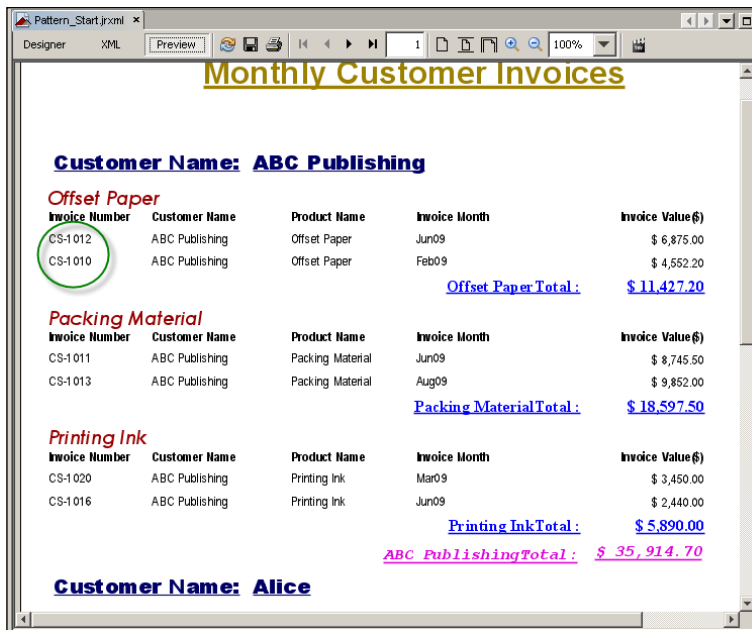
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1020	ABC Publishing	Printing Ink	Mar09	\$ 3,450.00
1016	ABC Publishing	Printing Ink	Jun09	\$ 2,440.00
Printing Ink Total :				\$ 5,890.00
ABC Publishing Total :				\$ 35,914.70

Customer Name: Alice

10. Select the **Text Field** component in the **Detail 1** section having the expression `$F{InvoiceID}`.
11. While the **Text Field** component of step 10 is still selected, select the **Pattern** property from the **Text field properties** section of the **\$F{InvoiceID}-Properties** window below the **Palette**. Type the text `CS-#` in the space beside this property, as shown in the next screenshot:



12. Switch to the **Preview** tab to see the effect of step 11.



Using background images and watermarks in your report

Sometimes, you may want to present your report with a background image spanning the entire report, such as a company theme image or a watermark. Such background images are frequently used by companies for several purposes, such as to mark the status of a document (for example, whether the document is a draft or an approved version) or to avoid tampering of a document.

In this recipe, you will learn how to insert a background image into your report.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb3` and copy sample data for this recipe into the database.

How to do it...

1. Open the `Background_Images.jrxml` file from the `Task6` folder in the source code for this chapter. The **Designer** tab of iReport shows a report, as shown in the following screenshot:

<u>Monthly Customer Invoices</u>	
Page Header	
Column Header	
<u>Customer Name:</u>	<u>$\\$F\{CustomerName\}$</u>
Header 1	
<u>$\\$F\{ProductName\}$</u>	Product Group Header 1
$\text{"< >"} \times \text{"< >"} \times \text{"1"} \text{invoice \# " + } \$F\{\text{InvoiceID}\} \text{, " + } \$F\{\text{CustomerName}\} \text{ "purchased " + } \$F\{\text{ProductName}\} \text{ " in " + } \$F\{\text{InvoicePeriod}\} \text{ " (Invoice value: } \$F\{\text{InvoiceValue}\} \text{)"} \times \text{"< >"} \times \text{"1"} \text{ "Total of " + } \$F\{\text{ProductName}\} \text{ + " Purchased by " + } \$F\{\text{CustomerName}\} \text{ + ": " + } \$F\{\text{InvoiceValue}\}$	
<u>$\text{"Grand Total of " + } \\$F\{\text{CustomerName}\} \text{ + ": " + } \\$F\{\text{InvoiceValue 1}\}$</u>	
Customer Group Footer 1	

- Switch to the **Preview** tab. You will see a report with invoices grouped based upon customer names. For each customer, you will notice that invoices are grouped based upon product names, forming a nested hierarchy. In this way, the report categorizes all invoices by product for each customer.

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

- **Invoice # 1010**, ABC Publishing purchased Offset Paper in Feb09 (Invoice value: \$ 4552.2)
- **Invoice # 1012**, ABC Publishing purchased Offset Paper in Jun09 (Invoice value: \$ 6875.0)

Total of Offset Paper Purchased by ABC Publishing: 6875.0

Packing Material

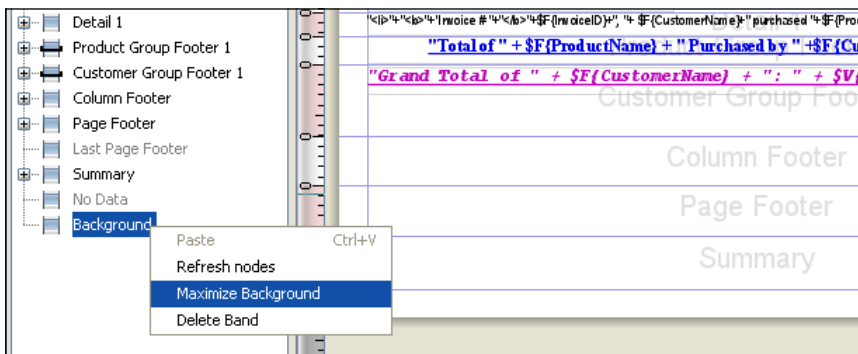
- **Invoice # 1011**, ABC Publishing purchased Packing Material in Jun09 (Invoice value: \$ 8745.5)
- **Invoice # 1013**, ABC Publishing purchased Packing Material in Aug09 (Invoice value: \$ 9852.0)

Total of Packing Material Purchased by ABC Publishing: 9852.0

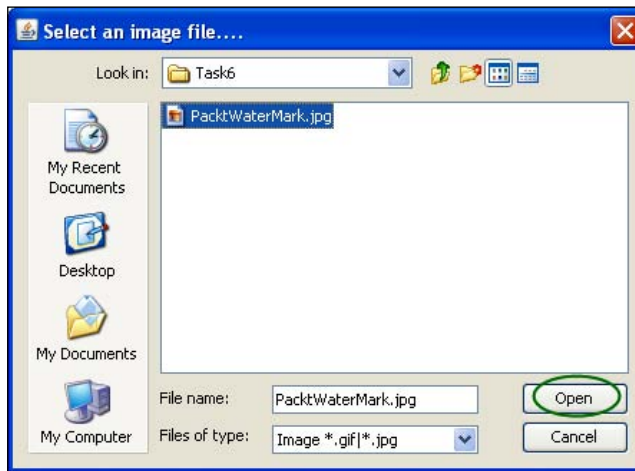
Printing Ink

- **Invoice # 1016**, ABC Publishing purchased Printing Ink in Jun09 (Invoice value: \$ 2440.0)

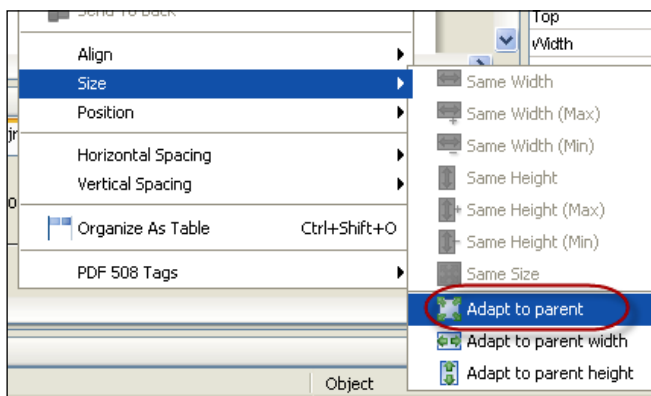
- Now you will insert an image into your report, which will serve as the background. Switch back to the **Designer** tab.
- The left-most part of the main iReport screen is a **Report Inspector** window, which shows a tree with many nodes. The last node is named **Background**. Right-click on the **Background** node, and a pop-up menu will appear. Select the **Maximize Background** option from the pop-up menu, as shown in the next screenshot. A new section named **Background** will appear below your report in the **Designer** tab.



5. Drag-and-drop an **Image** component from the **Palette** into the **Background** section of your report. The **Select an image file....** dialog will appear, from where you will browse to a background image named `PacktWaterMark.jpg`, which you will find in the `Task6` folder of the source code for this chapter. After selecting the image file, click on **Open**. You will see a picture box with a background image added into the background section.



6. To align the image component in the **Background** section, right-click on the **Image** component. A pop-up menu will appear; select the **Size** option from the pop-up menu, and a sub pop-up menu will appear. Select the **Adapt to parent** option from the sub pop-up menu, as shown in the following screenshot. The **Image** component will occupy all the available space in the **Background** section.



7. Switch to the **Preview** tab. You will see a report with a background picture, as shown in the next screenshot. Note that the image you see in the background of your report can also serve as a watermark.

Monthly Customer Invoices

Customer Name: ABC Publishing

Offset Paper

- **Invoice #1010**, ABC Publishing purchased Offset Paper in Feb09 (Invoice value: \$ 4552.2)
- **Invoice #1012**, ABC Publishing purchased Offset Paper in Jun09 (Invoice value: \$ 6875.0)

Total of Offset Paper Purchased by ABC Publishing: 6875.0

Packing Material

- **Invoice #1011**, ABC Publishing purchased Packing Material in Jun09 (Invoice value: \$ 8745.5)
- **Invoice #1013**, ABC Publishing purchased Packing Material in Aug09 (Invoice value: \$ 9852.0)

Total of Packing Material Purchased by ABC Publishing: 9852.0

Printing Ink

- **Invoice #1016**, ABC Publishing purchased Printing Ink in Jun09 (Invoice value: \$ 2440.0)
- **Invoice #1020**, ABC Publishing purchased Printing Ink in Mar09 (Invoice value: \$ 3450.0)

Total of Printing Ink Purchased by ABC Publishing: 3450.0

Grand Total of ABC Publishing: 35914.7

4

Working with a Variety of Data Sources

In this chapter, you will learn:

- ▶ Creating a report from relational data
- ▶ Connecting to an XML datasource
- ▶ Creating a report from XML data using XPath
- ▶ Using multiple relational databases to generate a report
- ▶ Creating a report from model beans of Java applications

Introduction

This chapter is all about data sources. Application data resides in a variety of sources such as relational databases, XML data sources, and Java objects (model beans or Plain Old Java Objects). iReport will need to access these data sources in order to generate a report.

iReport uses the concept of **loose coupling** between data sources and report design. Loose coupling means you can design your reports independent of the type of data source used. This means the same report design can work with any data source. You will experience the advantage of this loose coupling while executing the recipes of this chapter.

Relational databases are perhaps the most popular data sources used to hold application data. XML is another common data source, which is usually used to exchange data between applications.

I cover both of these in this chapter.

Another common data source is JavaBeans, which holds data at runtime. You may also need to configure iReport to work with JavaBeans. So this chapter also covers JavaBeans as a data source.

Creating a report from relational data

This recipe shows how you will connect JasperReports to your database. I am using an open source database named PostgreSQL to hold the sample data of almost all recipes of this cookbook. That's why in this recipe you will connect iReport with your PostgreSQL installation. This recipe also shows that you can connect iReport to any of the popular databases in a similar manner.

Getting ready

You will need PostgreSQL to follow this recipe. Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter, which shows how you will install and run PostgreSQL. Note that your installation of PostgreSQL should be up and running before you proceed.

The source code for this chapter also includes a file named `CreateDbIntoPGS.txt`, which will help you to create a database named `jasperdb5`.

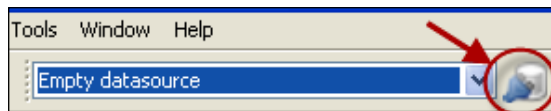
How to do it...

The following simple steps will show you how to connect iReport to a database:

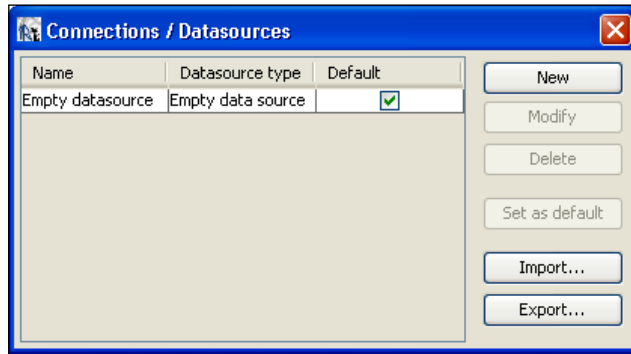
1. Run iReport; it will open with a **Welcome Window**, as shown in the following screenshot:



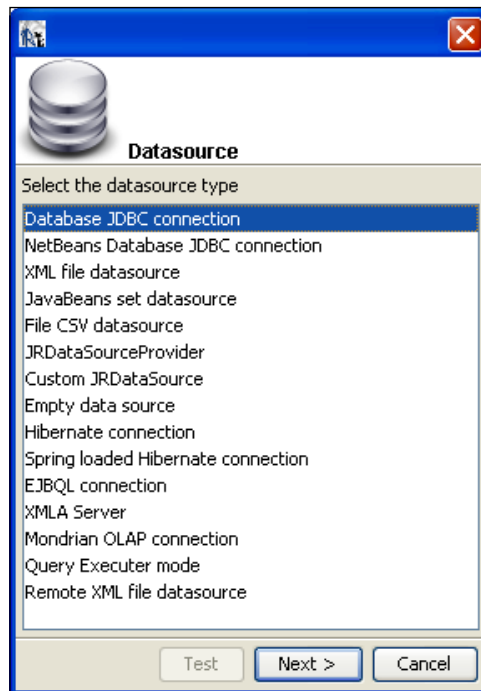
2. If you have not made any database connection so far in your iReport installation, you will see an **Empty datasource** shown selected in a drop-down list just below the main menu. Click on the **Report Datasources** icon shown encircled to the right of the drop-down list, as shown in the following screenshot:



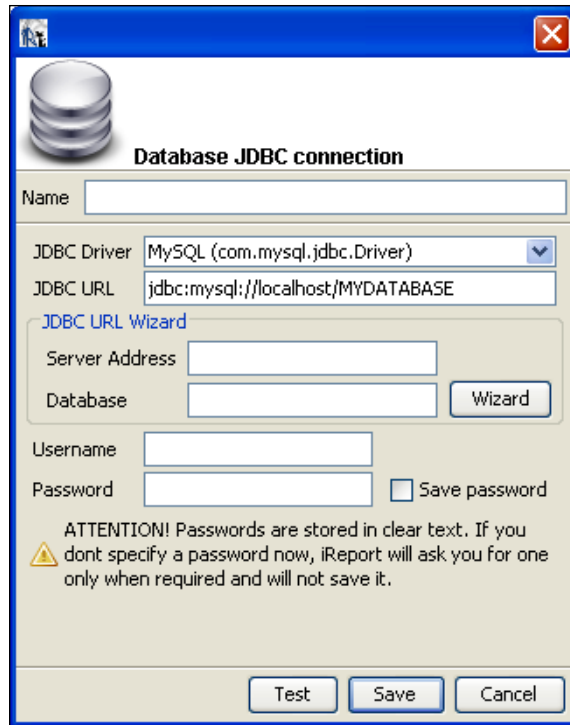
3. A new window named **Connections / Datasources** will open, as shown in the following screenshot. This window lists an **Empty datasource** as well as the datasources you have made so far.



4. Click the **New** button shown at the top right of the **Connections / Datasources** window. This will open a new **Datasource** selection window, as shown in the following screenshot:



5. You will see **Database JDBC connection** is selected by default. Click the **Next** button at the bottom of the **Datasource** window.
6. A new window named **Database JDBC connection** will open, as shown in the following screenshot:

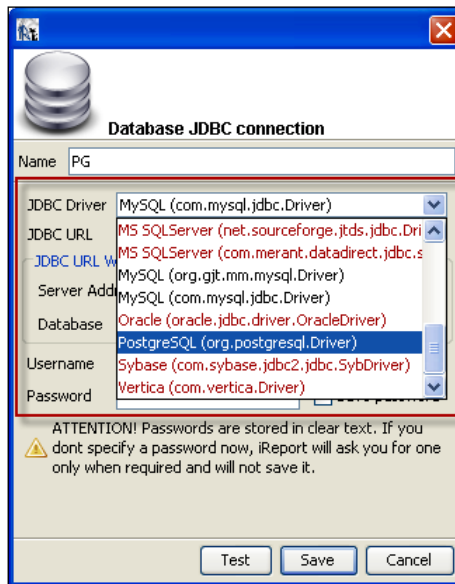


7. Enter **PG** as the name for your new database connection in the input box beside the **Name** field.

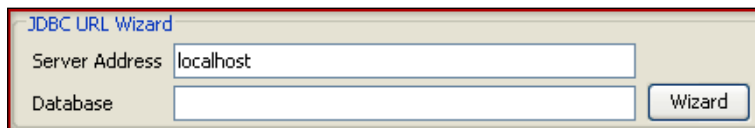


PG is just a name for the database connection you are creating. You can give any name and create any number of database connections.

8. Click on the **JDBC Driver** drop-down list; it will drop-down to show a list of available JDBC drivers. As you are connecting to the PostgreSQL database, select the **PostgreSQL (org.postgresql.Driver)** option from the drop-down list, as shown in the following screenshot:

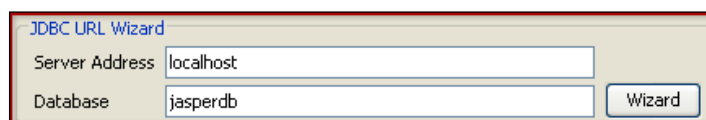


9. Enter **localhost** as the name of the server hosting your database in the input box beside the **Server Address** field, as shown in the following screenshot:



Note that here I am assuming your PostgreSQL is installed on the same PC as you are running iReport. If PostgreSQL is installed over a network, you will enter the IP address of the machine hosting PostgreSQL instead of **localhost**.

10. Enter **jasperdb5** as the name of the database instance created in the *Getting ready* section in the **Database** field, as shown in the following screenshot:





The name of the database instance will be different depending upon which database instance you are using. For this recipe you are using a database named **jasperdb5** (which you created earlier in the *Getting ready* section). But when you are executing some other recipe, you will enter the name of the database instance that contains sample data for that particular recipe.

11. Click the **Wizard** button available to the right of the **Database** text box. This will update the **JDBC URL** field, as shown in the following screenshot:

JDBC URL	jdbc:postgresql://localhost:5432/jasperdb
----------	---

12. Provide **postgres** as the value of the **Username** and **Password** fields, as shown in the following screenshot:

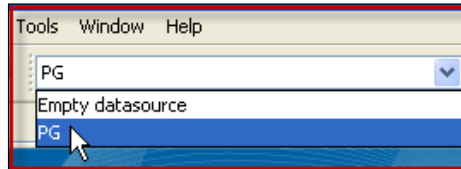
Username	postgres	<input type="checkbox"/> Save password
Password	••••••	

13. Check the **Save password** checkbox beside the **Password** field shown in step 12 to save your database password permanently for reports generated using iReport.
14. Click the **Test** button to test the new database connection. In the case of a successful connection, you will see a **Connection test successful** message in a dialog window. Dismiss the message by clicking the **OK** button.
15. Click the **Save** button to save the newly created connection. You will see that the **Connections / Datasources** window will open showing your new connection set as the default connection in the connections list, as shown in the following screenshot:

Name	Datasource type	Default
Empty datasource	Empty data source	<input type="checkbox"/>
PG	Database JDBC conn...	<input checked="" type="checkbox"/>

Buttons: New, Modify, Delete, Set as default, Import..., Export...

16. Now you are all set to generate a report from the data contained in your PostgreSQL installation. Whenever you are opening or viewing a report based on data contained in PostgreSQL, just make sure that **PG** is selected in the datasources drop-down list, as shown in the following screenshot:



There's more...

You have learned how to connect iReport to PostgreSQL. If you have some other database, you just need to select the appropriate JDBC driver from the **JDBC Driver** drop-down list in step 8 of the recipe.

iReport comes bundled with drivers for the following open source databases:

- ▶ MySQL
- ▶ PostgreSQL
- ▶ HSQLDB

If you are not using an open source database, you will need to separately download and install the relevant JDBC drivers.

Connecting to an XML datasource

XML is a popular data source used in many applications. JasperReports allows you to generate reports directly from XML data.

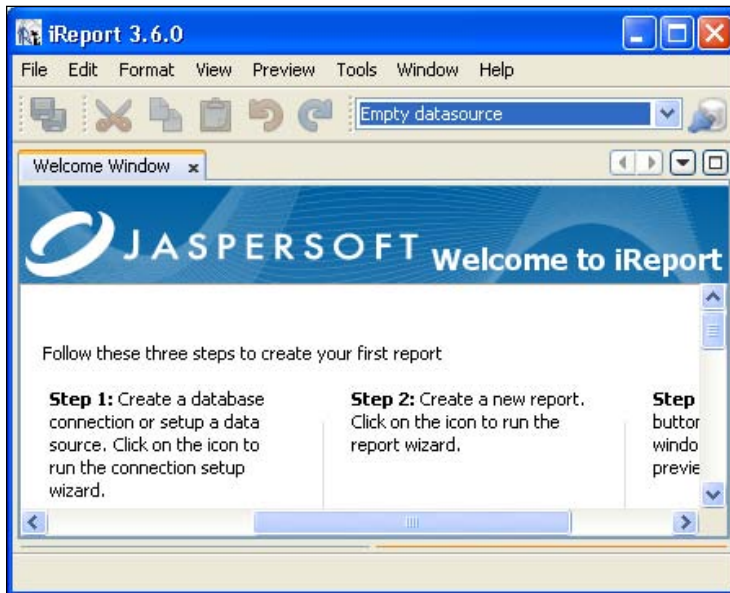
This recipe teaches you how to connect iReport to an XML file stored on your PC.

Getting ready

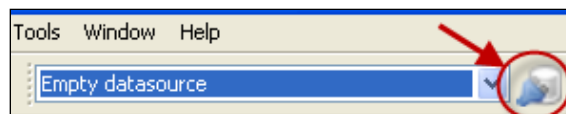
You need an XML file that contains report data. The `EventsData.xml` file is contained in the source code for this chapter. Unzip the source code file for this chapter and copy the `Task2` folder from the unzipped source code to a location of your choice.

How to do it...

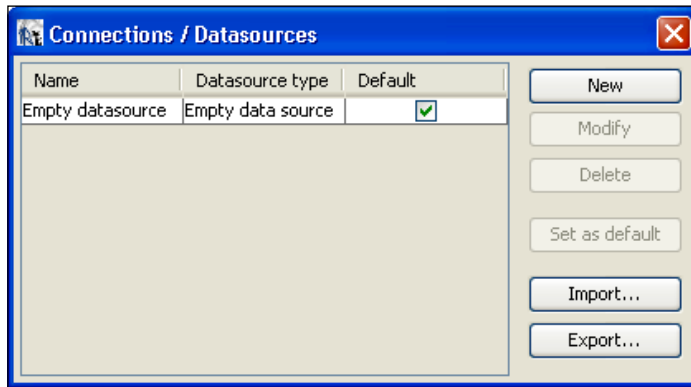
1. Run iReport; it will open showing a **Welcome Window**, as shown in the following screenshot:



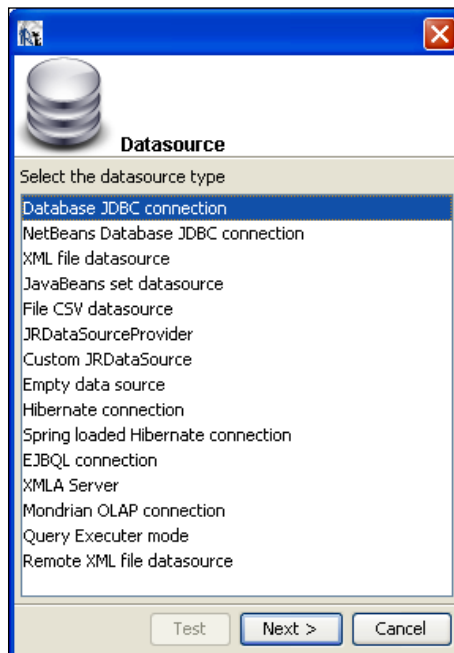
2. If you have not made any database connection so far in your iReport installation, you will see an **Empty datasource** shown selected in a drop-down list just below the main menu. Click on the **Report Datasources** icon shown encircled to the right of the drop-down list in the screen-shot shown below:



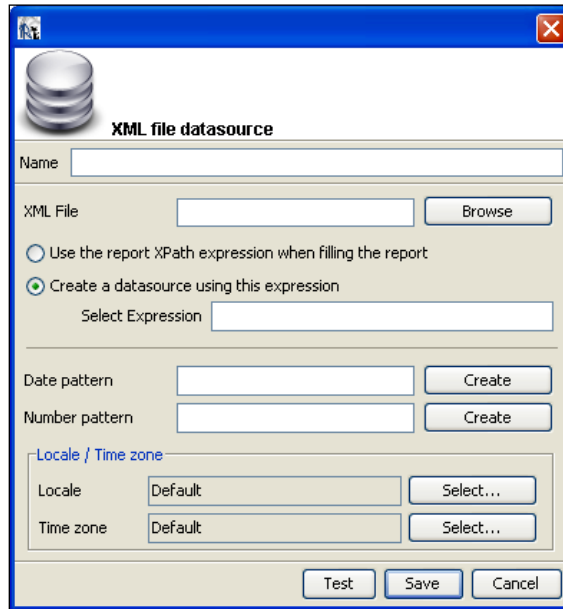
3. A new window named **Connections / Datasources** will open, as shown below. This window lists an **Empty datasource** as well as the data sources you have made so far.



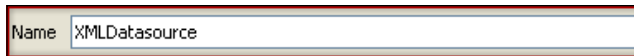
4. Click the **New** button at the top-right of the **Connections / Datasources** window. This will open a new **Datasource** selection window, as shown in the following screenshot:



5. Select **XML file datasource** from the datasources type list. Click **Next**.
6. A new window named **XML file datasource** will open, as in the following screenshot:



7. Enter **XMLDatasource** as the name for your new connection for the XML datasource in the text box beside the **Name** text field, as shown in the following screenshot:



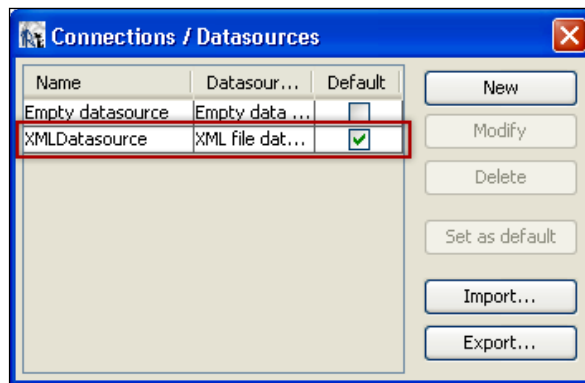
8. Click the **Browse** button beside the XML file text box to browse to the EventsData.xml file located in the Task2 folder that you copied in the *Getting ready* section. Click the **Open** button, as shown in the following screenshot:



9. Select the **Use the report XPath expression when filling the report** option in the **XML file datasource** window, as shown in the following screenshot:



10. Leave the other fields at their default values. Click the **Test** button to test the new XML datasource connection. You will see a **Connection test successful** message dialog.
11. Click the **Save** button to save the newly created connection. A **Connections / Datasources** window will open showing your new XML datasource connection set as the default connection in the connections list, as shown highlighted in the following screenshot:



Creating a report from XML data using XPath

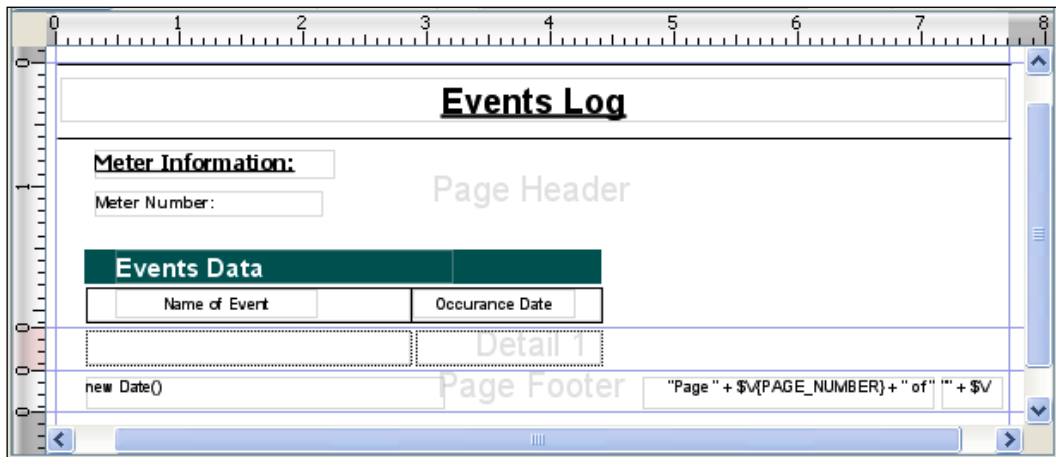
In this recipe, you will create a report from data stored in an XML file. In order to process an XML file and extract information from it, JasperReports uses XPath, which is a popular query language to filter XML data. So you will also learn how to use XPath expressions for report generation.

Getting ready

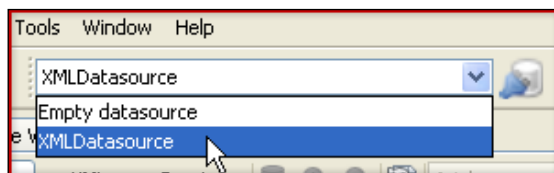
You need to connect your iReport installation to your XML file before starting this recipe. Refer to the *Connecting to XML datasource* recipe of this chapter to learn how to connect iReport to your XML file.

How to do it...

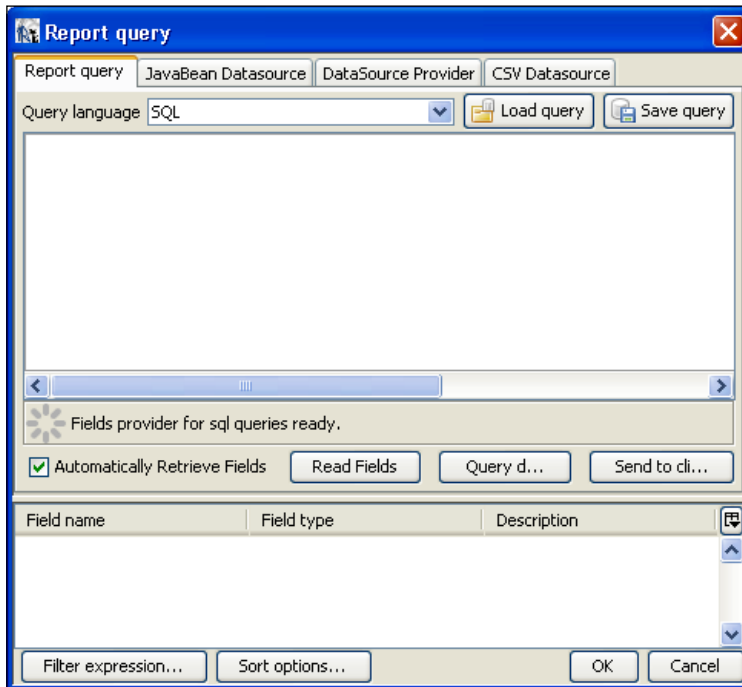
1. Open the `XMLDatasource.jrxml` file from the `Task3` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Page Header**, **Detail 1**, and **Page Footer** sections, as shown in the following screenshot:



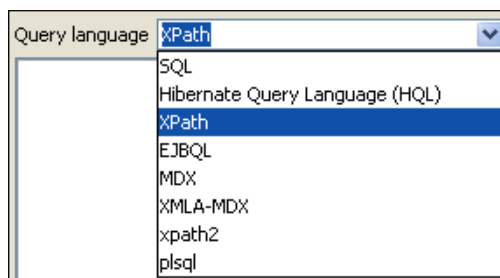
2. Select **XMLDatasource** from the drop-down list just below the main menu of iReport, as shown in the following screenshot:



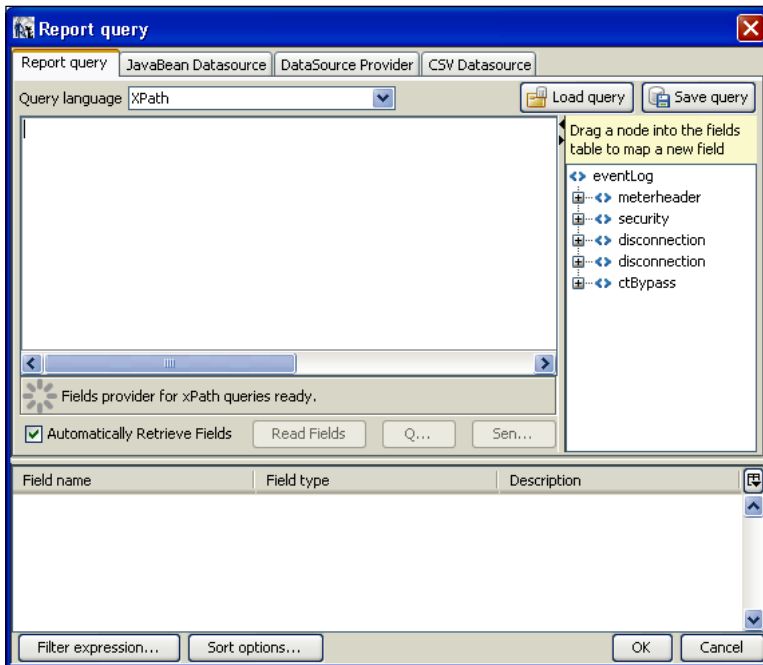
- Click the **Report query** button on the right of the **Preview** tab. A **Report query** dialog will appear, as shown in the following screenshot:



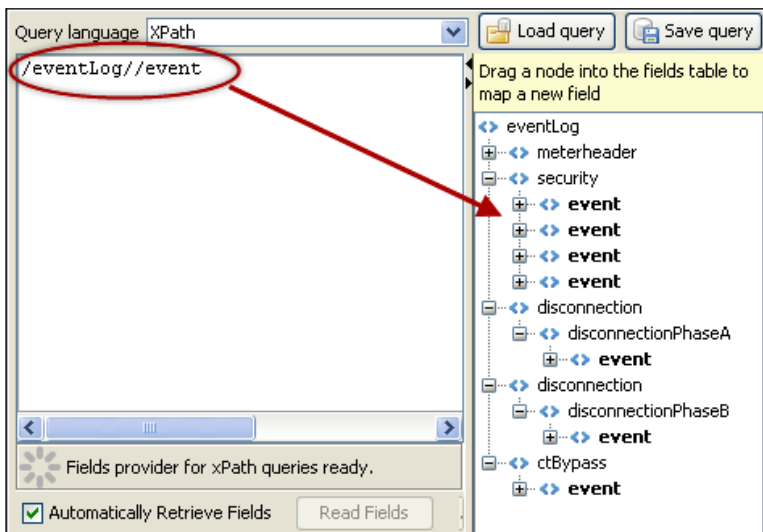
- You will see a **Query language** drop-down list at the top of the **Report query** tab. Select **XPath** as the query language from the drop-down list, as shown in the following screenshot:



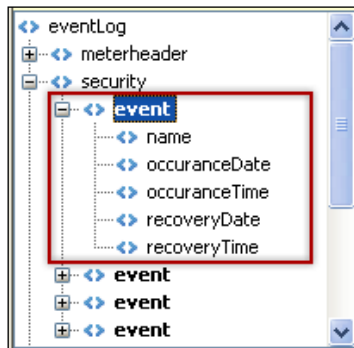
- The text area below the **Query language** drop-down list will split into two parts. The left part shows the actual **XPath** expression, whereas the right part shows XML nodes, as shown next. Expand the **security** node from the **eventLog** tree in the right part.



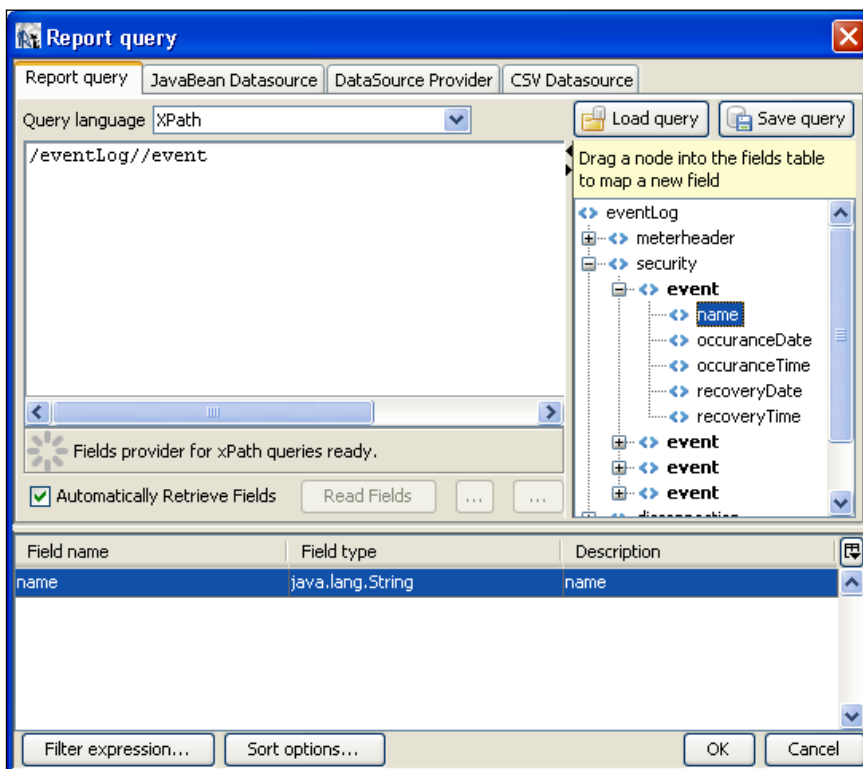
- Type `/eventLog//event` as the **XPath** expression. All **event** nodes will appear in bold, as shown in the following screenshot:



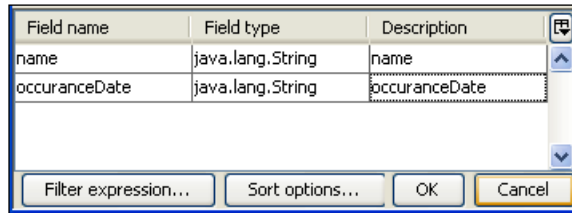
7. Add the child nodes of the first **event** node as fields to the report. For this purpose, double-click on the first **event** node, it will expand to show its child nodes, as shown in the following screenshot:



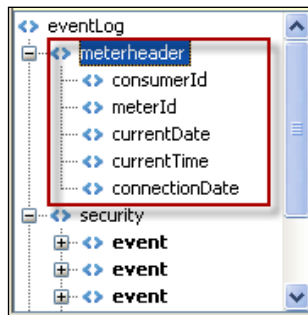
8. Right-click on the **name** child of the event node. A pop-up menu will appear; select the **Add node as field** option. This will add **name** as a field in the lower-half of the **Report query** tab, as shown in the following screenshot:



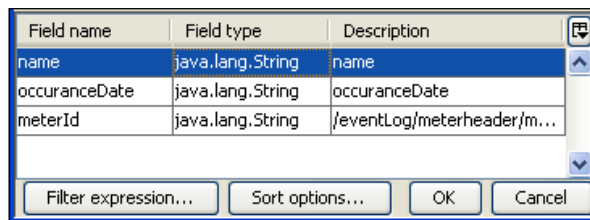
9. Similarly, repeat step 8 for the **occurrenceDate** child of the **event** node. The lower-half of the **Report query** tab will be updated to show the two children of the **event** node, as shown in the following screenshot:



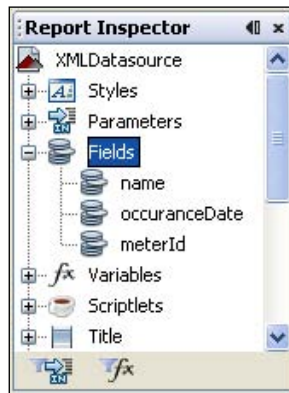
10. Double-click on the **meterheader** child of the **eventLog** node. It will expand to show its children, as shown in the following screenshot:



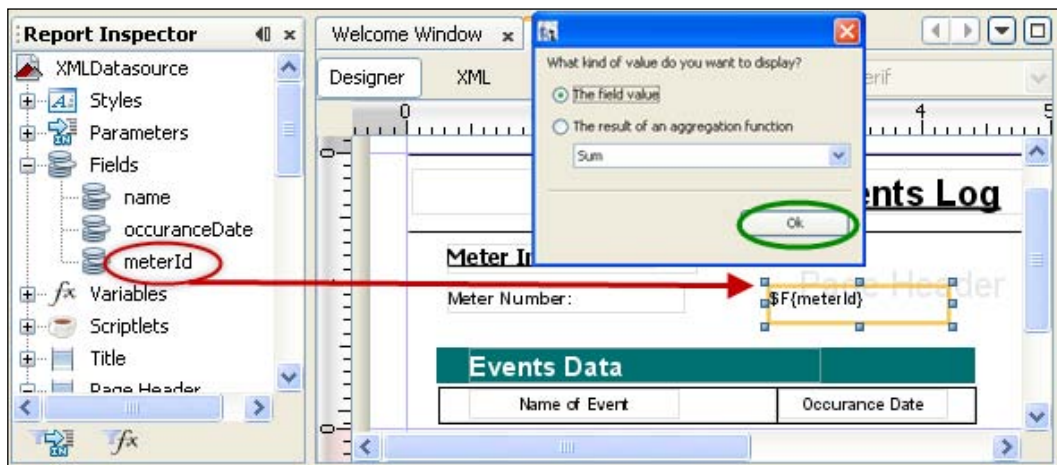
11. Repeat step 8 for the **meterId** child of the **meterheader** node. Finally, the lower-half of the **Report query** tab will show the **name**, **occurrenceDate**, and **meterId** fields, as shown next. Click the **OK** button at the bottom to dismiss the **Report query** dialog.



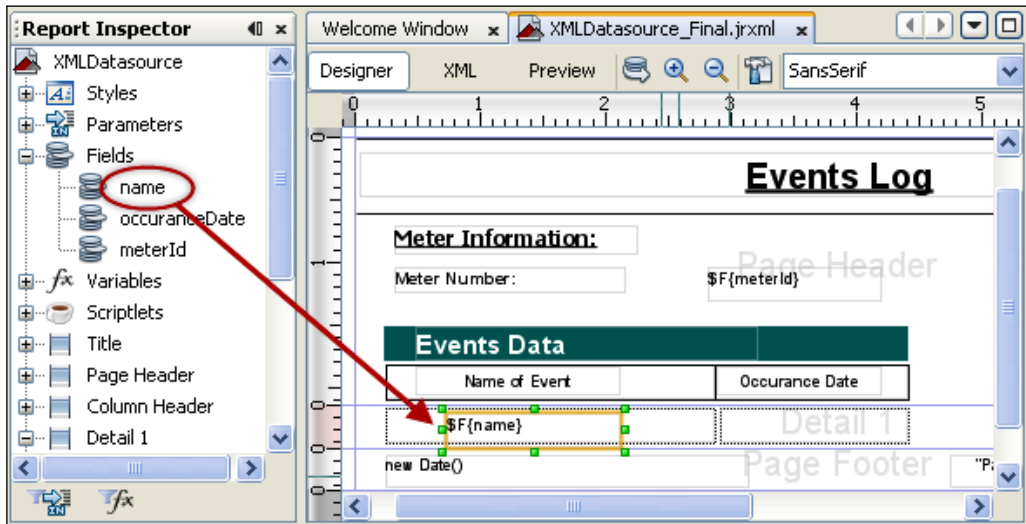
12. Double-click on the **Fields** node in the **Report Inspector** window on the left of your report. The **Fields** node will expand to show the fields' **name**, **occurrenceDate**, and **meterId** added in steps 8 to 11, as shown in the following screenshot:



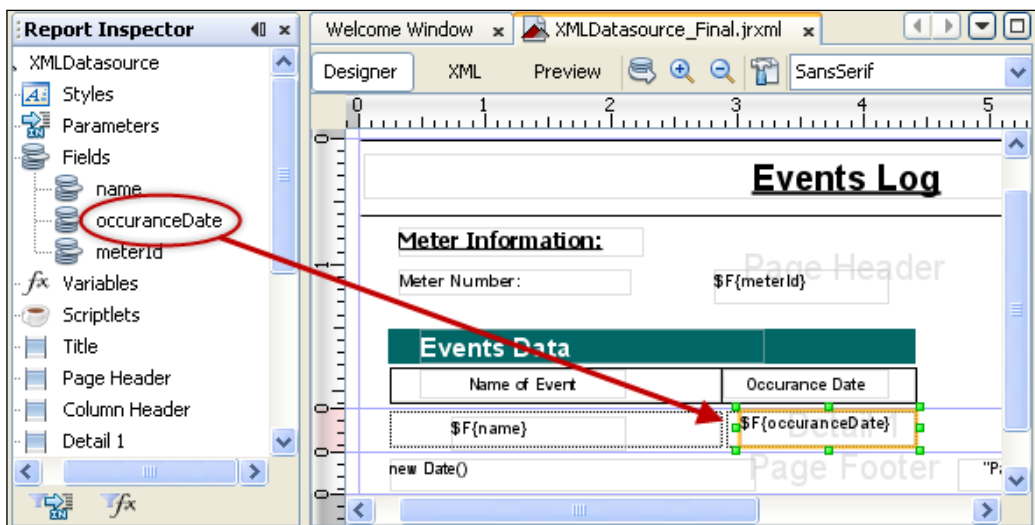
13. Drag-and-drop the **meterId** field from the **Fields** node into the **Page Header** section of your report. A dialog window will appear, as shown in the following screenshot; don't change anything just click the **Ok** button to dismiss it. A **Text Field** component with the expression `$F{meterId}` will appear in the **Page Header** section of your report.



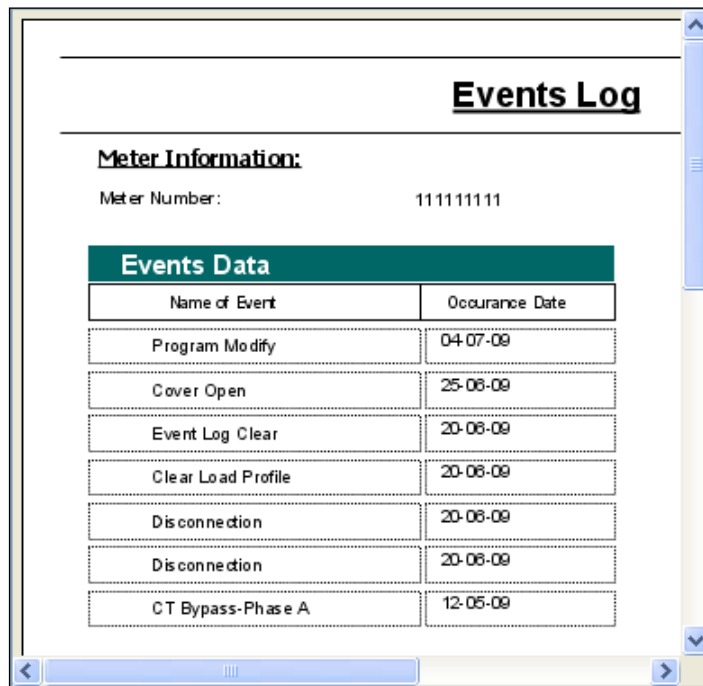
14. Drag-and-drop **name** field from the **Fields** node into the **Detail 1** section of your report under the label named **Name of Event**, as shown in the following screenshot:



15. Similarly, drag-and-drop **occurrenceDate** field from the **Fields** node into the **Detail 1** section of your report under the label named **Occurance Date**, as shown in the following screenshot:



16. Switch to the **Preview** tab and you will see your report showing the events log fetched from the XML datasource using the **XPath** expression you authored in step 6.



Events Log

Meter Information:

Meter Number: 111111111

Events Data	
Name of Event	Occurance Date
Program Modify	04-07-09
Cover Open	25-06-09
Event Log Clear	20-06-09
Clear Load Profile	20-06-09
Disconnection	20-06-09
Disconnection	20-06-09
CT Bypass-Phase A	12-05-09

How it works...

In this recipe you learned how to create a report when your report data source is XML.

In steps 2 to 11 you established a connection to the XML data source, and in step 12 iReport's **Report Inspector** window showed the XML data as local fields (**name**, **occurrenceDate**, and **meterId**). Then, in steps 13 to 15, you dragged-and-dropped the local fields to design your report.

Recall step 5 where the **Report query** dialog changed its view. iReport changed this dialog according to the data source used. All changes related to a data source are confined to this dialog only. The rest of the report-designing steps are mostly the dragging-and-dropping of local fields and do not depend on the type of data source.

This feature can be referred to as loose coupling between a data source and report design. It is very useful. It makes your report design robust and independent of the type of data source used. This means you will not have to redesign the report if your data source changes.

iReoprt is very flexible towards XML data files. It allows you to use XPath to process and filter well-formed XML files. You can use any well-formed XML file in your report and JasperReports will process it to generate local fields, which you can drag-and-drop to design your report.

Using multiple relational databases to generate a report

This recipe teaches you how to create a report with data coming from two different database sources. The main report has six columns, five of which come from one database. The sixth column comes from a subreport, which fetches its data from the second database.

Getting ready

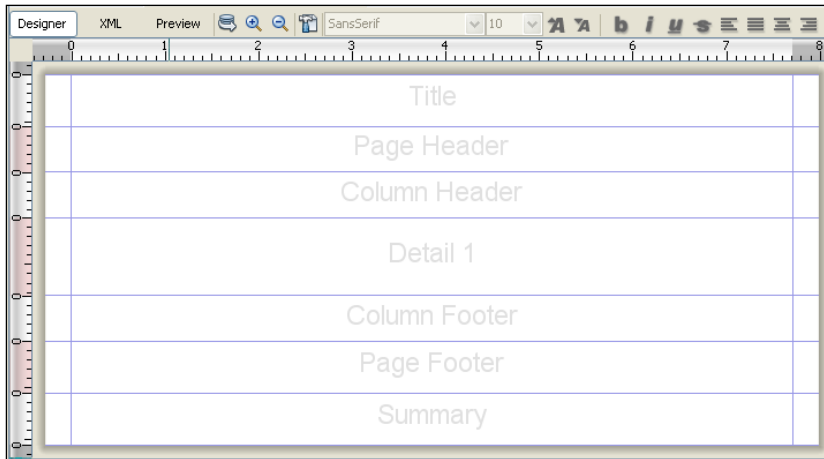
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes two files named `copySampleDataIntoPGS.txt` and `copySamplePaymentStatusDataIntoPGS.txt`. The `copySampleDataIntoPGS.txt` file will help you to create a database named `jasperdb5` and create a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data for this recipe. Similarly, the `copySamplePaymentStatusDataIntoPGS.txt` file will help you to create a database named `jasperdb5a` and create a table named `PaymentDetails` with two columns (`InvoiceID` and `PaymentStatus`) and copy sample data.

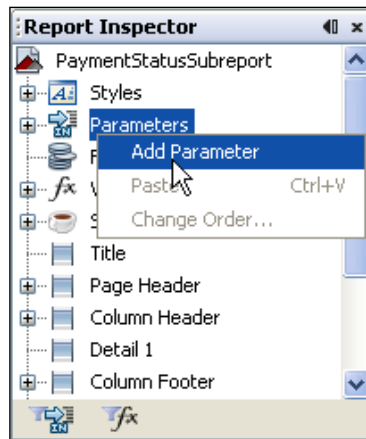
You will be using two JRXML files `MultiDBReport.jrxml` and `PaymentStatusSubreport.jrxml` in this recipe. You will find these files in the `Task4` folder of the source code download for this chapter. The `MultiDBReport.jrxml` file is the master report, which uses the other file as a subreport. The master report has to refer to its subreport using a complete path (you cannot use relative paths). This means you have to copy the two JRXML files to the `c:\JasperReportsCookBookSamples\` folder on your PC. I have hardcoded this complete path in the master report (`MultiDBReport.jrxml`).

How to do it...

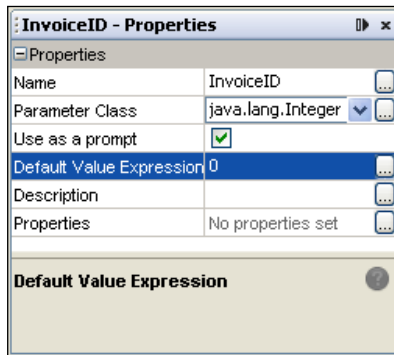
1. Open the `PaymentStatusSubreport.jrxml` file from the `c:\JasperReportsCookBookSamples\` folder. The **Designer** tab of iReport shows an empty report, as shown in the following screenshot:



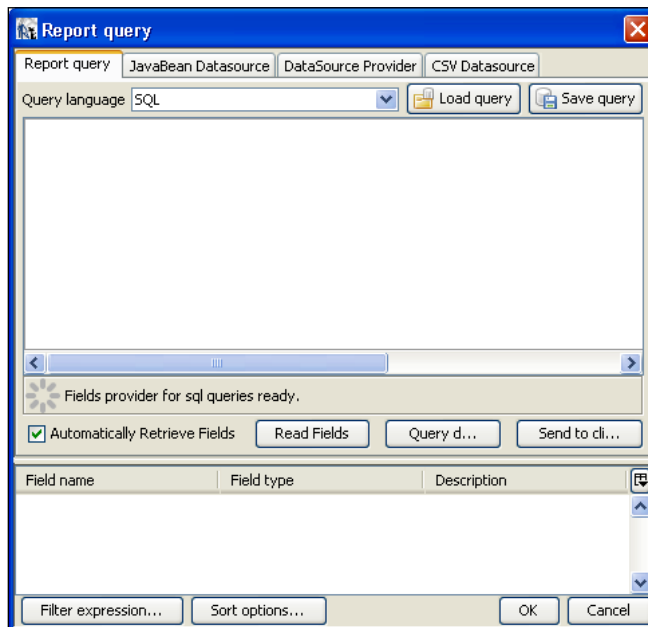
2. Right-click on the **Parameters** node in the **Report Inspector** window on the left of the **Designer** tab, as shown next. Choose the **Add Parameter** option from the pop-up menu.



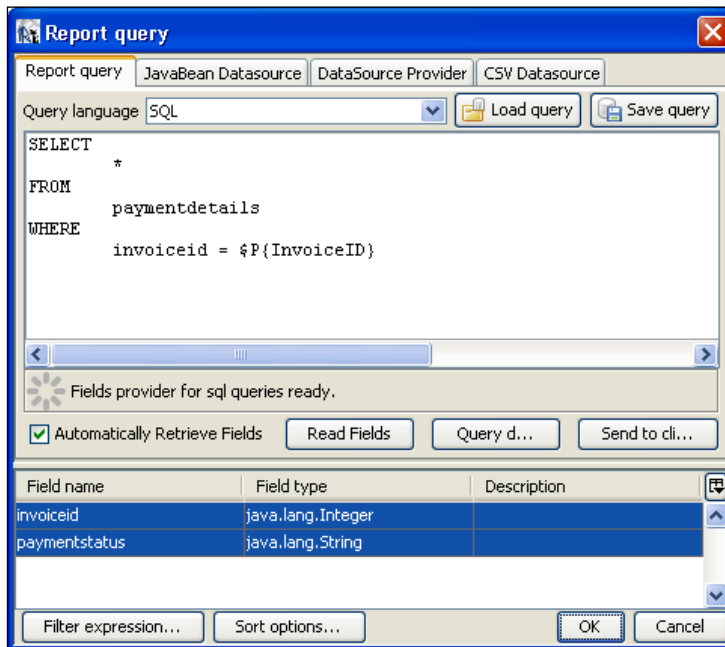
3. The **Parameters** node will expand to show the newly added parameter named **parameter1** at the end of the parameters list. Click on **parameter1**, its properties will appear in the **Properties** window below the palette of components on the right of your iReport main window.
4. Click on the **Name** property of the parameter and type **InvoiceID** as its value. The name of the **parameter1** parameter will change to **InvoiceID**.
5. Click on the **Parameter Class** property and select **java.lang.Integer** as its value.
6. Click on the **Default Value Expression** property and enter **0** as its value, as shown in the following screenshot. Leave the rest of the parameter properties at their default values.



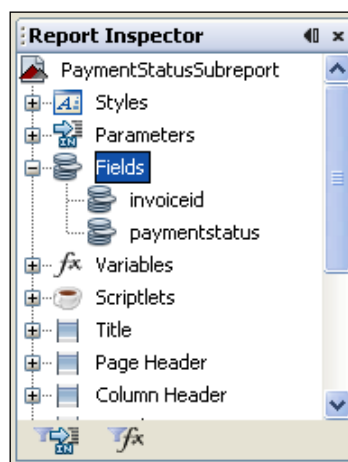
7. Click the **Report query** button on the right of the **Preview** tab; a **Report query** dialog will appear, as shown in the following screenshot:



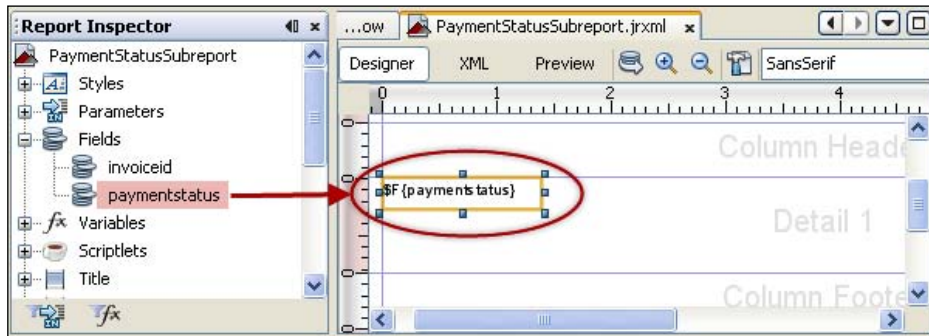
8. Type `SELECT * FROM paymentdetails WHERE invoiceid = $P{InvoiceID}` in the **Query editor**. The fields of the `paymentdetails` table will be shown in the lower-half of the **Report query** dialog. Click the **OK** button, as shown in the following screenshot:



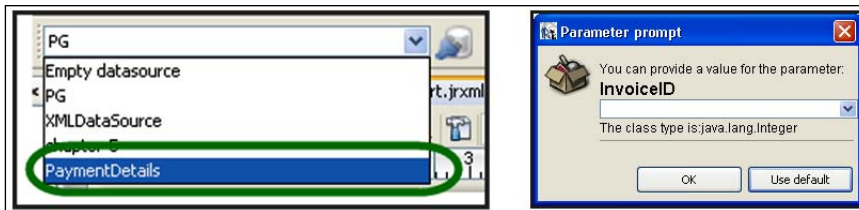
9. Double-click the **Fields** node in the **Report Inspector** window. You will see that it contains **invoiceid** and **paymentstatus** fields, as shown in the following screenshot:



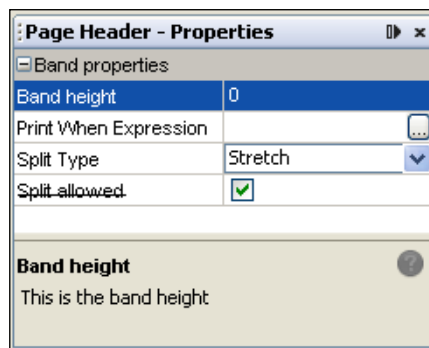
10. Drag-and-drop the **paymentstatus** field from the **Fields** node into the top-left corner of the **Detail 1** section, as shown in the following screenshot:



11. Select **PaymentDetails** in the **datasources** drop-down list, as shown in the left image given below. Then switch to the **Preview** tab; a **Parameter prompt** dialog will appear, which will ask you for the invoice ID, as shown in the right image given below. Enter 1001 as the value of the **InvoiceID** parameter. You will see a report containing a single record showing the payment status of the invoice having the ID 1001.



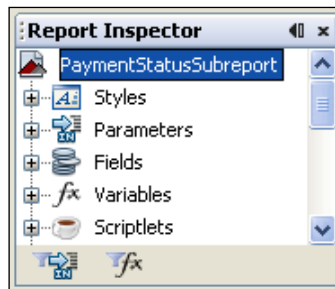
12. Switch back to the **Designer** tab. Click anywhere in the **Page Header** section; its properties will appear in the **Properties** window below the palette. Select the **Band height** property and set **0** as its value, as shown in the following screenshot:



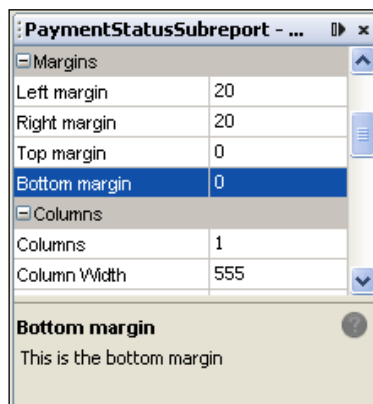
- 13 Similarly, repeat step 12 for the **Title**, **Column Header**, **Column Footer**, **Page Footer**, and **Summary** sections and set **0** as the value of the **Band height** property. This will set all sections to zero height except the **Detail 1** section.
14. Double-click on the blue line at the bottom of the **Detail 1** section. The height of the **Detail 1** section will become equal to the height of the text field dropped into the **Detail 1** section in step 10. The report in the **Designer** tab will look as shown in the following screenshot:



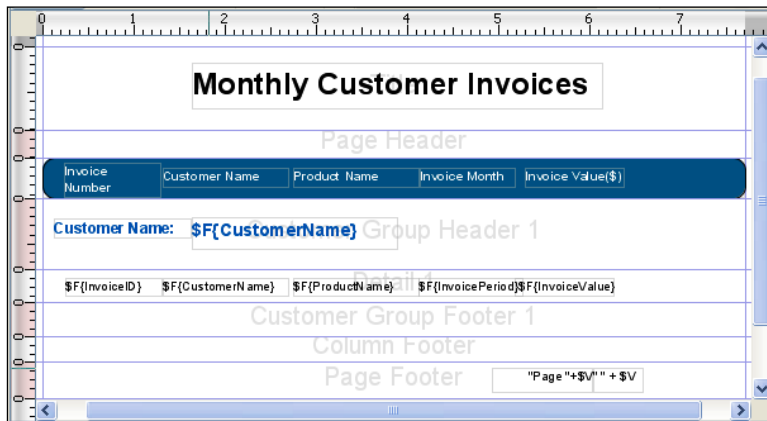
15. Click on the name of your report in the **Report Inspector** window on the left of the **Designer** tab, as shown in the following screenshot:



16. The report properties will appear in the **Properties** window below the palette. Set **0** as the value of the **Top margin** and **Bottom margin** properties in the **Margins** section of the **Properties** window, as shown in the following screenshot:



17. Now open another report named `MultiDBReport.jrxml` from the `C:\JasperReportsCookBookSamples` folder. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header1**, and **Detail 1** sections, as shown in the following screenshot:



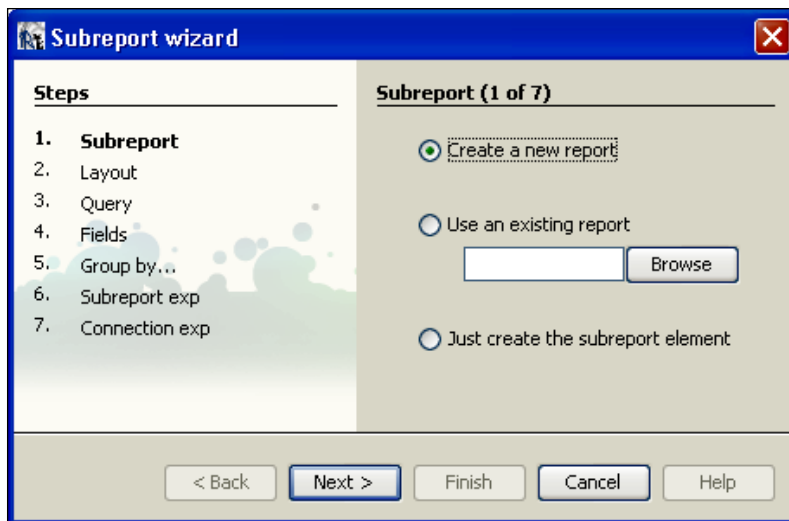
18. Select **PG** in the **datasources** drop-down list. Then click the **Preview** button and you will see a report containing invoices grouped by customer names, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0

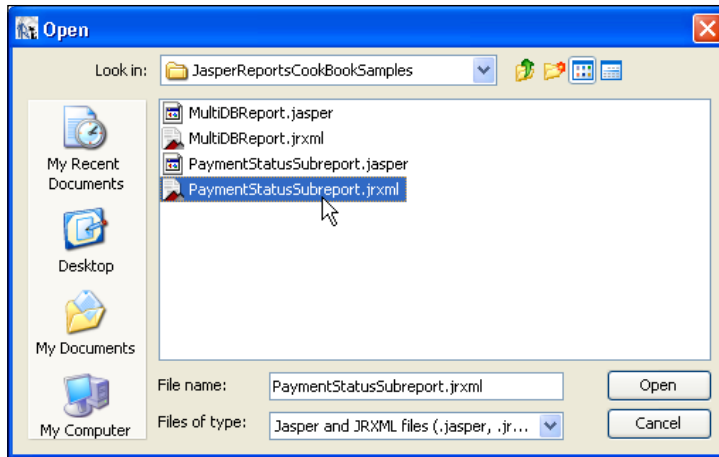
19. Copy-paste the **Invoice Value** column label and place the new copy of the **Invoice Value** label to the right of the existing label, as shown in the following screenshot:



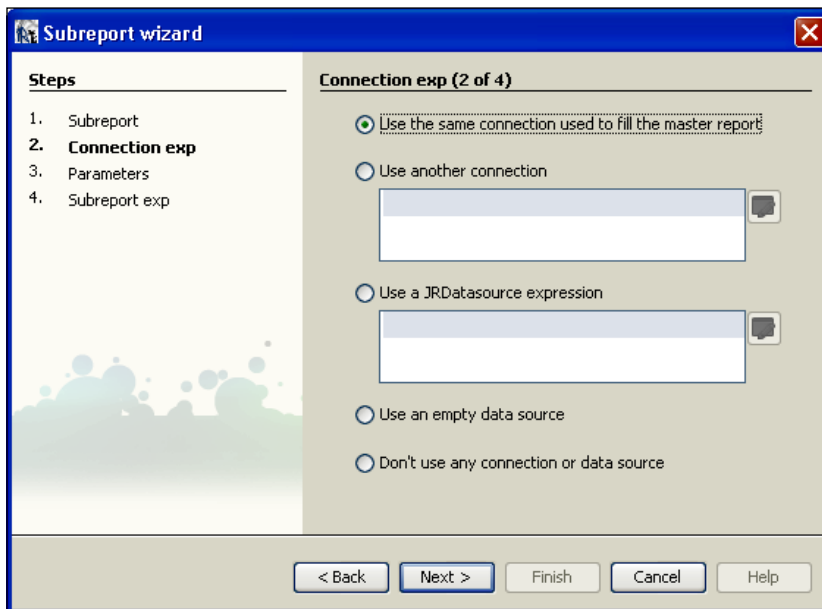
20. Double-click on the new copy of the **Invoice Value** label and type **Payment Status** as its value.
21. Drag-and-drop a **Subreport** component from the palette into the **Detail 1** section, just below the **PaymentStatus** label. A **Subreport wizard** dialog will appear, as shown in the following screenshot:



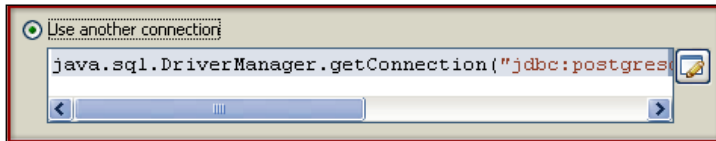
22. Choose the **Use an existing report** option and click the **Browse** button to browse to the `PaymentStatusSubreport.jrxml` file located in the `c:\JasperReportsCookBookSamples\` folder and click the **Open** button, as shown next. The browser dialog will disappear. Click the **Next** button in the **Subreport wizard** dialog.



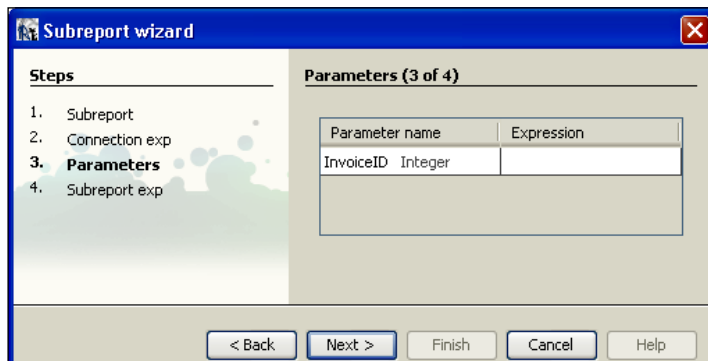
23. A **Connection exp (2 of 4)** dialog will appear, as shown in the following screenshot. Notice that now the left side of the **Subreport wizard** shows four steps (**Subreport**, **Connection exp**, **Parameters**, **Subreport exp**). You are at step 2 (**Connection exp**) which is shown in bold.



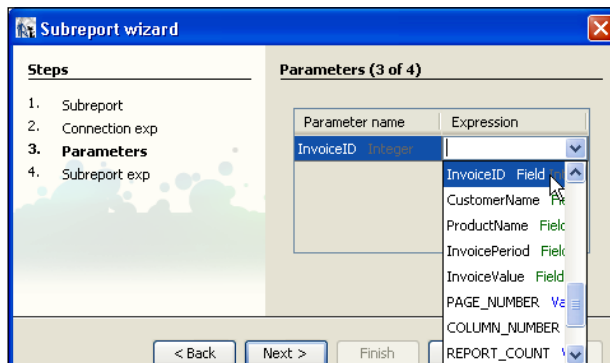
24. Choose the **Use another connection** option and type the `java.sql.DriverManager.getConnection("jdbc:postgresql://localhost:5432/jasperdb5a", "postgres", "postgres")` expression in the area below the **Use another connection** option, as shown next. Click the **Next** button at the bottom of the window.



25. A **Parameters (3 of 4)** dialog will appear. This will show the `InvoiceID` parameter of the `PaymentStatusSubreport.jrxml` subreport and allows you to enter an expression to map it to elements (that is **Fields** or **Variables**) of the main `MultiDBReport.jrxml` report.

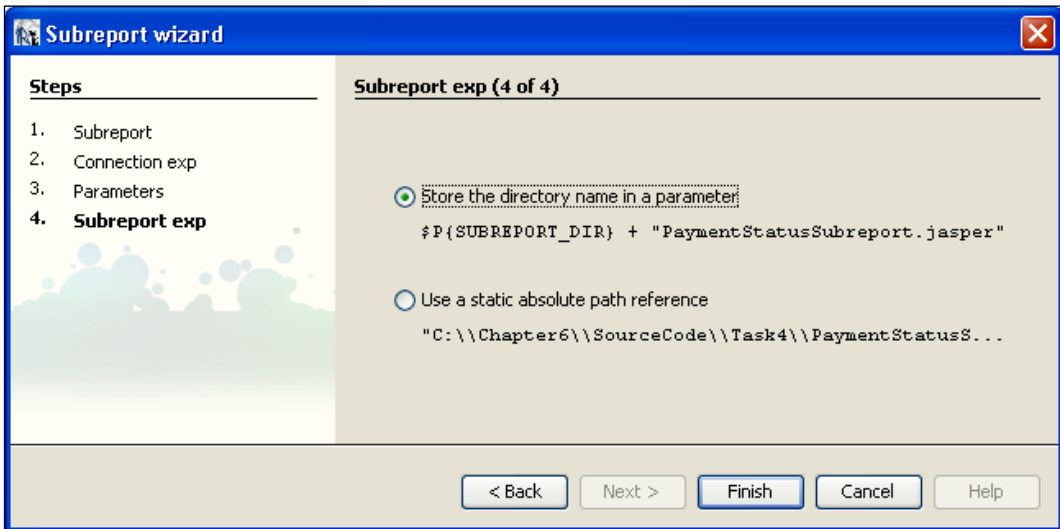


26. Click on the **Expression** field beside the `InvoiceID` parameter. A drop-down list will open. This will show all elements of the `MultiDBReport.jrxml` report. Select the `InvoiceID` field from the list, as shown in the following screenshot:

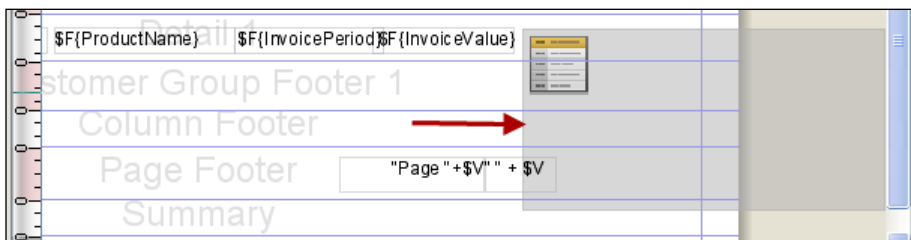


Click the **Next** button.

27. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected and click the **Finish** button.

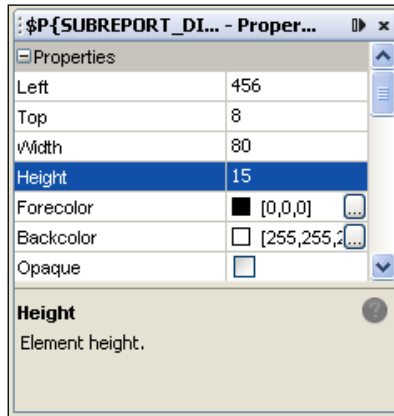


28. The **Subreport** element will be placed in the **Detail 1** section, as shown in the following screenshot:

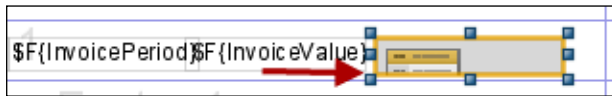


29. Click on the **Subreport** element; its properties will appear in the **Properties** window below the palette. Find the **Width** property and set **80** as its value.

30. Click on the **Height** property and set 15 as its value, as shown in the following screenshot:



31. The size of the **Subreport** element will become equal to the size of the other fields placed in the **Detail 1** section, as shown in the following screenshot:



32. Click the **Preview** button and you will see a report containing invoices with their status coming from the payment status subreport, as shown in the following screenshot:

Monthly Customer Invoices					
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)	Payment Status
Customer Name: ABC Publishing					
1012	ABC Publishing	Offset Paper	Jun09	6875.0	Paid
1010	ABC Publishing	Offset Paper	Feb09	4552.2	Paid
1011	ABC Publishing	Packing Material	Jun09	8745.5	UnPaid
1013	ABC Publishing	Packing Material	Aug09	9852.0	Over-due
1020	ABC Publishing	Printing Ink	Mar09	3450.0	Paid
1016	ABC Publishing	Printing Ink	Jun09	2440.0	Paid
Customer Name: Alice					
1038	Alice	JasperReport	Mar09	49.5	Paid
1007	Alice	JasperReport	Apr09	49.5	Paid
1014	Alice	WordPress Cookbook	Jan09	45.0	Paid
Customer Name: Bob					
1006	Bob	JasperReports	Mar09	50.0	UnPaid
1037	Bob	JasperReports	Mar09	50.0	Paid
1018	Bob	Printing Ink	Sep09	150.0	UnPaid
1015	Bob	WordPress Cookbook	May09	45.0	Paid
Page 1 of 4					

How it works...

In this recipe, you have done four things:

1. First you made a subreport with a parameter named **InvoiceID** in steps 1 to 16. The subreport is a very useful feature that allows you to design a portion of a report as a separate, independent report and then insert the portion into the main report.
2. Then you inserted the subreport into a main report in step 21.
3. You configured a new database connection for the subreport in step 24. Note from the screenshot of step 24 that you used the `java.sql.DriverManager` class to configure this database connection for the subreport.

When I configured the database connection in step 24, the name of my database server was `postgresql`, the network address of the machine hosting the server was `localhost`, the server was listening at port `5432`, the name of the database was `jasperdb5a`, and the username and password were both `postgres`. Putting all this together, I got the complete expression of step 24 as `"java.sql.DriverManager.getConnection ("jdbc:postgresql://localhost:5432/jasperdb5a", "postgres", "postgres")"`. This way you can configure any database (for example MySQL) for your subreport.

4. You mapped the **InvoiceID** parameter of the subreport to the **InvoiceID** field of the main report in steps 25 and 26.

The result of these four steps is that the main report fetches its five columns from its own database connection. Then it asks the subreport to provide data for the sixth column (that is, **Payment Status**). The subreport fetches data for the sixth column from another database connection and returns the data to the main report, which eventually displays the data coming from the subreport in the sixth column.

This way you can design reports using multiple databases.

Creating a report from model beans of Java applications

JasperReports allows you to generate reports directly from data contained in Java objects such as JavaBeans and **Plain Old Java Objects (POJOs)**.

This recipe shows you how to connect your iReport installation to your JavaBeans or POJOs, and then access the data contained in the beans to generate your report.

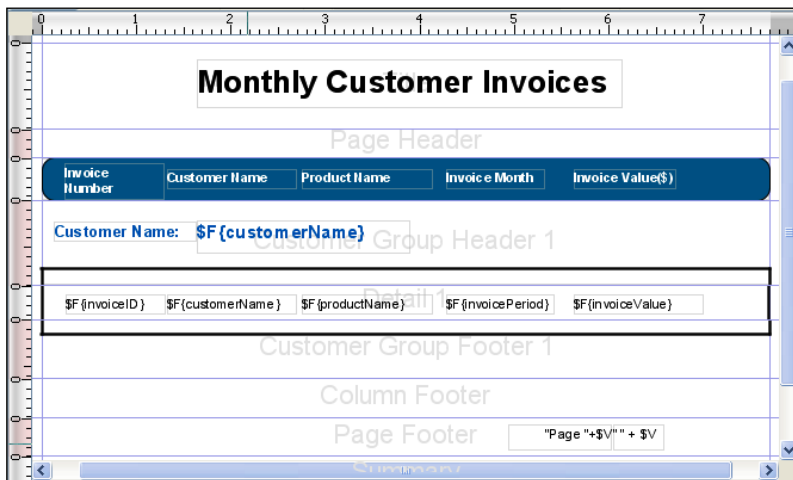
Getting ready

You need a Java JAR file that contains class files for the JavaBeans required for this recipe. A `custInvoices.jar` file is contained in the source code for this chapter. Unzip the source code file for this chapter and copy the `Task5` folder from the unzipped source code to a location of your choice.

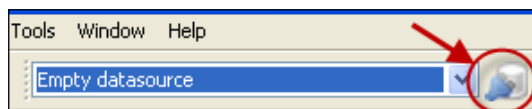
How to do it...

Let's start using Java objects as data storage units.

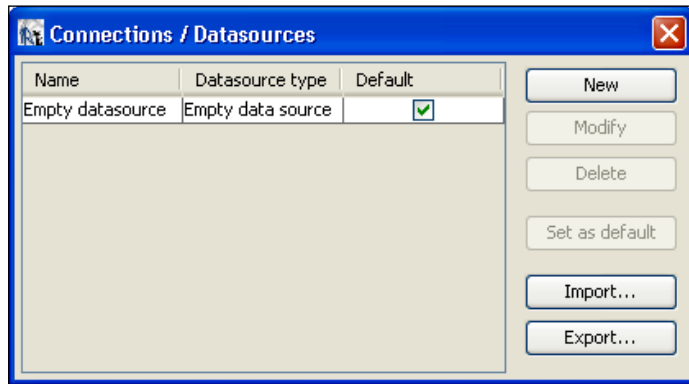
1. Open the `ModelBeansReport.jrxml` file from the `Task5` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header1** and **Detail 1** sections, as shown in the following screenshot:



2. If you have not made any database connection so far in your iReport installation, you will see an **Empty datasource** shown selected in a drop-down list just below the main menu. Click on the **Report Datasources** icon, shown encircled to the right of the drop-down list in the following screenshot:



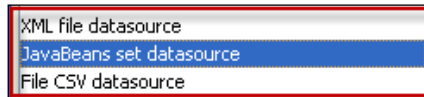
3. A new window named **Connections / Datasources** will open, as shown next. This window lists an **Empty data source** as well as the datasources you have made so far.



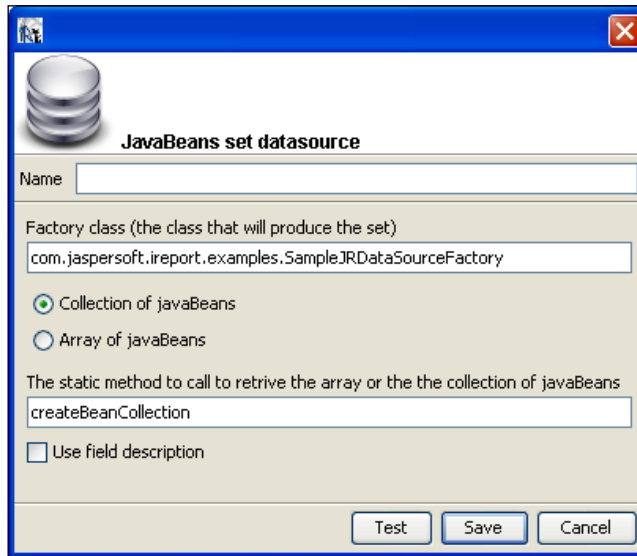
4. Click the **New** button at the top-right of the **Connections / Datasources** window. This will open a new **Datasource** selection window, as shown in the following screenshot:



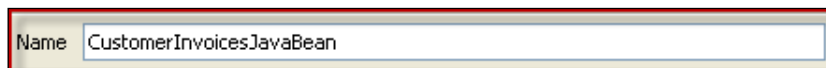
5. Select **JavaBeans set datasource** from the datasource types, as shown next. Click the **Next** button.



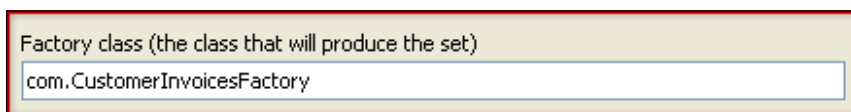
6. A new window named **JavaBeans set datasource** will open, as shown in the following screenshot:



7. Enter `CustomerInvoicesJavaBeans` as the name of your new connection in the text box beside the **Name** field, as shown in the following screenshot:

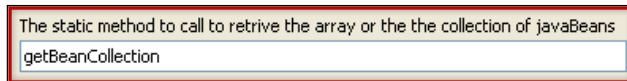


8. Enter `com.CustomerInvoicesFactory` as the name of the factory class in the text box beside the **Factory class** field, as shown in the following screenshot:

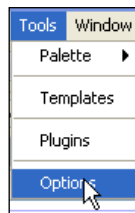


This `com.CustomerInvoicesFactory` class provides iReport with access to JavaBeans that contain your data.

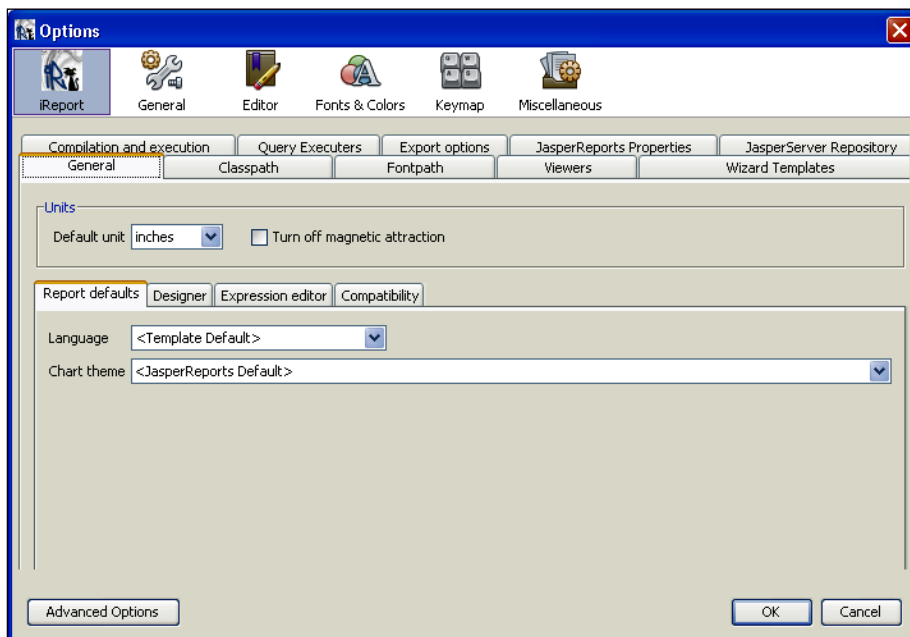
9. Enter `getBeanCollection` as the name of the static method in the text box beside **The static method...** field, as shown in the following screenshot:



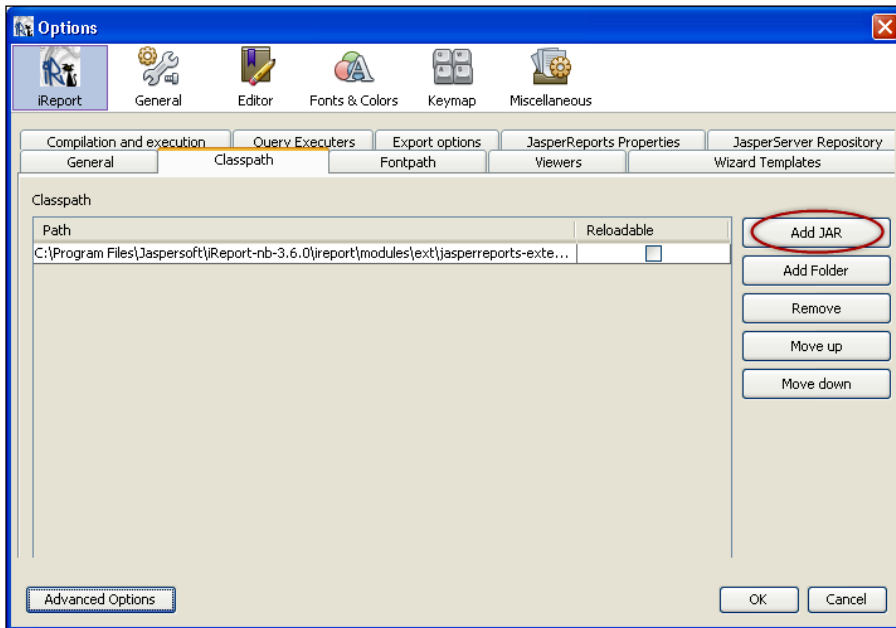
10. Leave the rest of the fields at their default values. Click the **Test** button to test your new connection to the JavaBeans datasource. You will see an **Exception** message dialog.
11. This exception message occurs because iReport can't find your factory class. Dismiss the message box by clicking **OK**.
12. Click the **Save** button at the bottom of the **JavaBeans set datasource** window and close the **Connections / Datasources** window as well.
13. Click **Tools** from the main iReport menu and select **Options** from the drop-down list, as shown in the following screenshot:



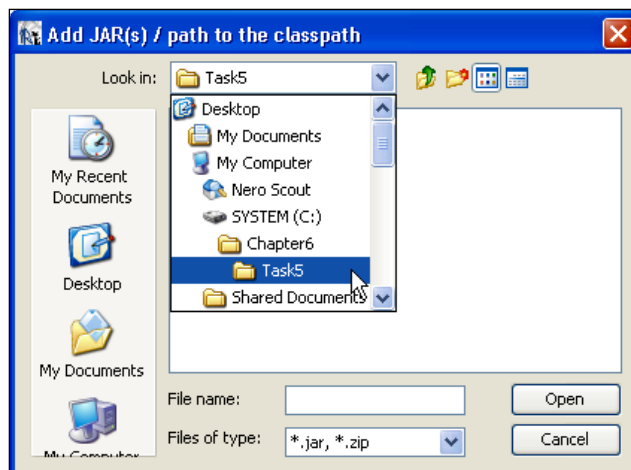
14. A window named **Options** will open, as shown in the following screenshot:



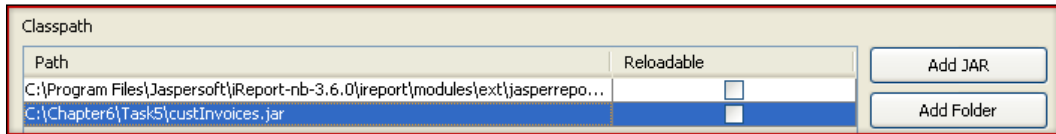
15. Click on the **Classpath** tab. You will see a default classpath in a path table, as shown next. Click the **Add JAR** button on the top-right part of the **Classpath** tab.



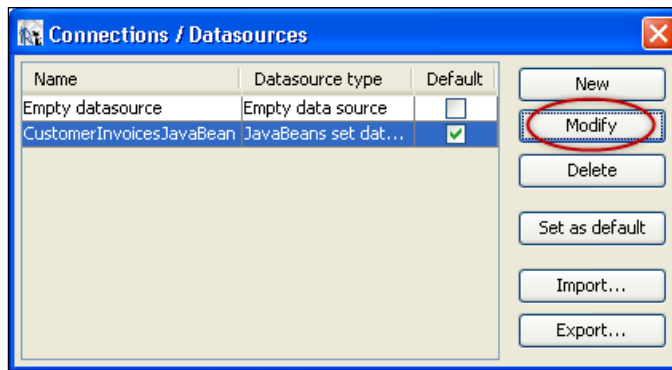
16. A file browser will appear. Browse to the Task5 folder that you copied in the *Getting ready* section. Select the `custInvoices.jar` file and click the **Open** button, as shown in the following screenshot:



17. The path to the JAR file containing JavaBeans will be added to the path table, as shown next. Click the **OK** button at the bottom of the **Options** window.

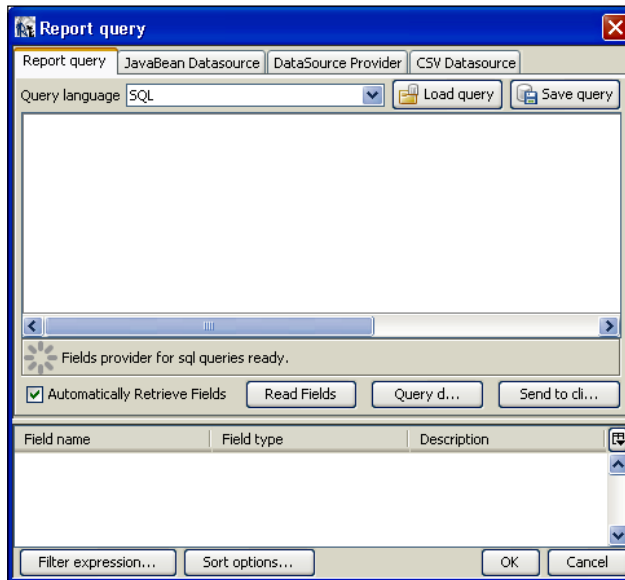


18. Click on the **Report Datasources** icon to the right of the datasources drop-down list just below the main menu, as shown encircled in the following screenshot. A new window named **Connections / Datasources** will open, as shown next. It will show a list of available connections. Select the **CustomerInvoicesJavaBean** connection and click the **Modify** button.

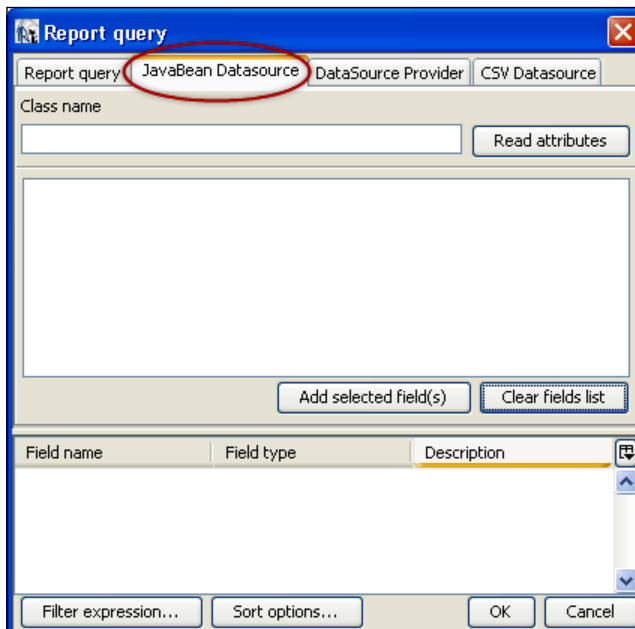


19. The **JavaBeans set datasource** window of step 6 will open. Click on the **Test** button at the bottom of the window. You will see a **Connection test successful!** message dialog. Click **OK** to dismiss it.
20. Click the **Save** button at the bottom of the **JavaBeans set datasource** window to save the connection settings.
21. You will see a **Connections/Datasources** window with the **CustomerInvoicesJavaBean** connection set as default. Click the close icon on the top-right corner of the window to close it.

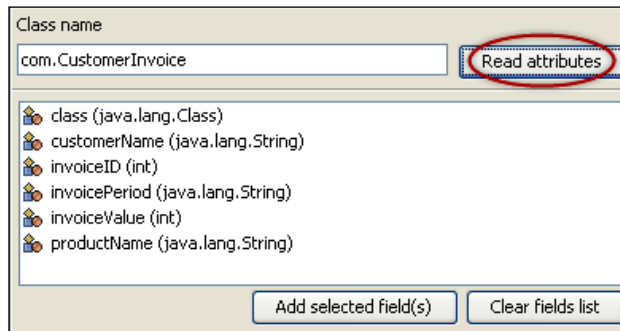
22. Click the **Report query** button on the right of the **Preview** tab. A **Report query** dialog will appear, as shown in the following screenshot:



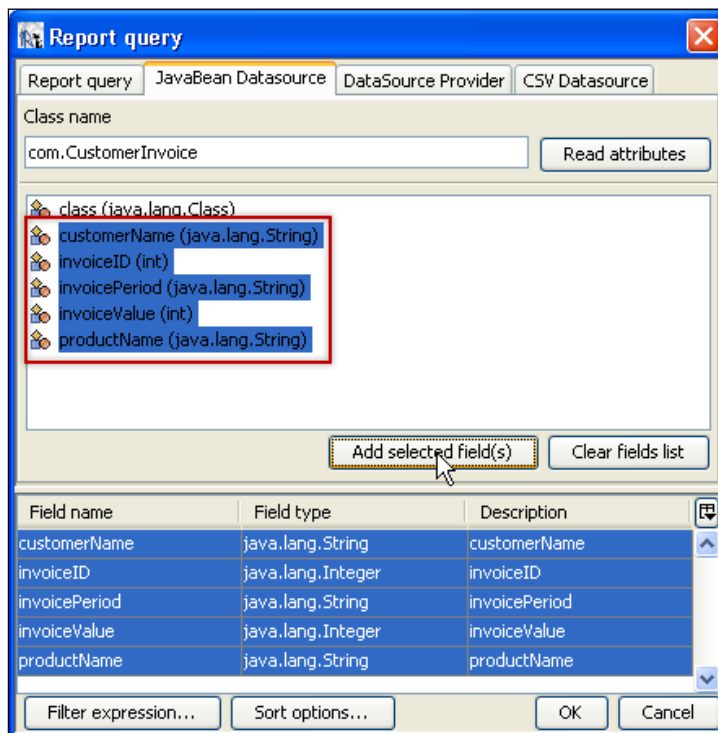
23. Click on the **JavaBeans Datasource** tab. The **JavaBeans Datasource** tab will appear, as shown in the following screenshot:



24. Enter `com.CustomerInvoice` as name of the class in the text box beside the **Class name** field. Then click the **Read Attributes** button. It will read the fields of the `com.CustomerInvoice` class and list them in the text area below the **Class name** field, as shown in the following screenshot:



25. Select all the fields of the `com.CustomerInvoice` class except the **class** field, as shown in the in the following screenshot. Click the **Add selected field(s)** button at the bottom of the text area containing the fields list. This will show selected fields in the lower-half of the **Report query** dialog. Click the **OK** button.



26. Switch to the **Preview** tab and you will see all the invoices listed in the report grouped by customer names.

How it works...

In this recipe you have learned how to create a connection to JavaBeans and then use the data stored in the beans to generate your report. For this purpose, you did four things:

1. You configured the name of a factory class in step 8 of the recipe.
2. The factory class should have a static method to instantiate or access your JavaBeans. You configured the name of this static method in step 9 of the recipe.
3. You provided a classpath to a JAR file, which contains the factory class as well as a class for the JavaBeans in step 16.
4. Then you configured the properties of your JavaBeans as fields in iReport in steps 23 to 25. This enables iReport to fetch data from your JavaBeans.

JasperReports internally calls the `getBeanCollection()` static method of the factory class. This method should contain the Java code that returns the actual JavaBeans from which you want to generate the report. For the sake of simplicity, I have returned a hardcoded object as shown in the following piece of code:

```
public static Vector getBeanCollection() {
    CustomerInvoice invoice001 = new CustomerInvoice (
        1001,"Packt Publishing","Packing Material","Mar09",5020);
    CustomerInvoice invoice002 = new CustomerInvoice (
        1002,"Packt Publishing","Offset Paper","Apr09",3000);
    //Other instance creation calls to the CustomerInvoice class
    Vector invoices = new Vector();

    invoices.add(invoice001);
    invoices.add(invoice002);
    //more calls to invoices.add() method

    return invoices;
}
```


5

Multi-page Reports

In this chapter, you will learn:

- ▶ Building a cover page for your multi-page report
- ▶ Creating a simple one-page TOC for your report
- ▶ Applying style to your simple TOC
- ▶ Resetting page numbering with the start of a particular record
- ▶ Implementing complex, multi-dimensional page numbering
- ▶ Showing multiple types of data in the same report
- ▶ Managing pagination of multiple types of data in a report

Introduction

This chapter covers multi-page reports. Real-world multi-page reports have certain common features, such as a cover page, a **Table of contents (TOC)**, and page numbering. The recipes of this chapter teach you about all these features.

Another important point: Multi-page reports are generally of two types. The first type is simple, in which the same type of data is repeated on several pages to form a multi-page report. The second type is complex, in which you have to mix different types of data, coming from different data sources, into a single report. I cover both types of multi-page reports in this chapter.

Building a cover page for your multi-page report

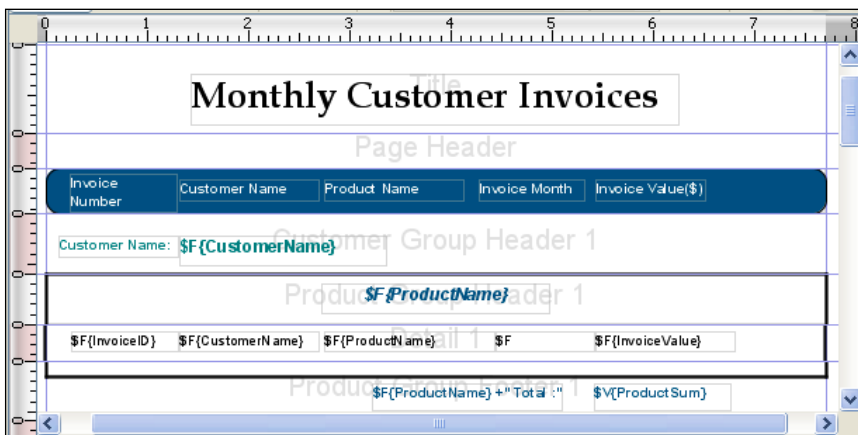
While working with multi-page reports, you will often need to design cover pages for your reports. This recipe teaches you a few simple tricks to use the **Title** section of JasperReports as a cover page.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb6` and copy sample data for this recipe into the database.

How to do it...

1. Open `CoverPageReport.jrxml` file from the `Task1` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in **Title**, **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, and **Product Group Footer 1** sections, as shown in the following screenshot:



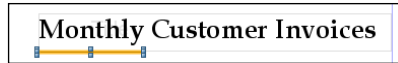
- Switch to the **Preview** tab and you will see a report containing customer invoices data.

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
<i>Offset Paper</i>				
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Offset Paper Total :				11427.0
<i>Packing Material</i>				
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				18597.0
<i>Printing Ink</i>				
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
Printing Ink Total :				5890.0

- Switch back to the **Designer** tab. Click anywhere inside the **Title** section. Its properties will appear in the **Properties** window below the **Palette** of components. Set 600 as the value of the **Band height** property.
- Select the title **Static Text** component and right-click on it. When a pop-up menu appears, select **Align**. Another pop-up will appear. Select **Align To Right Margin**. This will move the title **Static Text** component horizontally to touch the right margin of your report, as shown.



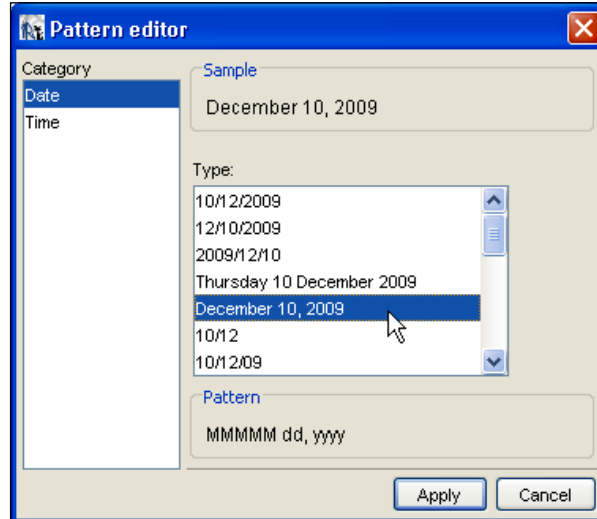
5. Right-click the title **Static Text** component. When a pop-up menu appears; select **Position**. Another pop-up will appear. Select **Center Vertically**.
6. Drag-and-drop a **Line** component from the **Palette** into the **Title** section just below the title **Static Text** component.



7. While the **Line** component is selected, its properties will appear in the **Properties** window below the **Palette**. Select the **Width** property and set 555 as its value.
8. Right-click the **Line** component. When a pop-up menu appears, select **Align**. Another pop-up will appear. Select **Align To Right Margin**. This will move the **Line** component to touch the right margin of your report, as shown.



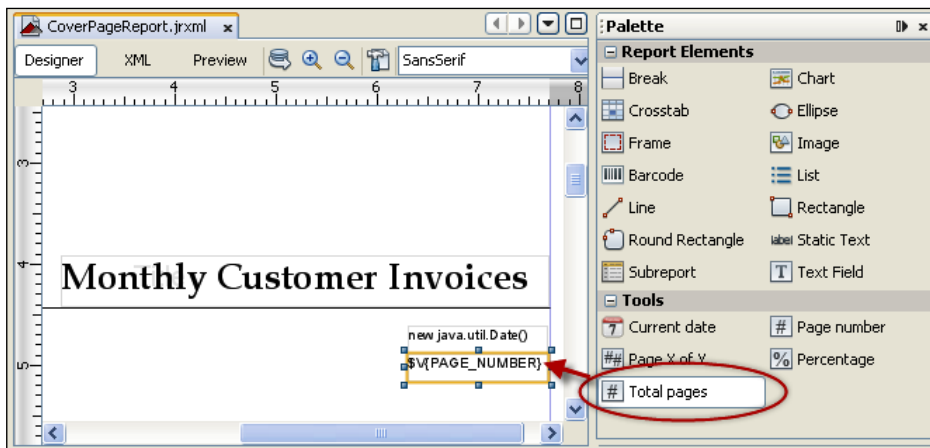
9. Drag-and-drop a **Current Date** component from the **Tools** section of the **Palette** into the **Title** section, just below the **Line** component dropped in step 6. A **Pattern editor** window will open allowing you to choose a date pattern for your date component, as shown in the next screenshot:



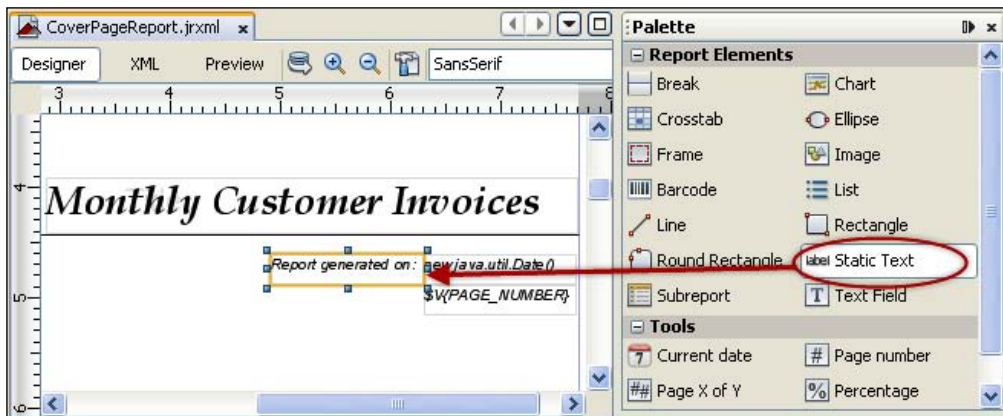
10. Choose a date pattern from the **Type** section of the **Pattern editor** window and press the **Apply** button. A `java.util.Date()` text field will be placed below the **Line** component, as shown in the following screenshot:



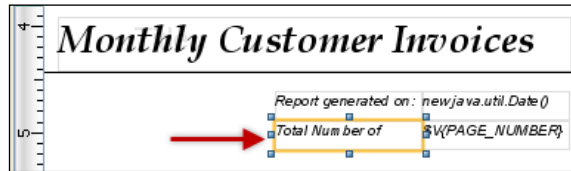
11. Drag-and-drop a **Total pages** component from the **Tools** section of the **Palette** into the **Title** section, just below the `java.util.Date()` text field, as shown.



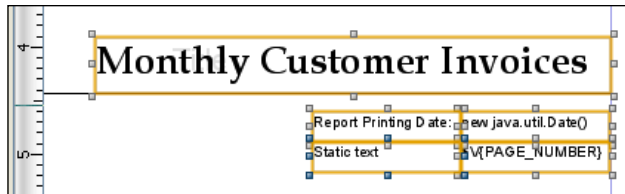
12. Drag-and-drop a **Static Text** component from the **Palette** into the **Title** section, just to the left of the `java.util.Date()` text field, as shown in the next screenshot. Double-click on it and type Report generated on: as its label.



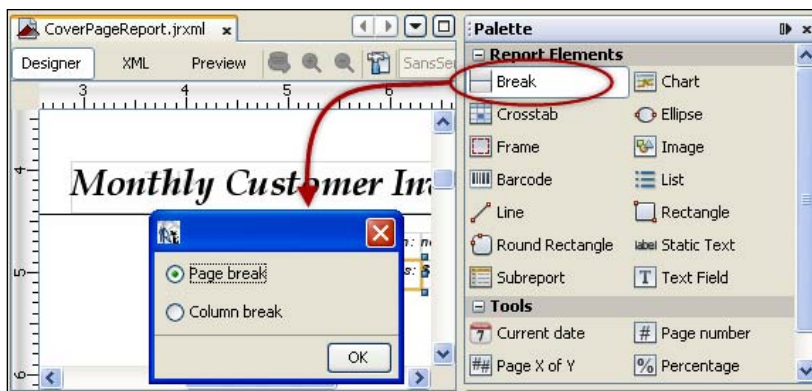
13. Drag-and-drop another **Static Text** component from the **Palette** into the **Title** section just to the left of the `$V{PAGE_NUMBER}` text field. Double-click on it and type **Total Number of Pages :** as its label.



14. Select all the components, except the **Line** component from the **Title** section.



15. The **Multiple Objects-Properties** window will appear below the **Palette**. Look for the **Italic** property under the **Text properties** section and check the box against it. It will give an italic look to all the selected components in the **Title** section.
16. Select the **Total Number of Pages :** **Static Text** component; its properties will appear in the **Properties** window below the **Palette**. Select the **Width** property and set **110** as its value.
17. Select the **Left** property and set **340** as its value.
18. Drag-and-drop a **Break** component from the **Palette** into the **Title** section, just below the **Total Number of Pages :** **Static Text** component. A prompt will appear, which allows you to choose between a **Page break** or **Column break**, as shown in the following screenshot. Press **OK** to accept the default **Page break** option.



19. Switch to the **Preview** tab and you will see that the title, date, and total number of pages appear on a separate cover page. The rest of the report starts from the second page.



How it works...

You have learned two simple tricks to build a cover page in this recipe. The first trick was in step 3 when you set the height of your **Title section** to 600. As the **Title section** appears only at the first page, increasing its height only affects the first page and no other pages.

The second trick was in step 18, when you inserted a line break at the end of the **Title section**. This makes your **Title section** appear on a separate page, giving the feel of a cover page.

You will also notice that the footer of the report in this recipe contains page numbers. JasperReports automatically manages page numbering correctly. When you inserted a page break in step 18, you can see that JasperReports automatically started counting the cover page as page 1, the next page as page 2, and so on.

Creating a simple, one-page TOC for your report

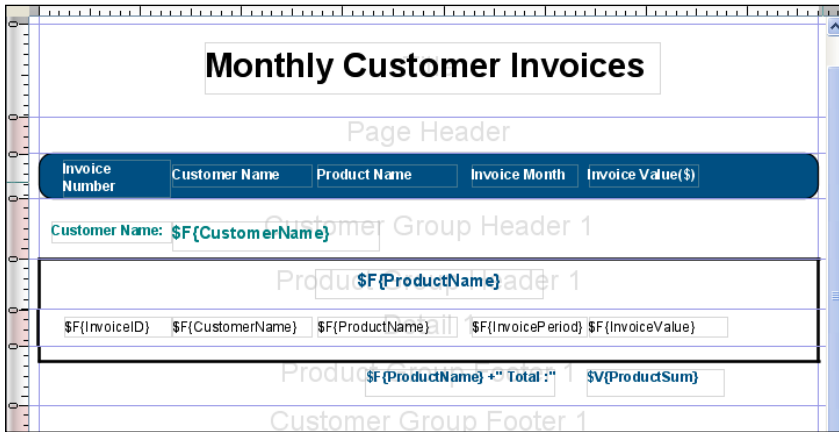
This recipe shows how you can build a simple one-page TOC for your report.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb6` and copy sample data for this recipe into the database.

How to do it...

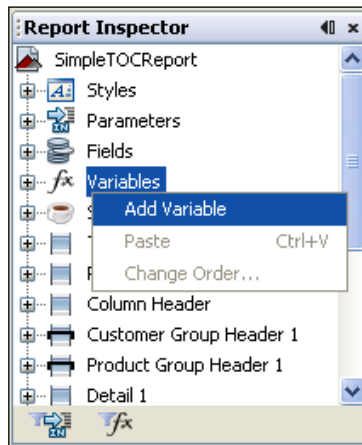
1. Open the `SimpleTOCReport.jrxml` file from the `Task2` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in **Title**, **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, and **Product Group Footer 1** sections, as shown in the following screenshot:



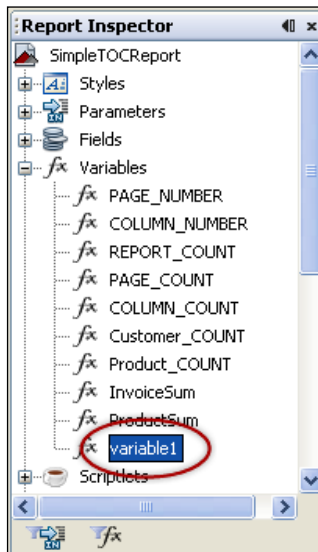
2. Switch to the **Preview** tab and you will see invoices for each customer grouped by product names.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
Offset Paper				
1012	ABC Publishing	Offset Paper	Jun09	68750
1010	ABC Publishing	Offset Paper	Feb09	45522
Offset Paper Total :				11427.0
Packing Material				
1011	ABC Publishing	Packing Material	Jun09	87455

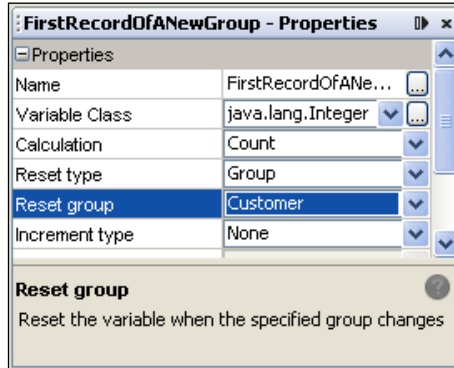
3. Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window on the left side of your report. From the pop-up menu that appears, select the **Add Variable** option.



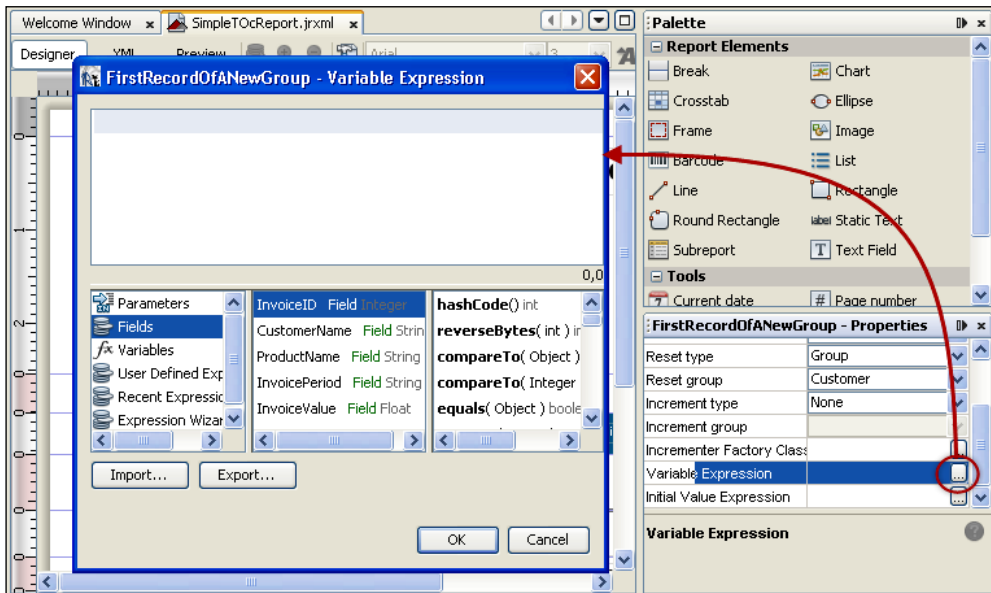
4. A new variable named **variable1** will be added at the end of the variables list.



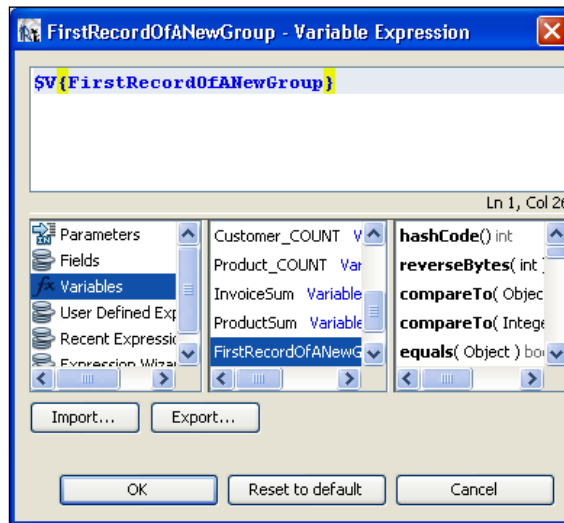
5. While **variable1** is selected, find the **Name** property in the **Properties** window below the **Palette** of components and change its value to `FirstRecordOfANewGroup`. Now the name of the **variable1** variable will change to `FirstRecordOfANewGroup`.
6. Select the **Variable Class** property and change its value to `java.lang.Integer`.
7. Select the **Calculation** property and change its value to `Count`.
8. Select the **Reset type** property and change its value to `Group`.
9. Select the **Reset group** property and change its value to `Customer`.



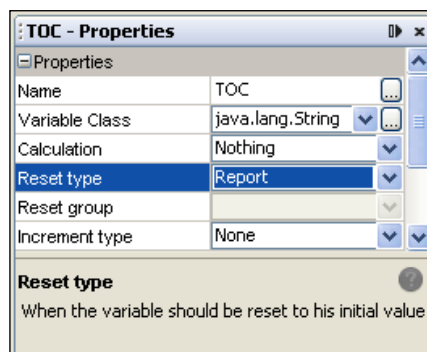
10. Select the **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open, as shown in the next screenshot:



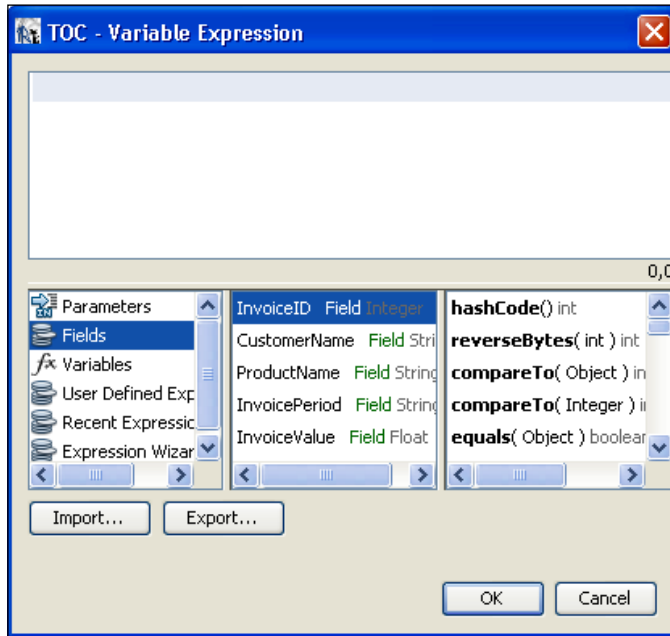
11. Select **Variables** in the first column of the lower-half of the **Variable Expression** window. Then double-click the `FirstRecordOfANewGroup` variable in the second column. A new expression `$V{FirstRecordOfANewGroup}` will appear in the **Variable Expression** window, as shown in the next screenshot. Press the **OK** button.



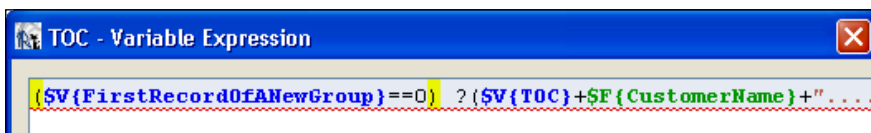
12. Right-click on the **Variables** node in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Variable** option.
13. A new variable named **variable1** will be added at the end of the variables list.
14. While **variable1** is selected, find the **Name** property in the **Properties** window below the **Palette** of components and change its value to `TOC`. Now the name of the **variable1** variable will change to `TOC`.
15. Select the **Variable Class** property and change its value to `java.lang.String`.
16. Select the **Reset type** property and change its value to `Report`, as shown in the following screenshot:



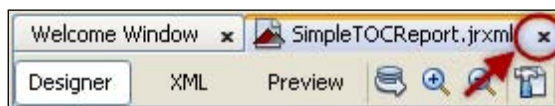
17. Look for the **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open.



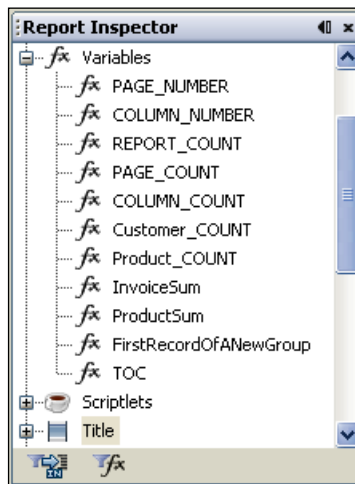
18. Type `($V{FirstRecordOfANewGroup}==0) ? ($V{TOC}+$F{CustomerName} + " " + $V{PAGE_NUMBER}+ "
") : $V{TOC}` in the expression editor window, as shown in the next screenshot. Press the **OK** button.



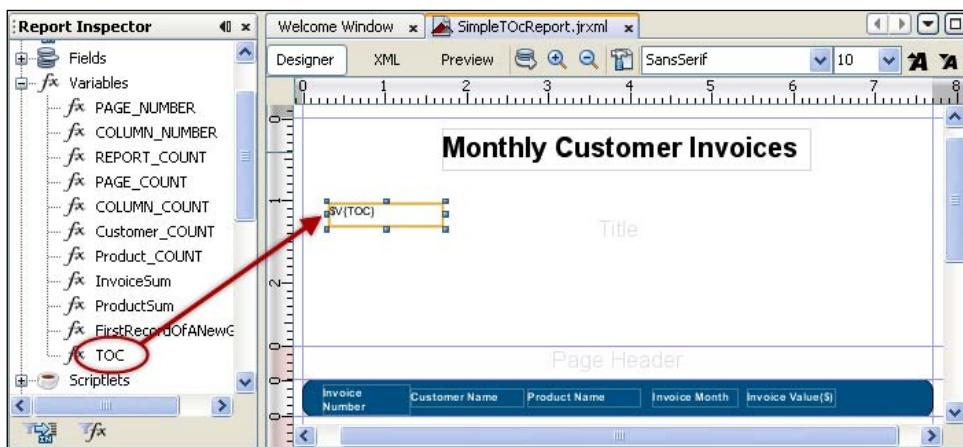
19. Click the **Initial Value Expression** property and type `new java.lang.String()` as its value. Now the properties of the **TOC** variable are all set.
20. Save your report by choosing the **Save** option from the **File** menu. Click the close icon to the right of your report title to close the report, as indicated in the following screenshot.



21. Re-open the saved SimpleTOCReport.jrxml file. You have to close and re-open your .jrxml file every time you play with the **Initial Value Expression** property of a variable. This is due to a bug in iReport version 3.6.0
22. Click the **Title** section of your report. Its properties will appear in the **Properties** window below the **Palette**. Look for the **Band height** property and set its value as 200.
23. Double-click the **Variables** node in the **Report Inspector** window on the left side of your report. The **Variables** node will expand showing all its children (that is, all variables present in the report) as shown in the following screenshot;



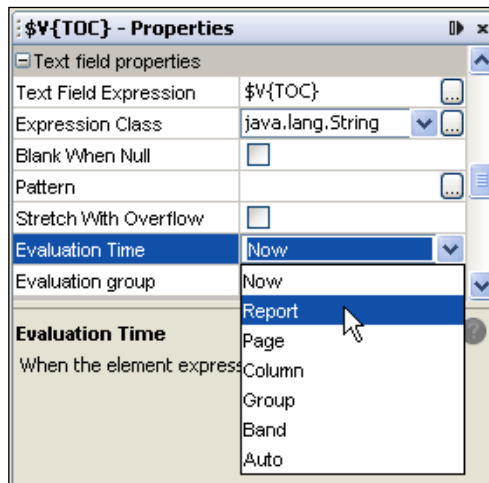
24. Drag-and-drop the variable named **TOC** from the **Variables** node into the **Title** section of your report, as shown in the next screenshot:



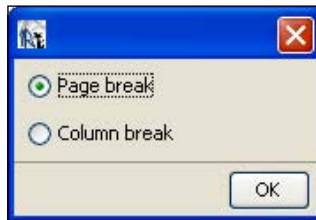
25. Click the `$V{TOC}` text field in the **Title** section. Its properties will appear in the **Properties** window below the **Palette**. Find the **Width** property and set its value as 300. Similarly, set 100 as the value of its **Height** property.
26. Click the **Markup** property under the **Text properties** section in the **Properties** window and select `html` as its value from the drop-down list beside it.
27. Switch to the **Preview** tab. You will see that the **TOC** field does not display any content, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
Offset Paper				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
Offset Paper Total :				11427.0

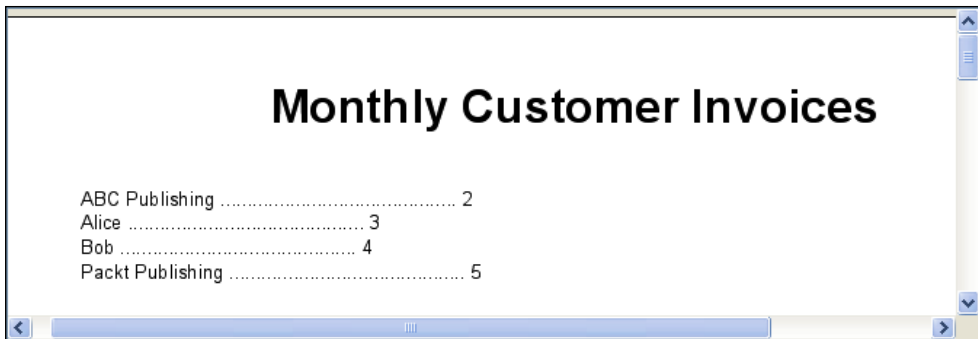
28. Switch back to the **Designer** tab. Click the `$V{TOC}` text field in the **Title** section. Its properties will appear in the **Properties** window below the **Palette**. Find the **Evaluation Time** property and select **Report** as its value from the drop-down list beside it.



29. Drag-and-drop a **Break** component from the **Palette** into the **Title** section below the `$v{TOC}` text field. You will see a prompt to choose between a **Page break** or **Column break**, as shown in the next screenshot. Press **OK** to accept the **Page break** option, selected by default.



30. Switch to the **Preview** tab. You will see that the TOC of your report is displayed below the title and the actual report starts on a new page due to the page break you inserted after the TOC.



How it works...

You have used a two-variable trick to build a one-page TOC for your customer invoices report. The two variables are `FirstRecordOfANewGroup` (which you named in step 5) and `TOC` (which you named in step 14). These two variables together build the TOC, as explained below.

The `FirstRecordOfANewGroup` variable keeps count of records for each customer. This variable resets at the start of records for each customer, because you set its **Reset type** property to `Group` and its **Reset group** property to `Customer` in steps 8 and 9, respectively. Moreover, the `FirstRecordOfANewGroup` variable is incremented by 1 for each record due to `count` being the value of the **Calculation** property you set in step 7. Therefore, the value of this variable is zero (0) at the start of each customer and increments as JasperReports processes records of that customer.

Now look at the `($V{FirstRecordOfANewGroup}==0) ? ($V{TOC} + $F{CustomerName} + " " + $V{PAGE_NUMBER}+"
") : $V{TOC}` expression for the TOC variable that you authored in step 18. This is a complex expression which does the following:

1. It checks whether it is the start of a new customer.
2. If it is the start of a new customer, the TOC variable concatenates the customer name and current page number into its current value.
3. If it is not the start of a new customer, the TOC variable simply holds its previous contents without any change.

This way, the TOC variable builds the complete TOC of the report. Note that you dropped the TOC variable as a field into the **Title** section in step 24 and set the **Evaluation Time** property of the `$V{TOC}` field to `Report`. This means that the value of the TOC variable will be copied to the `$V{TOC}` field only at the end of the report processing, so that the complete TOC is displayed.

Similarly, in step 26 you selected `html` as the value of the **Markup** property of the `$V{TOC}` field because you used a `
` HTML tag in the expression for the TOC variable earlier in step 18.

Applying a style to your simple TOC

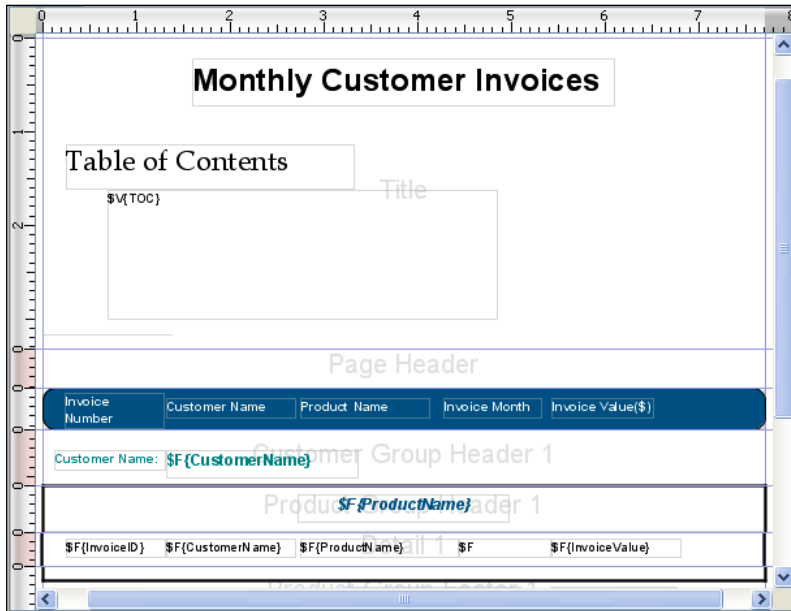
This recipe teaches you how to play with variables to give a stylish and professional look to the simple Table of contents (TOC) you designed in the *Creating a simple one-page TOC for your report* recipe. For this purpose, you will use four pairs of variables and text fields to design a stylish TOC. You will configure each pair individually.

Getting ready

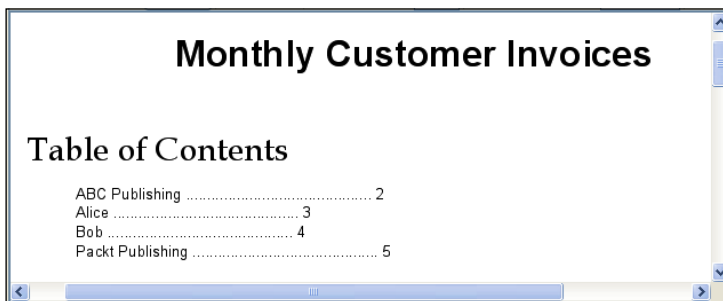
Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb6` and copy sample data for this recipe into the database.

How to do it...

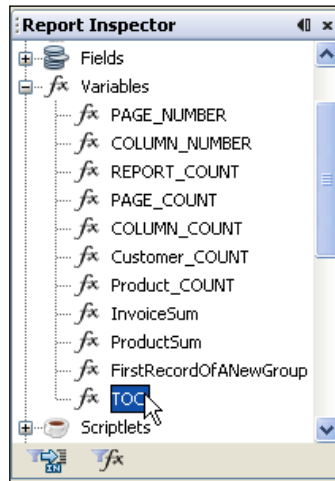
1. Open the `StylishTOCReport.jrxml` file from the `Task3` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in **Title**, **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, and **Product Group Footer 1** sections, as shown in the following screenshot:



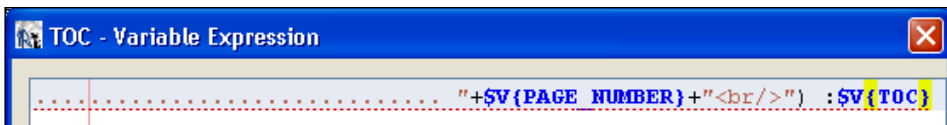
2. Switch to the **Preview** tab and you will see a simple TOC on the first page of your report.



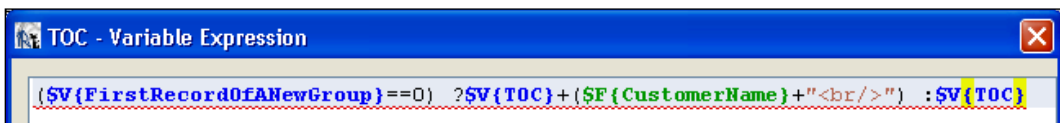
- Switch back to the **Designer** tab. Double-click the **Variables** node in the **Report Inspector** window on the left side of your report. The **Variables** node will expand showing all its children (that is, all variables present in the report), as shown in the next screenshot.



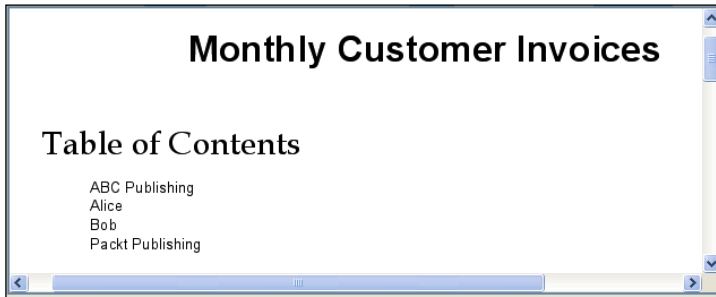
- Select the variable named **TOC**, and its **Properties** window will appear below the **Palette** of components. Select the **Variable Expression** property and click the button beside it. A **Variable Expression** window will open.



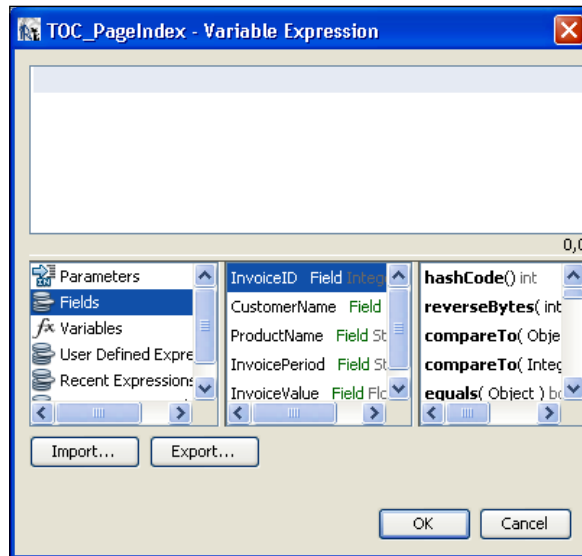
- Delete the existing expression and type `($V{FirstRecordOfANewGroup}==0) ? $V{TOC} + ($F{CustomerName} + "
") : $V{TOC}` as the expression in the **Variable Expression** window.



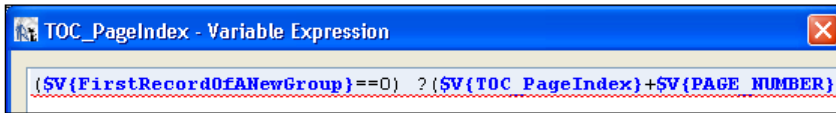
- Switch to the **Preview** tab and you will see that the dotted lines and page numbers in the TOC are gone, as shown in the following screenshot:



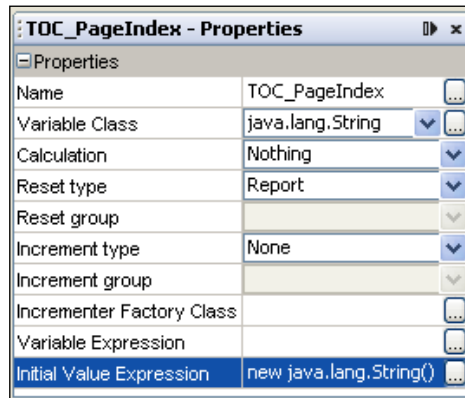
- Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Variable** option.
- A new variable named **variable1** will be added at the end of the variables list.
- While **variable1** is selected, find the **Name** property in the **Properties** window below the **Palette** of components and change its value to `TOC_PageIndex`. Now the name of the **variable1** variable will change to **TOC_PageIndex**.
- Select the **Variable Class** property and change its value to `java.lang.String`.
- Select the **Reset type** property and change its value to `Report`.
- Select the **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open, as shown in the following screenshot:



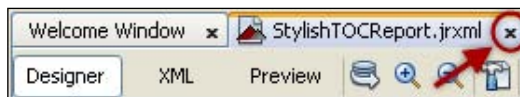
13. Type `($V{FirstRecordOfANewGroup}==0) ? ($V{TOC_PageIndex}+$V{PAGE_NUMBER} + "
") : $V{TOC_PageIndex}` as the expression in the **Variable Expression** window, as shown below. Press the **OK** button.



14. Select the **Initial Value Expression** property and type `new java.lang.String()` as its value.

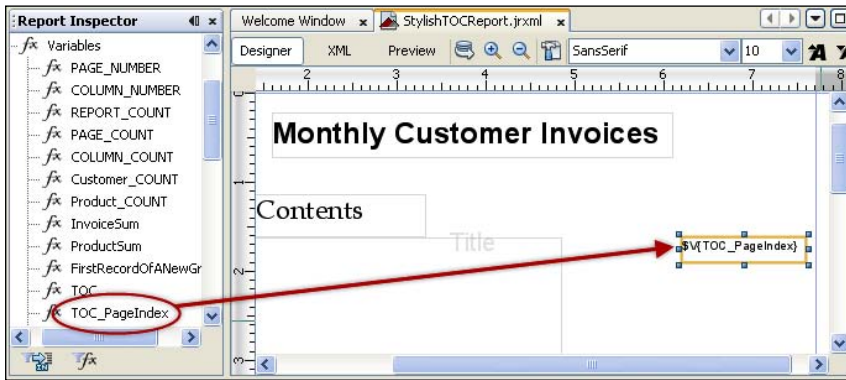


15. Save your report by choosing **Save** option from the **File** menu. Click the close icon on right of your report title to close the report, as indicated in the next screenshot:



16. Re-open the saved `StylishTOCReport.jrxml` file. You have to close and re-open your `.jrxml` file every time you play with the **Initial Value Expression** property of a variable. This is due to a bug in iReport version 3.6.0.

17. Double-click the **Variables** node in the **Report Inspector** window; it will show all its children (that is, all variables present in the report). Drag-and-drop, **TOC_PageIndex** variable from the **Variables** node to the far right of the **Title** section beside the `$V{TOC}` text field of your report, as shown in the next screenshot:

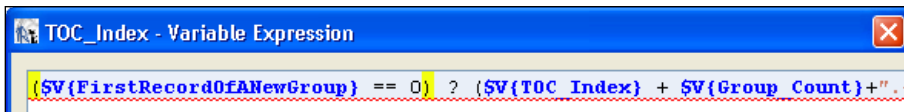


18. Click the `$V{TOC_PageIndex}` text field in the **Title** section. Its properties will appear in the **Properties** window below the **Palette**. Find the **Height** property and set its value as 100.
19. Find the **Evaluation Time** property of the `$V{TOC_PageIndex}` text field and select **Report** as its value from the drop-down list beside it. Similarly, look for the **Markup** property and select **html** as its value from the drop-down list beside it.
20. Switch to the **Preview** tab. You will see the TOC now showing customer names as well as page numbers.

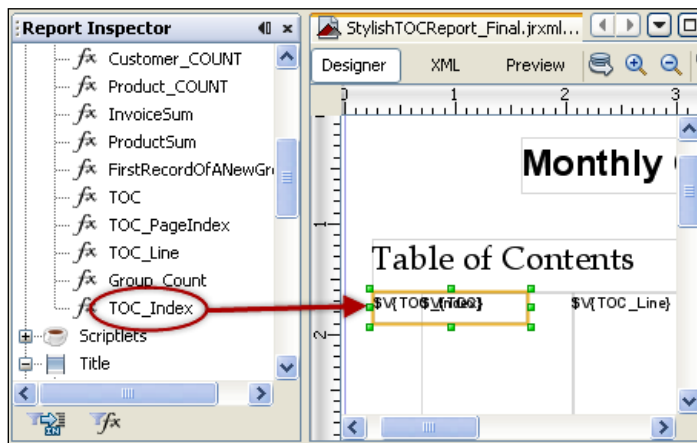
Monthly Customer Invoices	
Table of Contents	
ABC Publishing	2
Alice	3
Bob	4
Packt Publishing	5

21. Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Variable** option.
22. A new variable named **variable1** will be added to the end of the variables list.

23. While **variable1** is selected, find the **Name** property in the **Properties** window below the **Palette** of components and change its value to **TOC_Line**. Now the name of the **variable1** variable will change to **TOC_Line**.
24. Click the **Initial Value Expression** property and type `new java.lang.String()` as its value.
25. Select the **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open.
26. Type `($V{FirstRecordOfANewGroup} == 0) ? ($V{TOC_Index} + $V{Group_Count}) + ".
") : $V{TOC_Index}` as the **Variable Expression** value in the expression editor window.

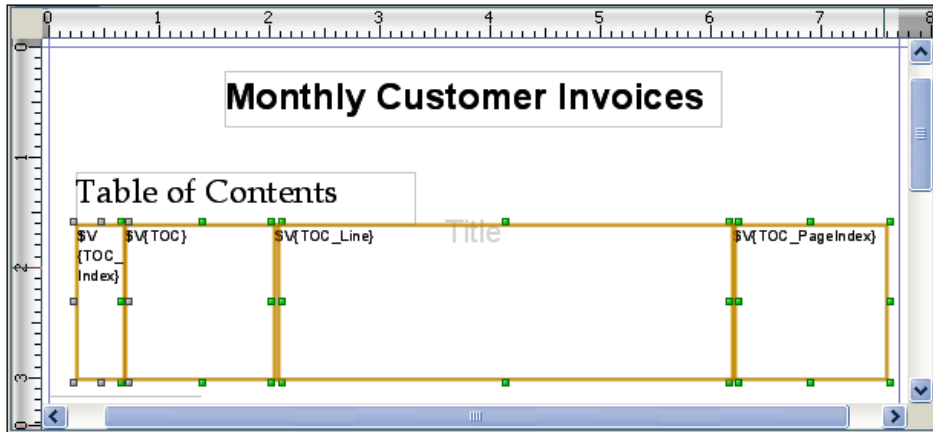


27. Repeat steps 15 and 16 to close and re-open the `StylishTOCReport.jrxml` file.
28. Drag-and-drop the **TOC_Index** variable from the **Variables** node into the **Title** section just to the left beside the `$V{TOC_Index}` text field, as the shown in the next screenshot:



29. While the `$V{TOC_Index}` text field is selected, find the **Width** property and set its value as 30. Similarly, set 100 as the value of the **Height** property.
30. Find the **Evaluation Time** property in the **Properties** window under the **Text field properties** section and select **Report** as its value from the drop-down list beside it.
31. Find the **Markup** property under the **Text properties** section in the **Properties** window and select **html** as its value from the drop-down list beside it.

32. Now adjust the placement of all the four text fields in the **Title** section close to each other by using mouse or arrow keys. Finally, the **Title** section will look as shown in the following screenshot:



33. Click on each of the four text fields one by one and set 12 as the value of the **Size** property in the **Properties** window.
34. Switch to the **Preview** tab, you will see that the TOC is showing customer names and page numbers, as shown in the next screenshot:

Monthly Customer Invoices		
Table of Contents		
1.	ABC Publishing	2
2.	Alice	3
3.	Bob	4
4.	Packt Publishing	5

How it works...

In this recipe you learned a trick of using pairs of variables and text fields to produce a stylish TOC. The recipe uses four pairs:

1. A variable named `TOC` and its corresponding `$V{TOC}` text field form the first pair, which was already present in the `StylishTOCReport.jrxml` file when you opened it in step 1. This pair displays the customer names. Refer to the earlier recipe *Creating a simple one-page TOC for your report* of this chapter to learn how this pair works.
2. A variable named `TOC_PageIndex` and its corresponding `$V{TOC_PageIndex}` text field form the second pair, which you configured in steps 7 to 18. This pair displays all the page numbers.
3. A variable named `TOC_Line` and its corresponding `$V{TOC_Line}` text field form the third pair, which you configured in steps 21 to 32. This pair displays the dotted line between customer names and page numbers.
4. The last pair of variable named `TOC_Index` and its corresponding `$V{TOC_Index}` text field form the fourth pair, which you configured in steps 34 to 51. This pair uses a supporting variable named `Group_Count`, which holds the count for the customer names, to display incremental index numbers for the TOC.

The common theme between all the pairs of variables and text fields is the use of string concatenation. The variable of each pair keeps on building its content or value for each customer. At the end of report processing, the variable simply copies its value into its corresponding text field. As you want the variable to copy its value into the field only at the end of report processing, you set `Report` as the value of the **Evaluation Time** property for all text fields. Similarly, because each variable uses a `
` HTML tag in its expression, you have to set `html` as the value of its **Markup** property.

Resetting page numbering with the start of a particular record

You may need to reset the page numbering of your report at certain places. For example, if you are generating a weekly or monthly report of all customer invoices, you may want to reset page numbering in your report with the start of every customer. This is sometimes convenient, as it allows you to generate just one report (that is, a report of all invoices issued in a month), which actually consists of many reports (that is, a report for each of your customers).

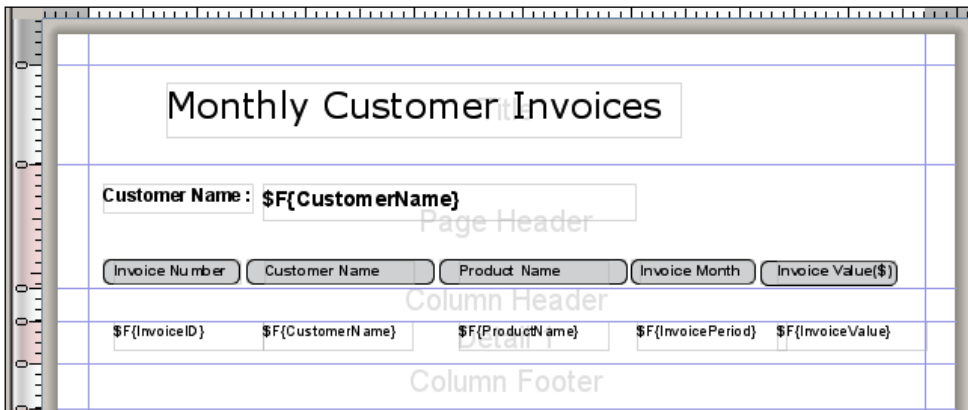
JasperReports offers an interesting feature known as grouping of records, which you can use to do many tricks in report designing. This recipe demonstrates how to use the grouping feature to reset page numbering in your report.

Getting ready

Refer to `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed forward. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb6` and copy sample data for this recipe into the database.

How to do it...

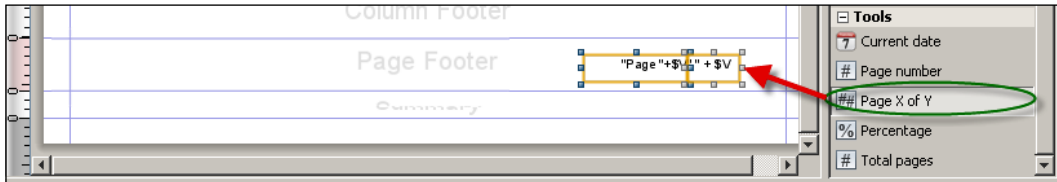
1. Open `ResetPageCount.jrxml` file from the `Task4` folder of the source code of this chapter. The **Designer** tab of iReport shows the report containing a title, a page header, and empty page footer as shown in the following screenshot:



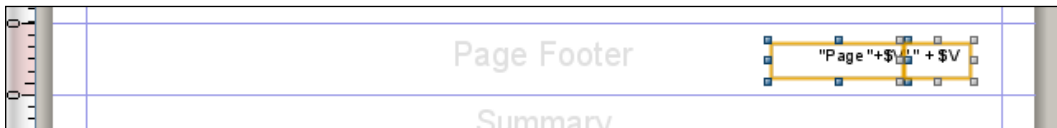
2. Switch to the **Preview** tab to see the preview of your report. You will see a random arrangement of records and no page numbers in your report.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1005	Packt Publishing	Packing Material	Mar09	2050.0
1004	Packt Publishing	Packing Material	Mar09	2050.0
1006	Bob	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5

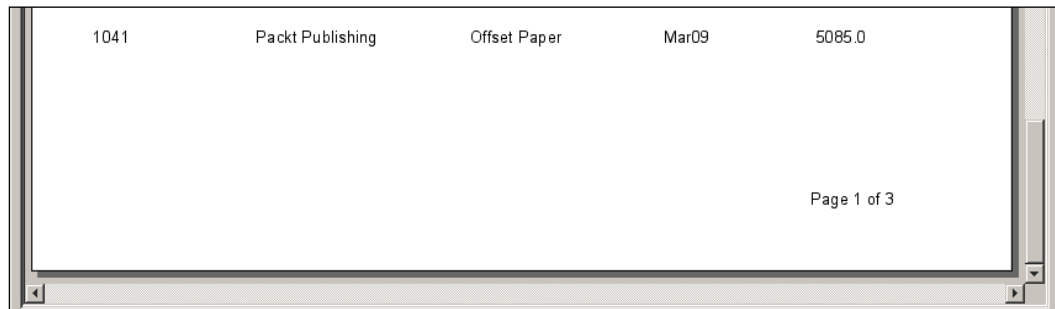
- Switch to the **Designer** tab. Drag a new **Page X of Y** component from the **Tools** section of the **Palette** and drop it into the **Page Footer** section of your report. This will add two **Text Field** components into the **Page Footer** section of your report. The first component will have an expression "Page " + \$V{ PAGE_NUMBER } + " of " and the second component will have an expression + \$V{ PAGE_NUMBER }.



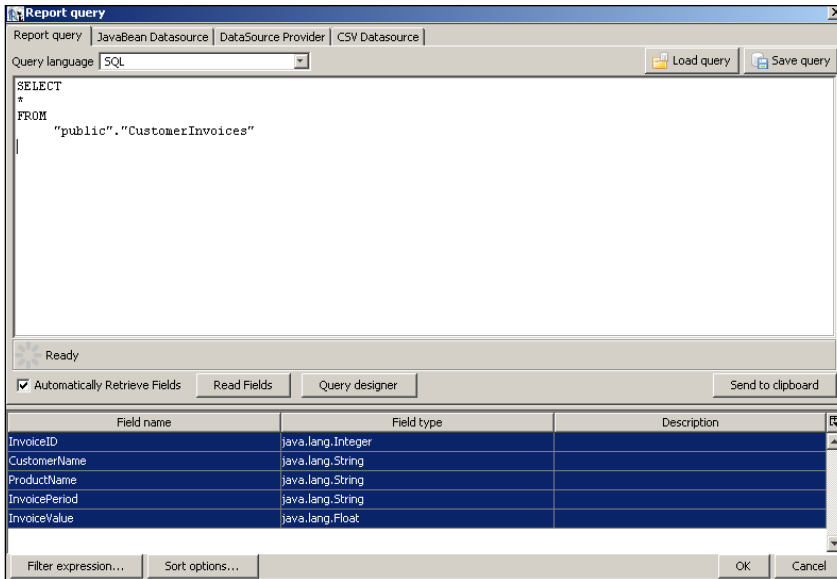
- Select both the **Text Field** components you just dropped into the **Page Footer** section of your report by holding down the **Ctrl** key and left-clicking them one by one. Now release the **Ctrl** key and align the components horizontally and vertically to the top-right position in the **Page Footer** section of your report using the arrow keys of your keyboard, as shown in the next screenshot:



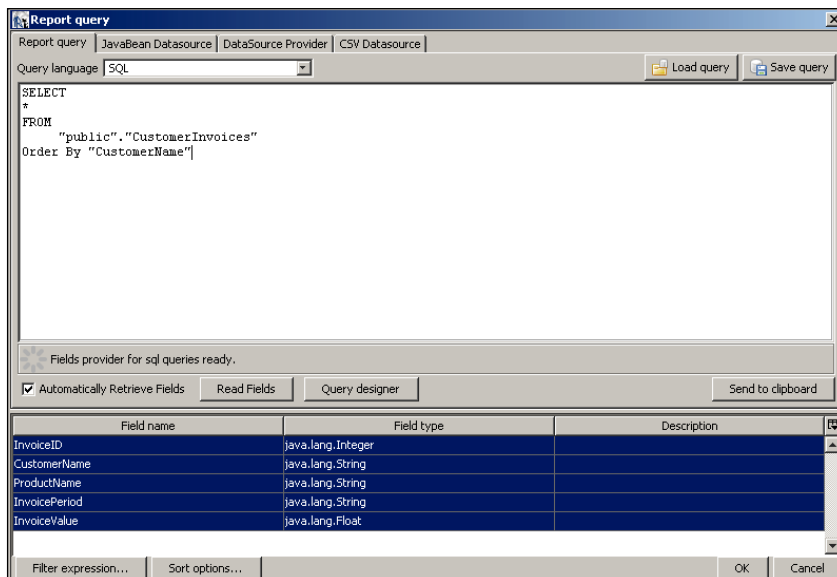
- Switch to the **Preview** tab to see page numbers in your report page footer.



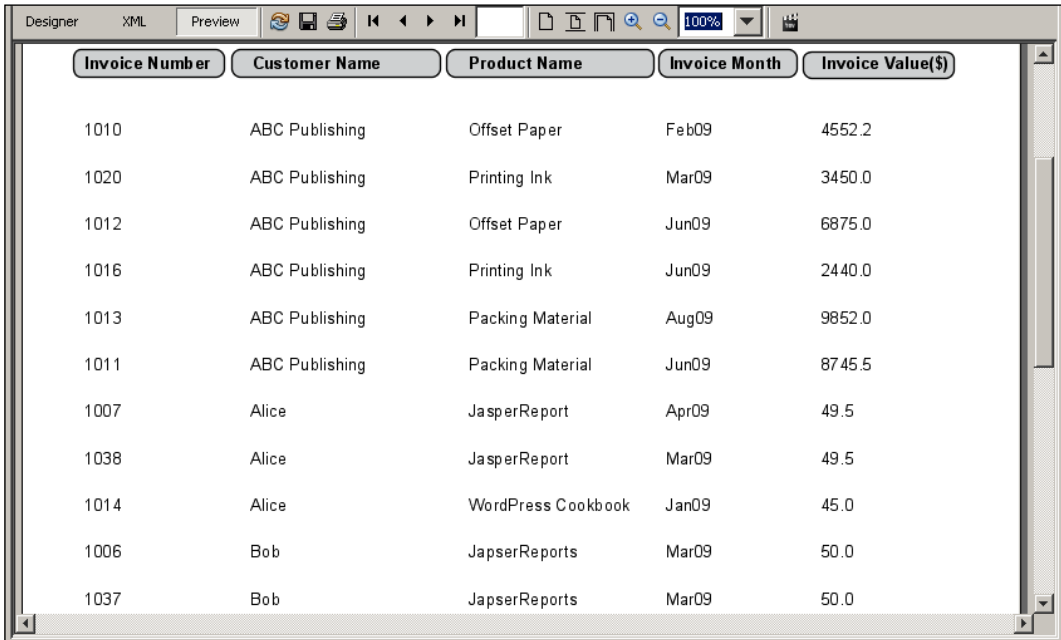
- Switch to the **Designer** tab. Press **Report query** button at the top of the report window. A **Report query** dialog will appear, as shown in the following screenshot:



- While your cursor is in Query editor, press the *Enter* key to bring it to the next blank line and type `Order By "CustomerName"` as shown in the next screenshot. Click the **OK** button.



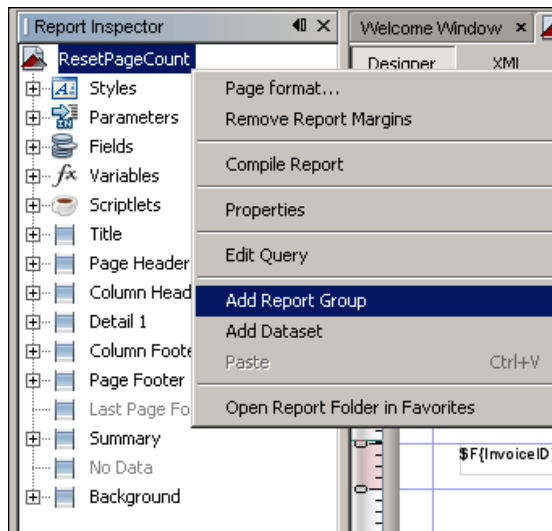
- Switch to the **Preview** tab to see the preview of your report. Compare this preview with the preview of step 2. You will notice that the records are now arranged by order of customer names.



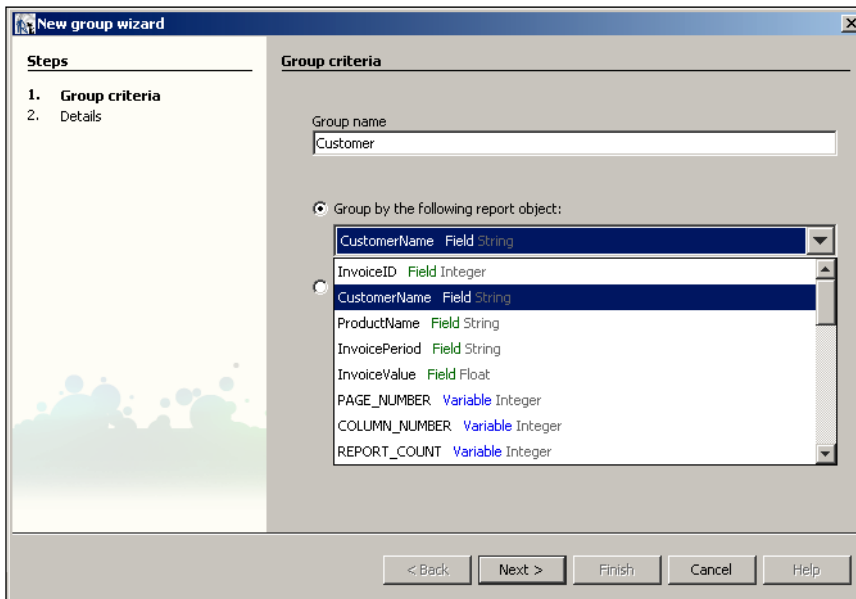
The screenshot shows the JasperReports Preview tab. The report displays a table with 5 columns: Invoice Number, Customer Name, Product Name, Invoice Month, and Invoice Value(\$). The data is sorted by Customer Name, showing records for ABC Publishing, Alice, and Bob.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1013	ABC Publishing	Packing Material	Aug09	9852.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1007	Alice	JasperReport	Apr09	49.5
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1006	Bob	JapserReports	Mar09	50.0
1037	Bob	JapserReports	Mar09	50.0

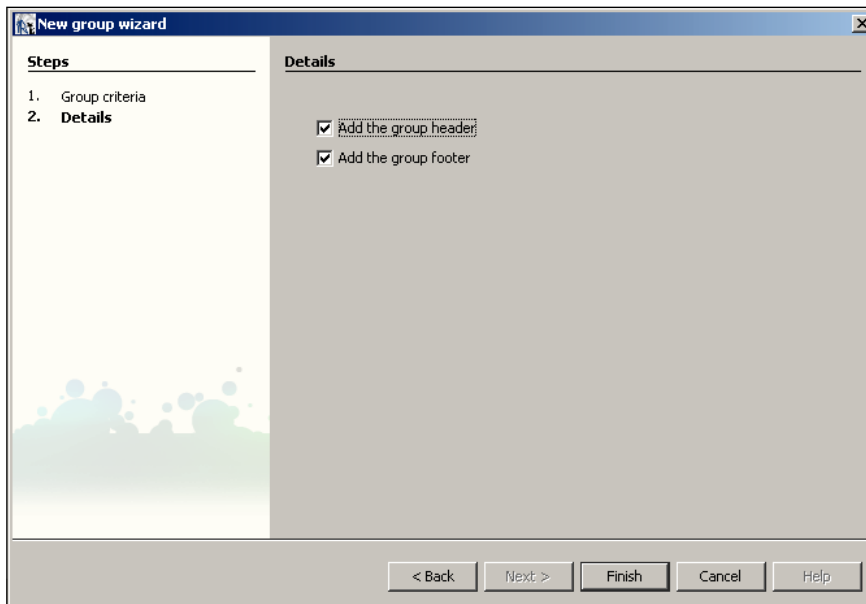
- Switch to the **Designer** tab. Right-click on the top-most element `ResetPageCount` in the **Report Inspector** and click on **Add Report Group**.



10. A **New group wizard** dialog box will appear. Type **Customer** in the **Group name** field and select **CustomerName** Field from the **Group by the following report object:** option, as shown in the following screenshot:



11. Press the **Next** button, and the dialog will change.



12. Press the **Finish** button to dismiss the dialog. You will notice that **Customer Group Header 1** and **Customer Group Footer 1** sections have been added into your report.

Monthly Customer Invoices

Customer Name: \${CustomerName}

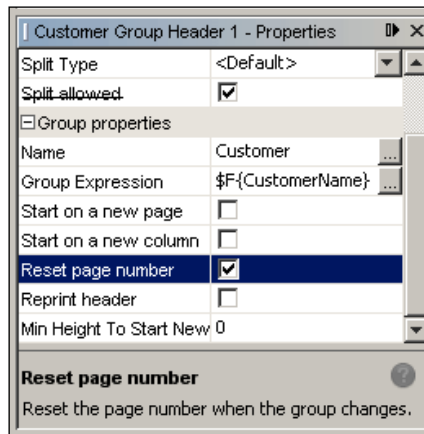
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Group Header 1				
\${InvoiceID}	\${CustomerName}	\${ProductName}	\${InvoicePeriod}	\${InvoiceValue}
Customer Group Footer 1				

13. Switch to the **Preview** tab to see the preview of your report. You will see a clear grouping of records based on customer names, separated by wide spaces, as shown in the next screenshot. Navigate through all the pages and note that the page numbers are counted for the whole report.

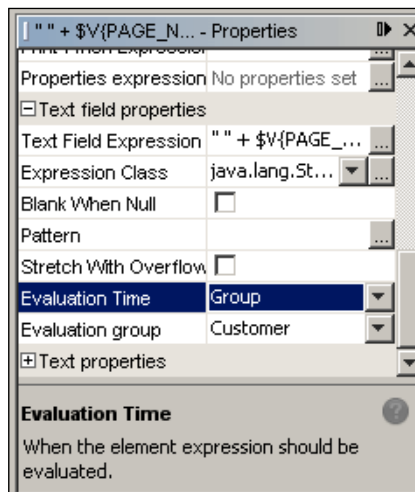
1007	Alice	JasperReport	Apr09	49.5
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1006	Bob	JapserReports	Mar09	50.0
1037	Bob	JapserReports	Mar09	50.0

Page 1 of 3

14. Switch to the **Designer** tab. Click on the **Group Header 1** section of your report and go to the **Group properties** section of **Properties** below **Palette** and select the **Reset page number** property. Mark the checkbox beside it.



15. Now select the second **Text Field** component from step 3, which has the expression + \$V{ PAGE_NUMBER } and go to the **Text field properties** section of **Properties** below **Palette**. Select the **Evaluation Time** property and change its value to **Group**.



16. Switch to the **Preview** tab to see the preview of your report. Press the **Next Page** button at the top of your report to navigate through all pages of your report. Compare this preview with the preview of step 13. You will notice that each customer group starts with a new page, and also the page numbering resets with the start of a new customer.

Customer Name : Packt Publishing					Customer Name : Packt Publishing				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)	Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1021	Packt Publishing	Binding Material	Jan09	2580.0	1005	Packt Publishing	Packing Material	Mar09	2050.0
1022	Packt Publishing	Binding Material	Feb09	4808.0	1004	Packt Publishing	Packing Material	Mar09	2050.0
1023	Packt Publishing	Packing Material	Jun09	5870.0	1008	Packt Publishing	Offset Paper	Jan09	8058.5
1024	Packt Publishing	Binding Material	Jan09	8770.0	1009	Packt Publishing	Offset Paper	Feb09	5085.0
1043	Packt Publishing	Binding Material	Mar09	6522.0	1039	Packt Publishing	Offset Paper	Mar09	8058.5
1025	Packt Publishing	Printing Ink	Aug09	5882.0	1040	Packt Publishing	Offset Paper	Mar09	5085.0
1026	Packt Publishing	Binding Material	Sep09	4550.0	1041	Packt Publishing	Offset Paper	Mar09	5085.0
1027	Packt Publishing	Printing Ink	Oct09	4440.0	1042	Packt Publishing	Printing Ink	Mar09	8500.0
1028	Packt Publishing	Offset Paper	May09	8660.0	1017	Packt Publishing	Printing Ink	Aug09	6540.0
1029	Packt Publishing	Offset Paper	May09	8660.0	1019	Packt Publishing	Printing Ink	Jan09	5240.0
1030	Packt Publishing	Printing Ink	Jan09	5240.0					
1031	Packt Publishing	Printing Ink	Jan09	5240.0					
1032	Packt Publishing	Packing Material	Mar09	5020.25					
1033	Packt Publishing	Offset Paper	Jan09	3000.5					
1034	Packt Publishing	Offset Paper	Jun09	3000.5					
1035	Packt Publishing	Offset Paper	Aug09	3000.5					
1001	Packt Publishing	Packing Material	Mar09	5020.25					
1036	Packt Publishing	Offset Paper	Sep09	3000.5					
1002	Packt Publishing	Offset Paper	Apr09	3000.5					
1003	Packt Publishing	Offset Paper	Mar09	4150.05					

Page 1 of 2

Page 2 of 2

How it works...

You have used the grouping feature of JasperReports in this recipe. A group is specified by a `<group>` tag with three children named `groupExpression`, `groupHeader`, and `groupFooter`. Switch to the **XML** tab to see what a `<group>` tag looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<jasperReport>
  <!-- other JasperReports XML tags -->
  <group name="Customer" isResetPageNumber="true">
    <groupExpression>
      <![CDATA[${F}{CustomerName}]]>
    </groupExpression>
    <groupHeader>
```

```

        <band height="29"/>
    </groupHeader>
    <groupFooter>
        <band height="37"/>
    </groupFooter>
</group>
<pageFooter>
    <band height="43" splitType="Stretch">
        <textField>
            <reportElement x="426" y="7" width="80" height="20"/>
            <textElement textAlignment="Right"/>
            <textFieldExpression class="java.lang.String">
                <![CDATA["Page " + ${PAGE_NUMBER} + " of"]]>
            </textFieldExpression>
        </textField>
        <textField evaluationTime="Group" evaluationGroup="Customer">
            <reportElement x="506" y="7" width="40" height="20"/>
            <textElement/>
            <textFieldExpression class="java.lang.String">
                <![CDATA[" " + ${PAGE_NUMBER}]]>
            </textFieldExpression>
        </textField>
    </band>
</pageFooter>
<!-- other JasperReports XML tags -->
</jasperReport>

```

Recall that you chose `CustomerName` in the **Group by the following report object** option in step 10 of the recipe. This is why you can see `CustomerName` as part of the content of the `<groupExpression>` tag in the code shown above. Grouping is done by whatever field you chose in the **Group by the following report object** option.

Also notice that the `<group>` tag has an attribute named `isResetPageNumber` whose value is `true`. iReport authored this attribute when you checked the checkbox in step 14 of this recipe. As you can guess, this attribute results in resetting the page numbering.

Notice the first `<textField>` tag inside `<pageFooter>` tag in the code shown above. This `<textField>` tag corresponds to the first component of step 3 (with an expression "Page " + `${PAGE_NUMBER}` + " of"). This component displays the page numbering and is reset every time a new group starts.

Look at the second `<textField>` tag that has an `evaluationTime` attribute whose value is `Group`. This means that this page numbering field will be evaluated at the end of a group, therefore showing the total number of pages in a group.

Implementing complex multi-dimensional page numbering

Large multi-page reports normally require comprehensive pagination schemes. For example, a report may be split into sections and you may require complex pagination, which gives both section-specific and report-level page numbering.

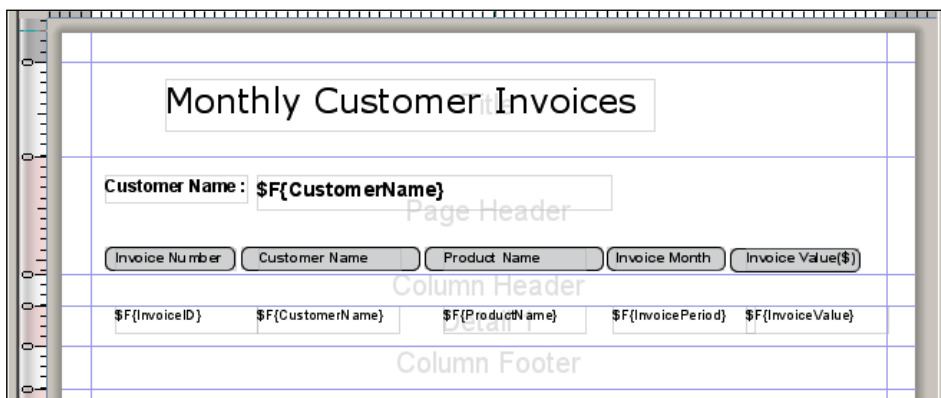
This recipe teaches you how to implement two separate and independent page numbering schemes in the same footer. One scheme works on an individual section of a report, while the other works on the complete report. You will design the page numbering of a customer invoices report. The page numbering will cover the total report as well as page numbering for individual customers.

Getting ready

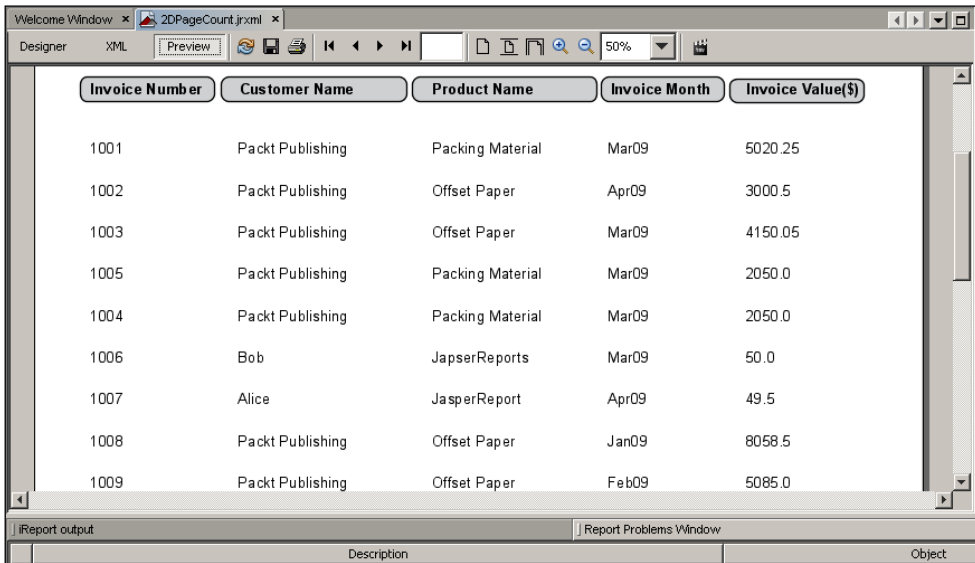
Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb6` and copy sample data for this recipe into the database.

How to do it...

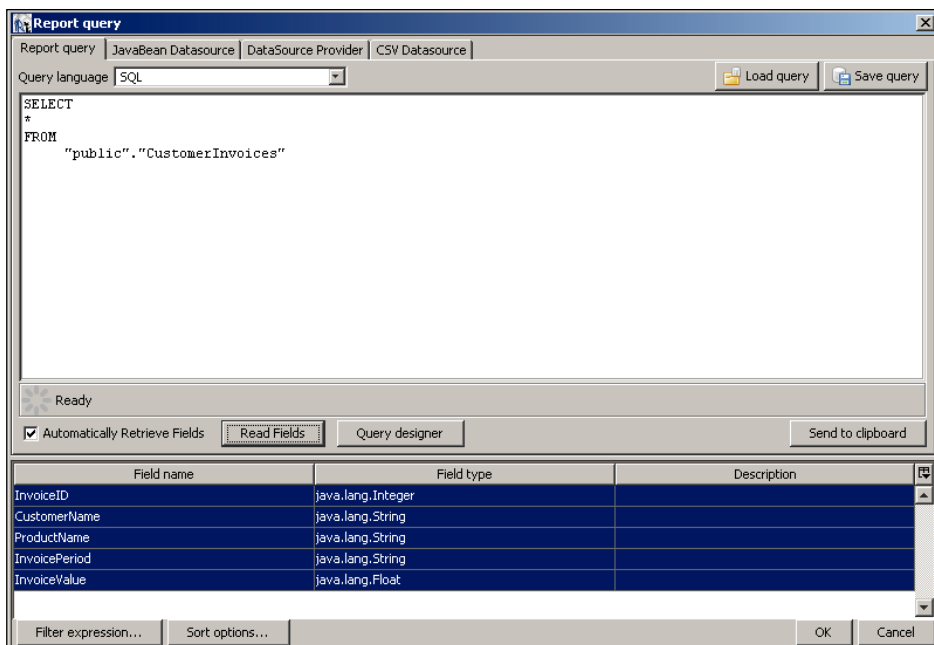
1. Open `2DPageCount.jrxml` file from the `Task5` folder of the source code of this chapter. The **Designer** tab of iReport shows that the report contains a title, a page header, and an empty page footer, as shown:



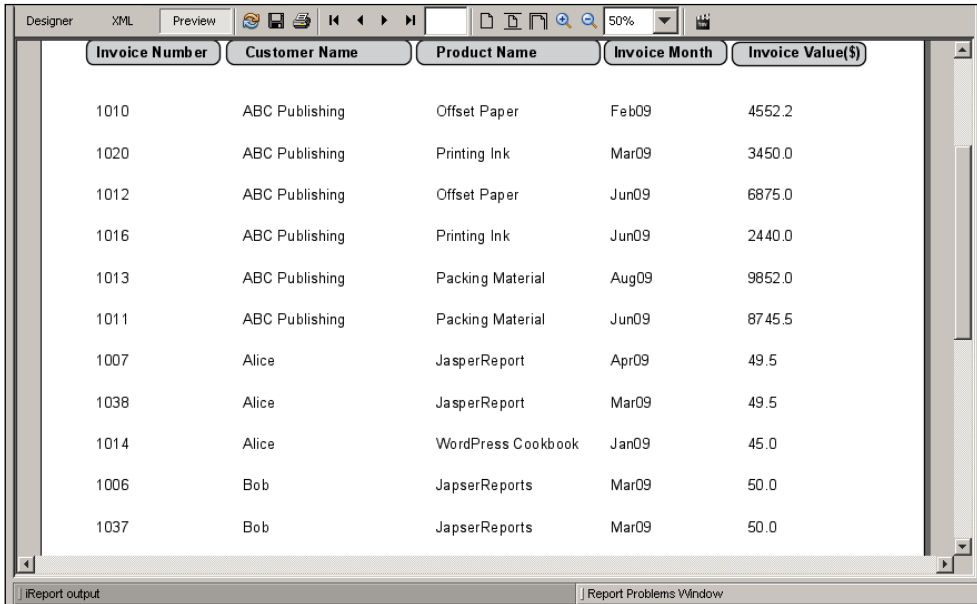
- Switch to the **Preview** tab to see the preview of your report. You will see a random arrangement of records, as shown in the following screenshot:



- Switch to the **Designer** tab. Press the **Report query** button at the top of the report window. A **Report query** dialog will appear.



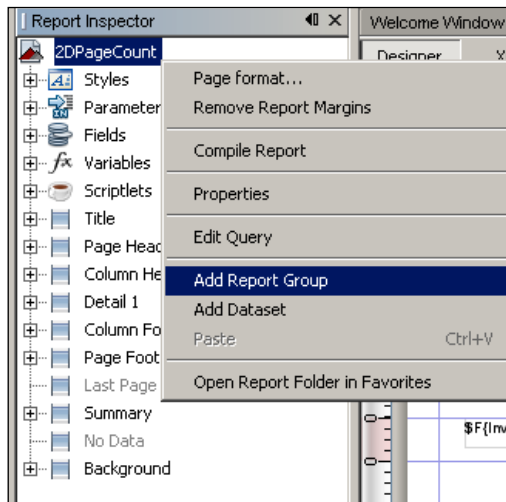
4. While your cursor is in Query editor, press the *Enter* key to bring it to the next blank line type Order By "CustomerName, " and click the **OK** button.
5. Switch to the **Preview** tab to see the preview of your report. You will see that records are now arranged by order of **Customer Name**, as shown in the next screenshot:



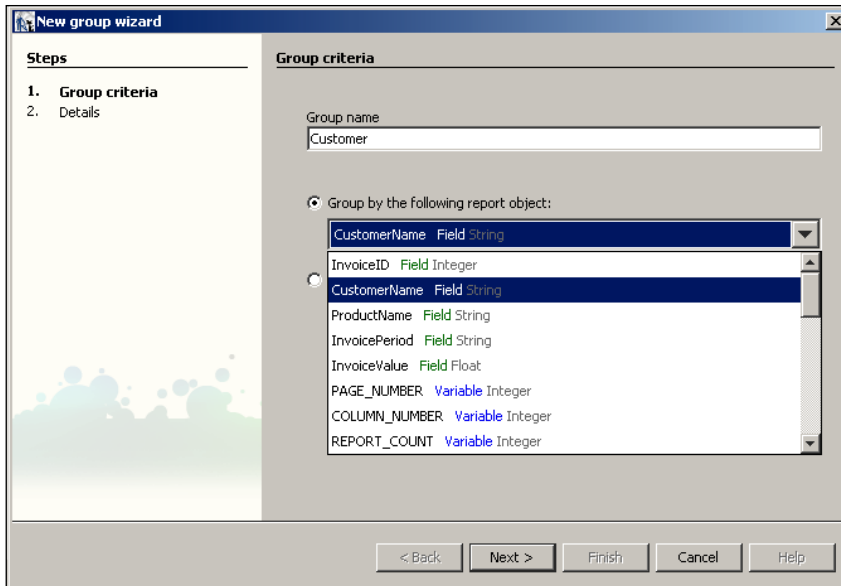
The screenshot shows the iReport Preview tab with a report table. The table has five columns: Invoice Number, Customer Name, Product Name, Invoice Month, and Invoice Value(\$). The records are sorted by Customer Name, showing three customers: ABC Publishing, Alice, and Bob. Each customer has multiple records for different products and months.

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1013	ABC Publishing	Packing Material	Aug09	9852.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1007	Alice	JasperReport	Apr09	49.5
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1006	Bob	JapserReports	Mar09	50.0
1037	Bob	JapserReports	Mar09	50.0

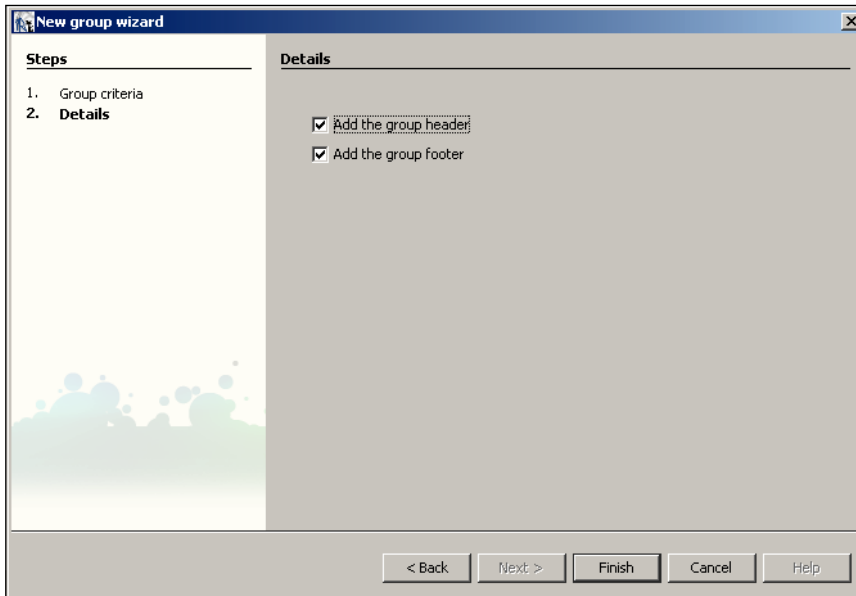
6. Switch to the **Designer** tab. Right-click on the top-most element, **2DPageCount**, in the **Report Inspector** and click on **Add Report Group**.



7. A **New group wizard** dialog box will appear.
8. Type **Customer** as the **Group name** and select **CustomerName** Field as the **Group by the following report object:** option.



9. Press the **Next** button and the dialog will change as below:



10. Press the **Finish** Button to dismiss the dialog. You will notice that **Customer Group Header 1** and **Customer Group Footer 1** sections have been added into your report.

Monthly Customer Invoices

Customer Name:

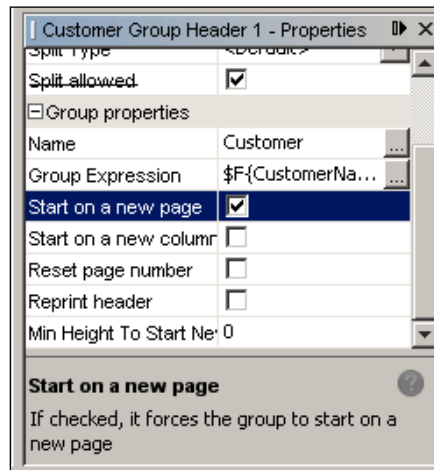
Invoice Number Customer Name Product Name Invoice Month Invoice Value(\$)

\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}
Customer Group Footer 1				

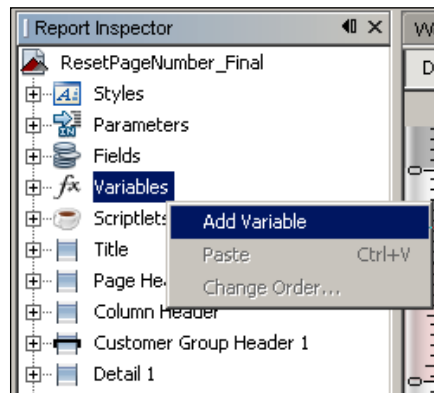
11. Switch to the **Preview** tab to see the preview of your report. You will see clear grouping of records based on customer name separated by wide spaces, as shown in the following screenshot:.

InvoiceID	CustomerName	ProductName	InvoicePeriod	InvoiceValue
1007	Alice	JasperReport	Apr09	49.5
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1006	Bob	JapserReports	Mar09	50.0
1037	Bob	JapserReports	Mar09	50.0

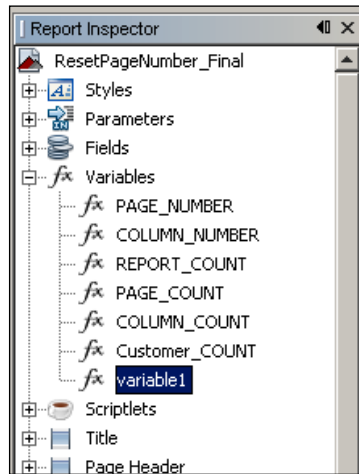
12. Switch to the **Designer** tab. Click on the **Group Header 1** section of your report, go to the **Group properties** section of **Properties** below **Palette**, and select the **Start on a new page** property. Mark the checkbox beside it.



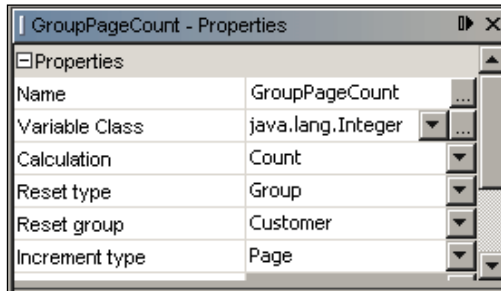
13. Switch to the **Preview** tab to see the preview of your report. Press the **Next Page** button at the top of your report to navigate through all pages of your report. You will notice that each customer group starts with a new page.
14. Switch to the **Designer** tab. Right-click on the **Variables** section of the **Report Inspector** and click **Add Variable**.



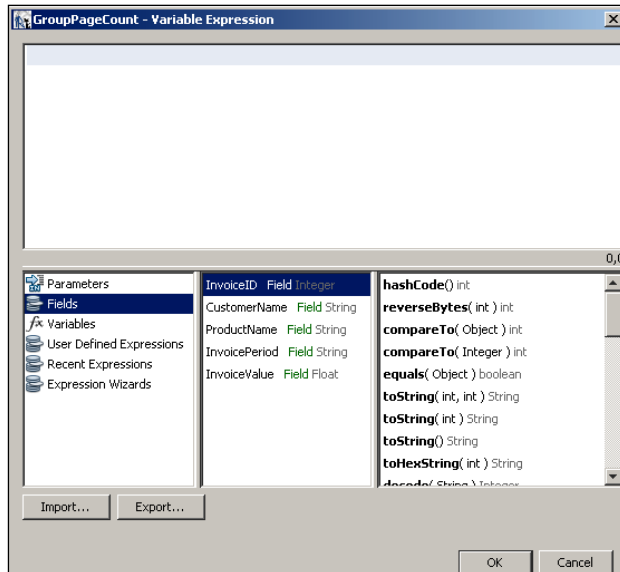
15. A new variable, **variable1**, will be added to the variable list.



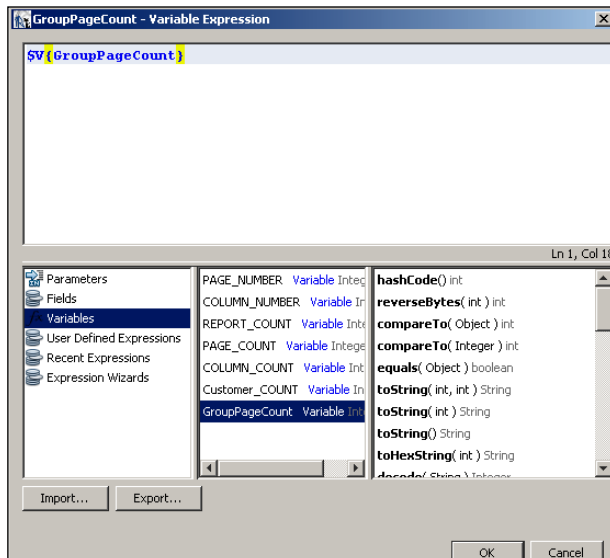
16. While **variable1** is selected, go to **Properties** below **Palette** to change the properties of **variable1**. Select **Name** and change it to **GroupPageCount**.
17. Select **Variable Class** and change it to **java.lang.Integer**.
18. Select **Calculation** and change it to **Count**.
19. Select **Reset Type** and change it to **Group**.
20. Select **Increment type** and change it to **Page**.



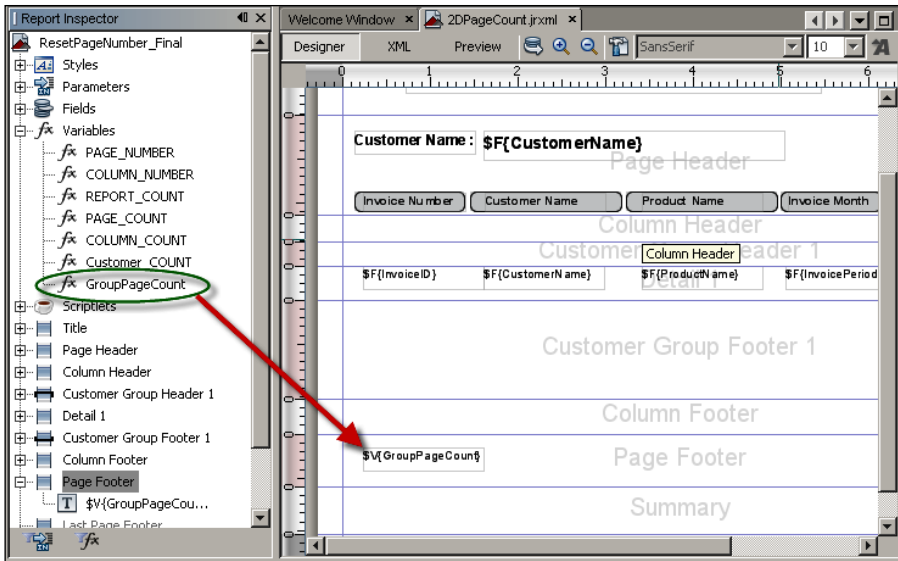
21. Select the **Variable Expression** property (from the **Properties** window below the **Palette** of components) and press **Ctrl + Space Bar**. A **GroupPageCount – Variable Expression** dialog box will appear, as shown in the next screenshot:



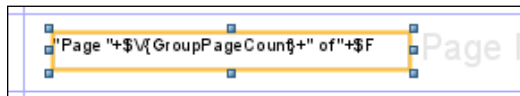
22. Click **Variables** in the first column of the lower-half of the **Expression editor** window. This will list all variables in the adjacent second column. From the list of variables, select the **GroupPageCount** Variable and double-click on it. This will set `$V{GroupPageCount}` in the **Expression editor**, as shown in the following screenshot:



23. Click the **OK** button to dismiss this dialog.
24. From the **Variables** section of the **Report Inspector**, drag **GroupPageCount** and drop it into the **Page Footer** section of your report, as shown in the next screenshot:

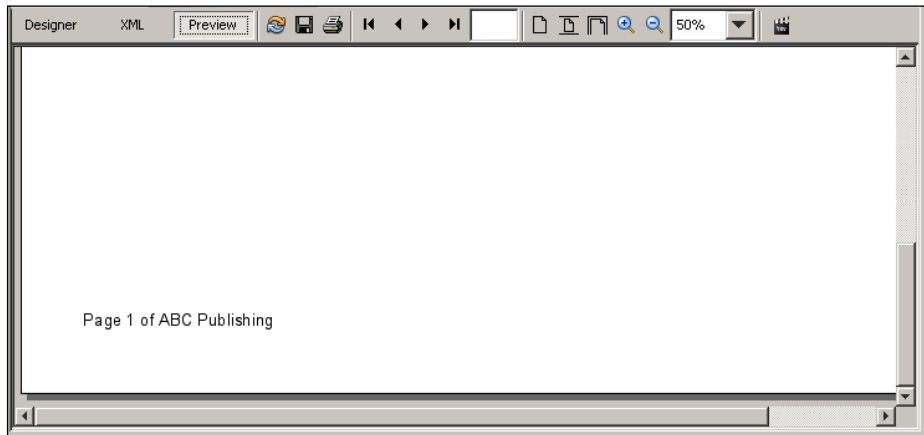


25. A **Text Field** component with an expression `$V{GroupPageCount }` will appear in **Page Footer** section of your report. Double-click the **Text Field** component and replace `$V{GroupPageCount }` with `"Page " + $V{GroupPageCount } + "` of `" + $F{CustomerName }`.
26. Enlarge the rectangular boundary of the **Text Field** component by dragging one of its corners, as shown in the following screenshot:



27. While the **Text Field** component is selected, go to the **Text field properties** section of **Properties** below **Palette**. Select the **Expression Class** property and change it to `java.lang.String`.

28. Switch to the **Preview** tab and scroll the vertical scroll bar down until you see **Page 1 of ABC Publishing** at the bottom-left corner of your report.



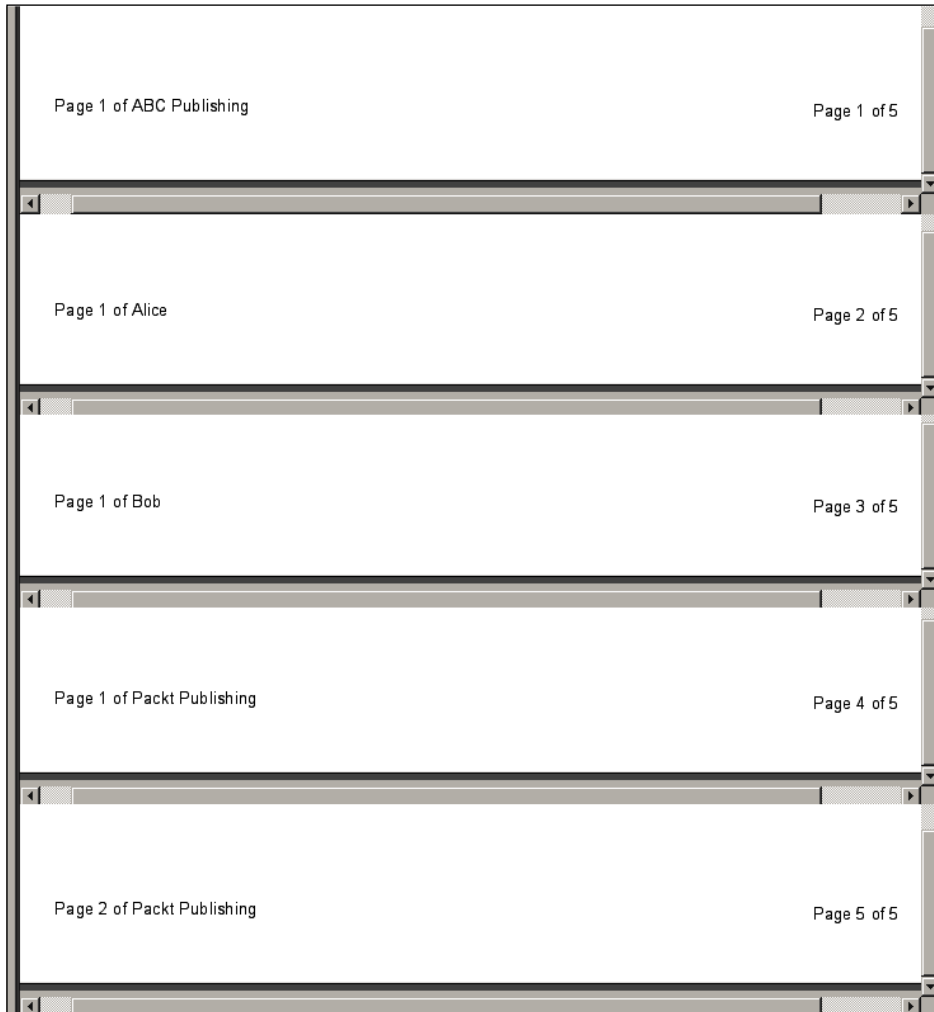
29. Switch to the **Designer** tab. Drag a new **Page X of Y** component from the **Tools** section of the **Palette** and drop it into the **Page Footer** section of your report. This will add two **Text Field** components into the **Page Footer** section of your report. The first component will have an expression "Page " + $\$V\{PAGE_NUMBER\}$ + " of " and the second component will have an expression " " + $\$V\{PAGE_NUMBER\}$.
30. Select both the **Text Field** components you just dropped into the **Page Footer** section of your report by holding down the **Ctrl** key and left-clicking them one by one. Now release the **Ctrl** key and align the components horizontally and vertically to the top-right position in the **Page Footer** section of your report using the arrow keys of your keyboard, as shown in the following screenshot:



31. Switch to the **Preview** tab to see the preview of your report. You will see **Page 1 of ABC Publishing** at the bottom-left and **Page 1 of 5** at the bottom-right of your report.



32. Press the **Next Page** button at the top of your report to navigate through all pages of your report. You will see page numbers **Page 1 of 5**, **Page 2 of 5**, **Page 3 of 5**, **Page 4 of 5**, and **Page 5 of 5**, respectively at the bottom-right of each page and **Page 1 of ABC Publishing**, **Page 1 of Alice**, **Page 1 of Bob**, **Page 1 of Packt Publishing**, and **Page 2 of Packt Publishing**, respectively at the bottom-left of each page. You will notice that your page count at the right-hand side counts pages for the complete report, while on the left-hand side, you are displaying the page count for a group.



How it works...

In this recipe you have used the **Page X of Y** component in step 29 for normal page numbering (displayed in the right-bottom corner of the report) and a user-defined variable `GroupPageCount` for counting page numbering for a group. This group page numbering starts from 1 each time a new customer group starts.

Page X of Y component is actually a collection of **Page number** and **Total pages** components. So when you dropped the **Page X of Y** component, iReport inserted two **Text Field** components into the **Page Footer** section of your report. The first **Text Field** component (**Page number**) had `Now` as the value of its **Evaluation Time** property, which means the value of this **Text Field** will be evaluated at the moment when the footer is being displayed. Therefore the first **Text Field** component will always hold the current page number.

On the other hand, the second **Text Field** component had `Report` as the value of its **Evaluation Time** property, which means the value of this **Text Field** will be evaluated at the end of the report. Therefore the second **Text Field** component will hold the total page count of the report.

For counting page numbers in a group, we used the `GroupPageCount` variable. In steps 18 and 20 you changed the value of its **Calculation** property to `Count` and **Increment type** property to `Page`, which together make a page counter. In this way the `GroupPageCount` variable increments itself with every page change. Moreover, we changed its **Reset type** property to `Group`, which makes it reset each time a group changes. For more details about using user-defined variables, refer to *Chapter 9* of this book, where we have explained in detail the functionality of user-defined variables.

Showing multiple types of data in the same report

JasperReports allows full flexibility of combining different types of data into a single report. The data can even come from multiple databases.

This recipe shows how you will display multiple types of data in a single report. You will use multiple subreports and multiple-detail sections to achieve your goal.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code of this chapter also includes two files named `copySampleDataIntoPGS.txt` and `copyCustomerHistorySampleDataIntoPGS.txt`. The `copySampleDataIntoPGS.txt` file will help you create a database named `jasperdb6` and create a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data for this recipe. Similarly, the `copyCustomerHistorySampleDataIntoPGS.txt` file will help you create a database named `jasperdb6a` and create a table named `CustomerHistory` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data.

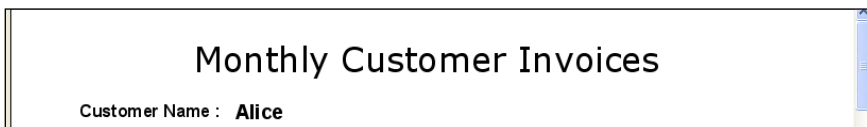
You will be using three JRXML files `MultipleData.jrxml`, `CustHistorySubreport.jrxml`, and `ProductInvoices.jrxml` in this recipe. You will find these files in the `Task6` folder of the source code download of this chapter. The `MultipleData.jrxml` file is the master report, which uses the other two files as subreports. The master report has to refer to all of its subreports using a complete path (you cannot use relative paths). This means you have to copy the three JRXML files to the `C:\JasperReportsCookBookSamples\` folder in your PC. I have hardcoded this complete path in the master report (`MultipleData.jrxml`).

How to do it...

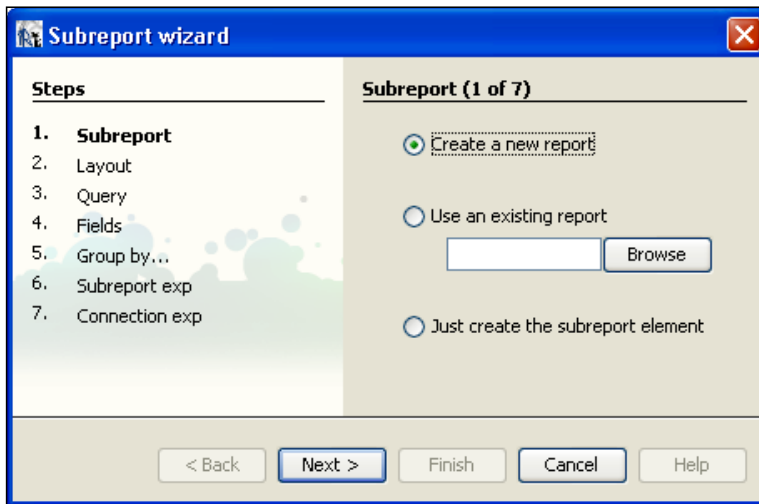
1. Open the `MultipleData.jrxml` file from the `C:\JasperReportsCookBookSamples` folder of your PC. The **Designer** tab of iReport shows a report with data in the **Title** and **Customer Group Header 1** sections, as shown in the following screenshot:



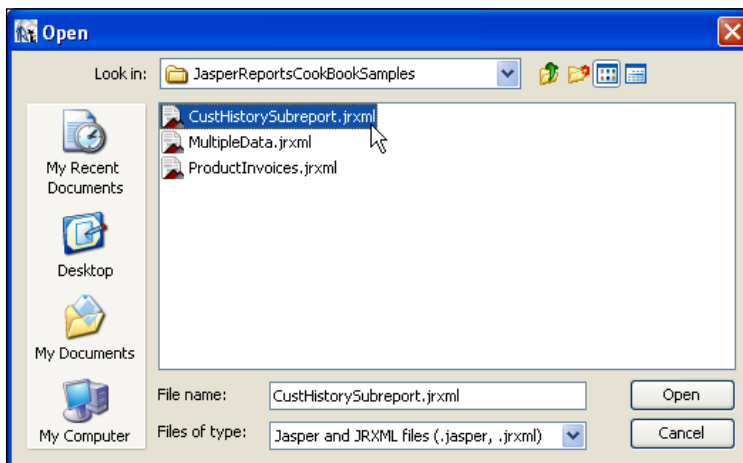
2. Switch to the **Preview** tab and you will see a report with just a title and a customer name.



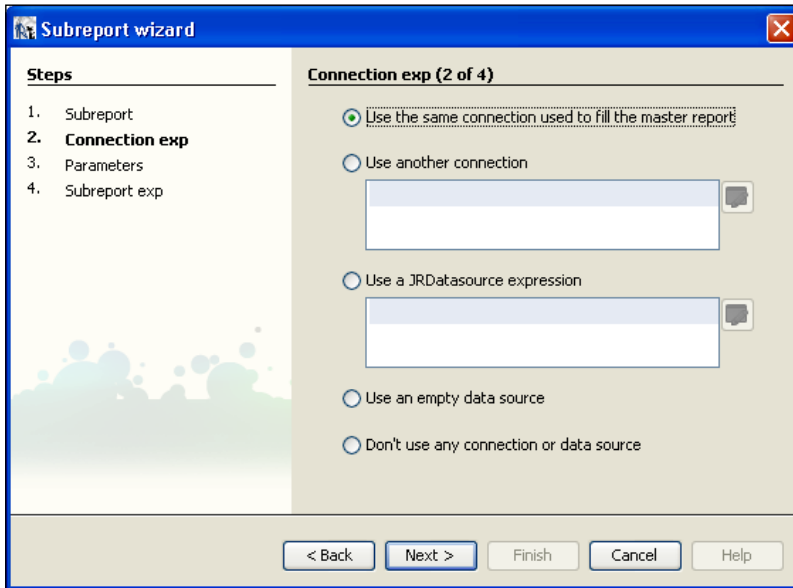
- Switch back to the **Designer** tab. Drag-and-drop a **Subreport** component from the **Palette** into the **Detail 1** section. A **Subreport wizard** dialog will appear, as shown in the next screenshot.



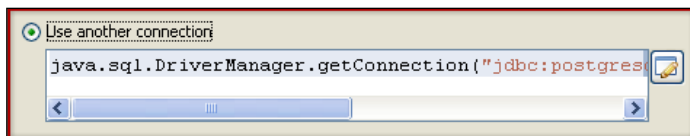
- Select the **Use an existing report** option and click on the **Browse** button to browse to the `CustHistorySubreport.jrxml` file placed in the `C:\JasperReportsCookBookSamples` folder of your PC and press the **Open** button, as shown in the next screenshot. The browse dialog will disappear. Press the **Next** button in the **Subreport wizard** dialog.



5. A **Connection exp (2 of 4)** dialog will appear, as shown. Notice that now the left-side of the **Subreport wizard** shows four steps (**Subreport**, **Connection exp**, **Parameters**, **Subreport exp**). You are at step 2 (**Connection exp**), which is shown in bold.



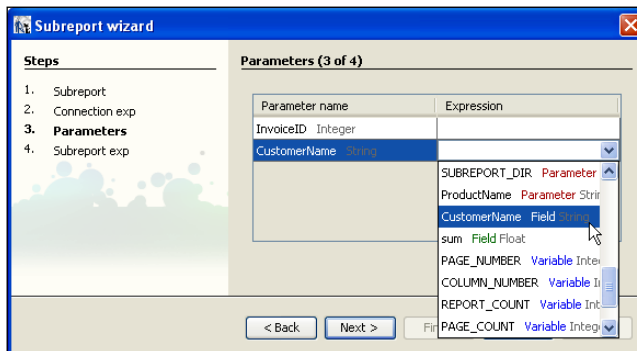
6. Choose the **Use another connection** option and type the `java.sql.DriverManager.getConnection("jdbc:postgresql://localhost:5432/jasperdb6a", "postgres", "postgres")` expression in the area below the **Use another connection** option, as shown in the following screenshot. Press the **Next** button at bottom of the window.



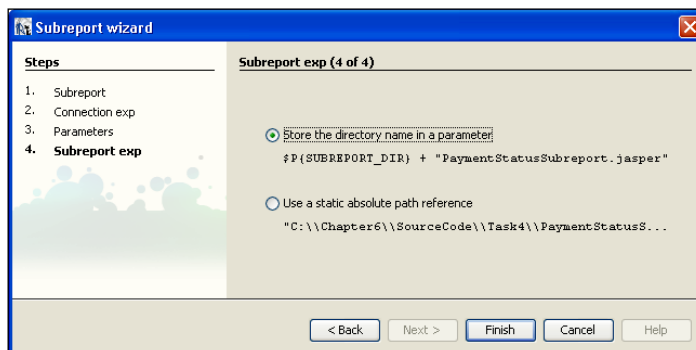
7. A **Parameters (3 of 4)** dialog will appear. This will show InvoiceID and CustomerName parameters of the CustHistorySubreport.jrxml subreport and allows you to enter expressions to map these to a number of elements (that is, **Fields** or **Variables**) of the main MultipleData.jrxml report.



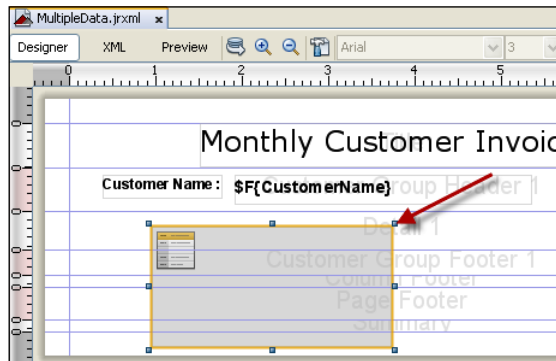
8. Click the **Expression** field against the CustomerName parameter. A drop-down list will open. This will show all elements of the MultipleData.jrxml report. Select CustomerName Field from the list, as shown. Press the **Next** button.



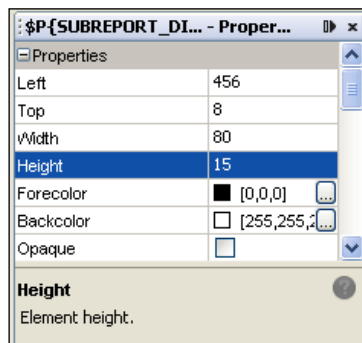
9. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected and press the **Finish** button.



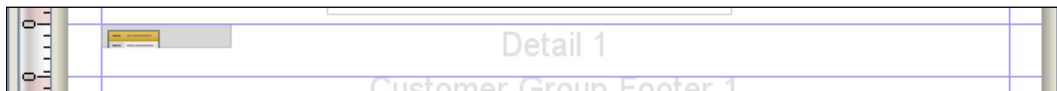
10. The **Subreport** element will be placed in the **Detail 1** section, as shown in the following screenshot:



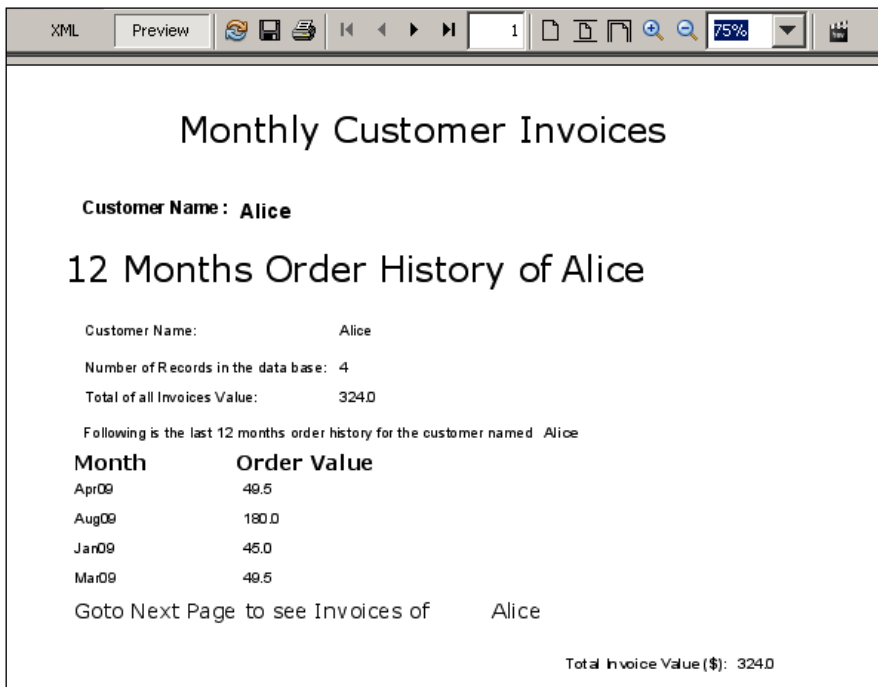
11. Click the **Subreport** element, and its properties will appear in the **Properties** window below the **Palette**. Find the **Width** property and set its value as 80.
12. Click the **Height** property and set its value as 15.



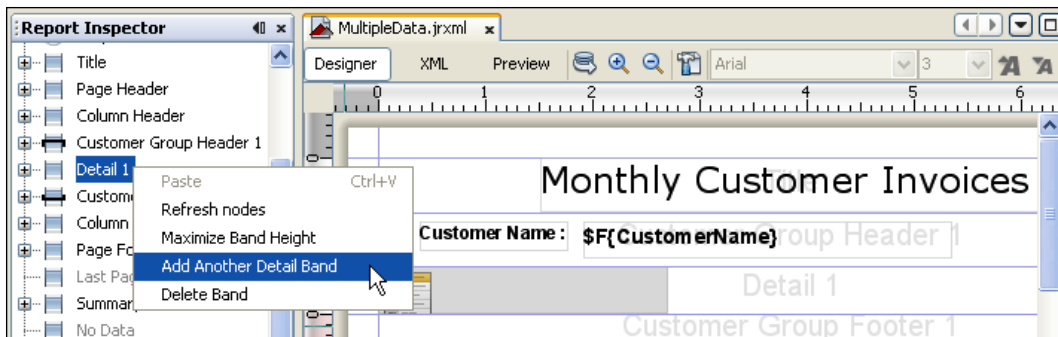
13. The size of the **Subreport** element will become smaller. Align it to the top-left position inside the **Detail 1** section using your mouse, as shown in the next screenshot:



14. Switch to the **Preview** tab, and compare the report that you see this time with the report that you saw earlier in step 2. You will find that now your report shows a 12-month order history for each customer.



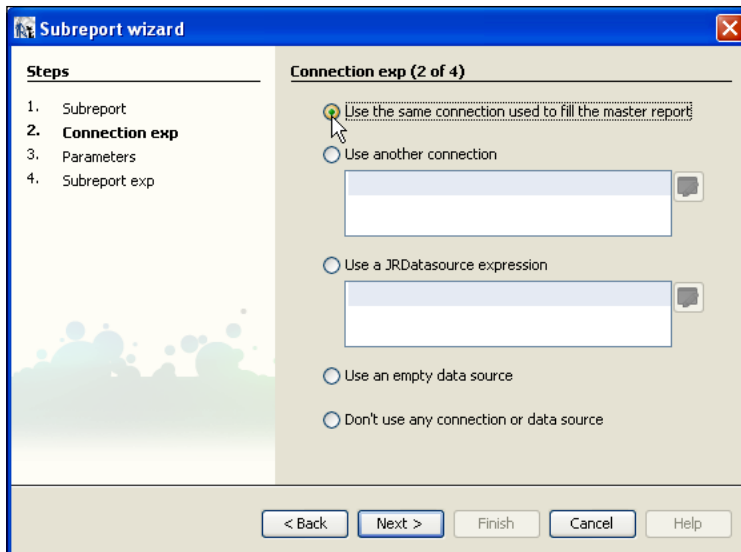
15. Switch back to the **Designer** tab. Right-click on the **Detail1** node in the **Report Inspector** window. A pop-up menu will appear. Click on the **Add Another Detail Band** option in the pop-up menu.



16. A new section named **Detail2** will appear in your report.

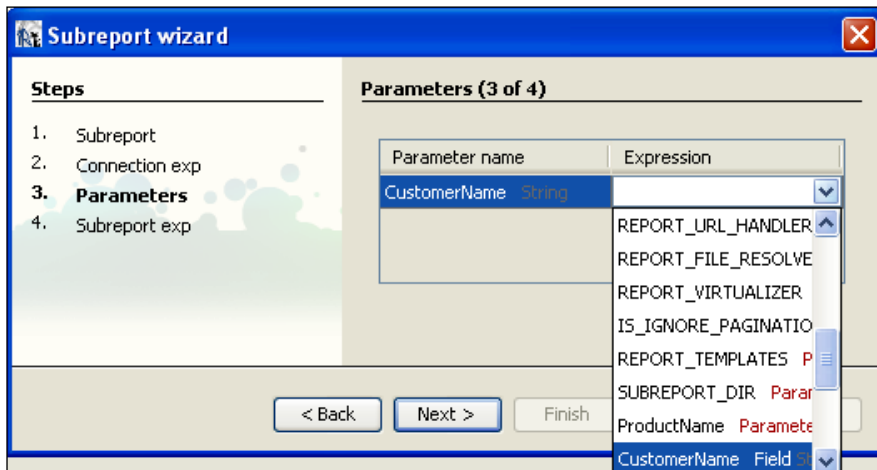


17. Drag-and-drop a **Subreport** component from the **Palette** into the **Detail 2** section. A **Subreport wizard** dialog will appear.
18. Select the **Use an existing report** option and press the **Browse** button to browse to the `ProductInvoices.jrxml` file placed in the `C:\JasperReportsCookBookSamples` folder of your PC and press **Open** button. The browse dialog will disappear. Press the **Next** button in the **Subreport wizard** dialog.
19. A **Connection exp (2 of 4)** dialog will appear. Choose the **Use the same connection used to fill the master report** option as shown. Press the **Next** button at the bottom of the window.

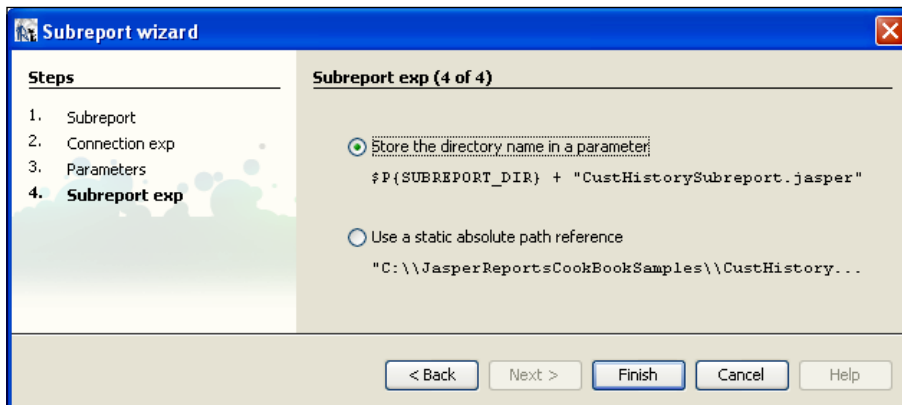


20. A **Parameters (3 of 4)** dialog will appear. This will show the `CustomerName` parameter of the `ProductInvoices.jrxml` subreport and allows you to enter expressions to map these to a number of elements (that is, **Fields** or **Variables**) of the main `MultipleData.jrxml` report.

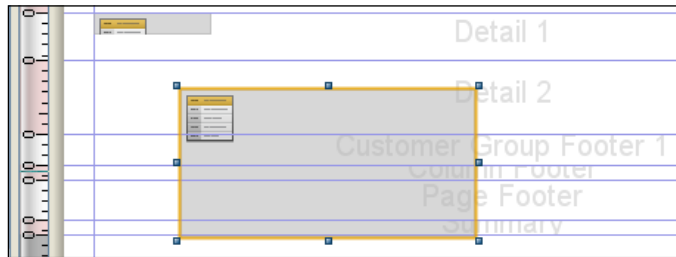
21. Click the **Expression** field beside the CustomerName parameter. A drop-down list will open. This will show all elements of the `MultipleData.jrxml` report. Select `CustomerName Field` from the list. Click the **Next** button.



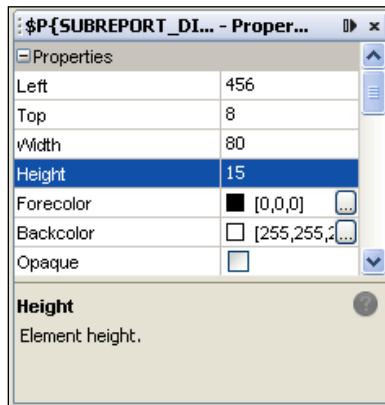
22. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected and click the **Finish** button.



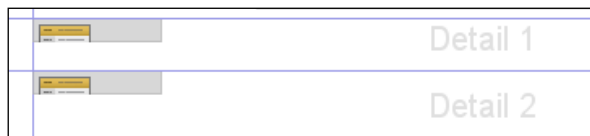
23. The **Subreport** element will be placed in the **Detail 2** section, as shown in the next screenshot:



24. Click this **Subreport** element (of the **Detail2** section), and its properties will appear in the **Properties** window below the **Palette**. Find the **Width** property and set its value as 80.
25. Click the **Height** property and set its value as 15.



26. The size of the **Subreport** element will become smaller. Align it to the top-left position inside the **Detail2** section using your mouse.



27. Switch to the **Preview** tab, and compare the report that you see this time with the report that you saw earlier in step 16. You will find that now your report shows customer invoices in addition to the order history, as shown in the following screenshot:

The screenshot shows a JasperReports preview window with the title "Monthly Customer Invoices". The customer name is "Alice". The report displays a summary of the last 12 months of order history for Alice, including the number of records (4) and the total value of all invoices (\$324.0). Below this, a table lists the last 12 months of order history for Alice.

Month	Order Value
Apr09	49.5
Aug09	180.0
Jan09	45.0
Mar09	49.5

Below the table, there is a link to "Goto Next Page to see Invoices of Alice".

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1007	Alice	JasperReport Cookbook	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1038	Alice	JasperReport Cookbook	Mar09	49.5
1057	Alice	JasperReports for Java	Aug09	180.0

Total Invoice Value (\$): 324.0

How it works...

You have learned to combine different types of data from different sources into a single report. For this purpose, you inserted two subreports into a master report. Each of your subreports uses its own database connection, which you authored in step 6 for your first subreport and step 19 for your second subreport.

Note that the first subreport uses a database connection that is different from the master report's database connection. This is why you chose the **Use another connection** option in step 6.

The second subreport uses the same database connection as the master report. So you chose the **Use the same connection used to fill the master report** option in step 19.

Also notice that you passed the `CustomerName` field as parameter to the first subreport in steps 7 and 8. Similarly, you passed the same `CustomerName` field as parameter to the second subreport in steps 20 and 21. This way, both subreports provide their specific types of data for a specific customer at a time. The master report calls each subreport for different customers, one at a time, and organizes the presentation of all data in a single multi-page report.

Managing pagination of multiple types of data in a report

JasperReports allows full flexibility of managing and interlinking of pagination for data spanning multiple subreports. The subreports may contain different types of data and may serve as building blocks for a single master report.

In this recipe you will learn how to make complex pagination for multiple types of data reports and interlinking pagination of each datatype using built-in features of JasperReports.

This recipe also teaches you how to use data-specific and general pagination in parallel with each other. Data-specific pagination is contributed by each building block (subreport), while the general pagination is contributed by the master report.

Getting ready

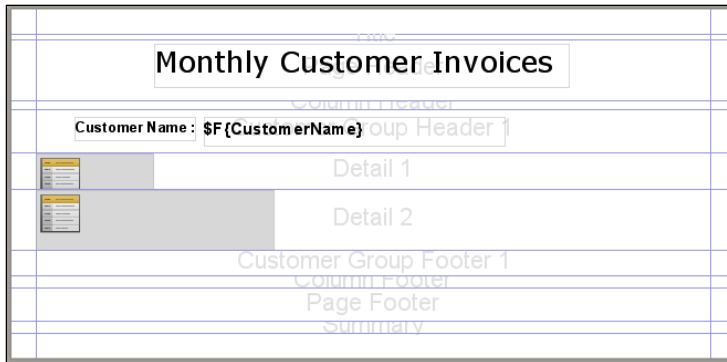
Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code of this chapter also includes two files named `copySampleDataIntoPGS.txt` and `copyCustomerHistorySampletDataIntoPGS.txt`. The `copySampleDataIntoPGS.txt` file will help you create a database named `jasperdb6` and create a table named `CustomerInvoices` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data for this recipe. Similarly, the `copyCustomerHistorySampletDataIntoPGS.txt` file will help you create a database named `jasperdb6a` and create a table named `CustomerHistory` with five columns (`InvoiceID`, `CustomerName`, `InvoicePeriod`, `ProductName`, and `InvoiceValue`) and copy sample data.

You will be using three JRXML files `MultipleData.jrxml`, `CustHistorySubreport.jrxml` and `ProductInvoices.jrxml` in this recipe. You will find these files in the `Task6` folder of the source code download of this chapter. The `MultipleData.jrxml` file is the master report, which uses the other two files as subreports. The master report has to refer to all its subreports using a complete path (you cannot use relative paths). This means you have to copy the three JRXML files to the `C:\JasperReportsCookBookSamples\` folder in your PC. I have hard-coded this complete path in the master report (`MultipleData.jrxml`).

How to do it...

1. Open the `MultipleData.jrxml` file from the `C:\JasperReportsCookBookSamples\` folder in your PC. The **Designer** tab of iReport shows a report with data in the **Title**, **Customer Group Header 1**, **Detail1**, and **Detail2** sections, as shown in the following screenshot:



2. Switch to the **Preview** tab and you will see a report with 12-months order history and invoices for a number of customers.

Preview

Monthly Customer Invoices

Customer Name : Alice

12 Months Order History of Alice

Customer Name: Alice

Number of Records in the data base: 4

Total of all Invoices Value: 324.0

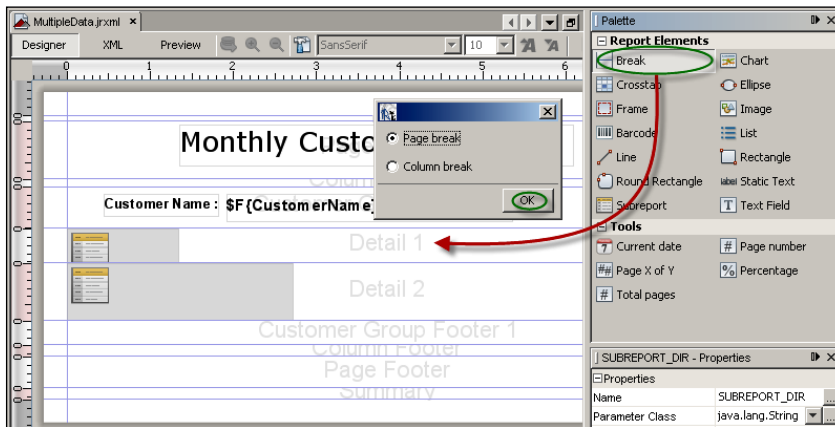
Following is the last 12 months order history for the customer named Alice

Month	Order Value
Apr09	49.5
Aug09	180.0
Jan09	45.0
Mar09	49.5

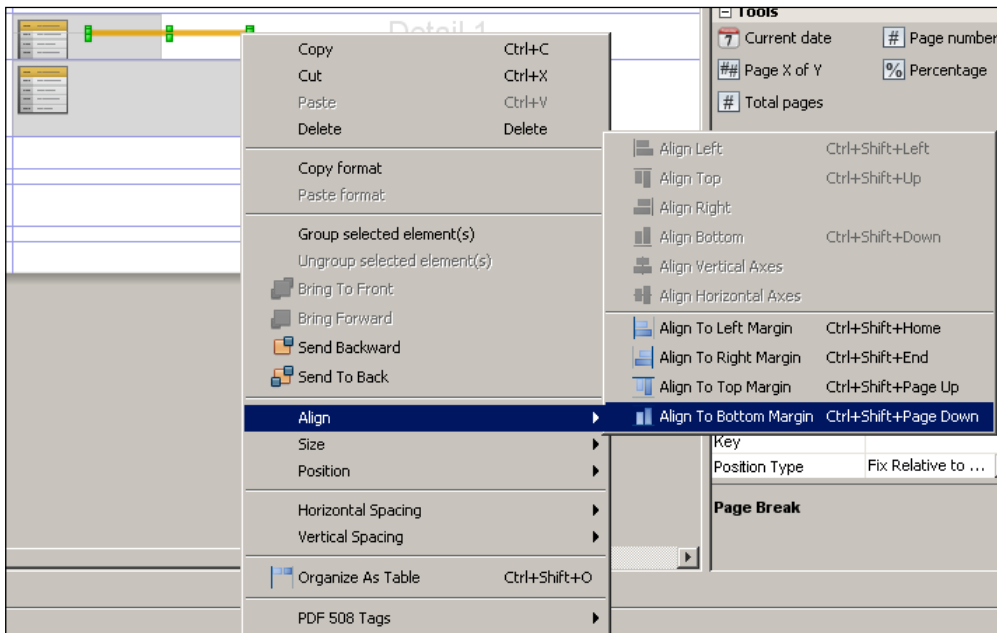
Goto Next Page to see Invoices of Alice

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1007	Alice	JasperReport Cookbook	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1038	Alice	JasperReport Cookbook	Mar09	49.5
1057	Alice	JasperReports for Java	Aug09	180.0

- Switch back to the **Designer** tab. Drag-and-drop a **Break** component from the **Palette** into the **Detail1** section of your report. A dialog will appear. Select the **Page break** option in the dialog and press the **OK** button, as shown below:



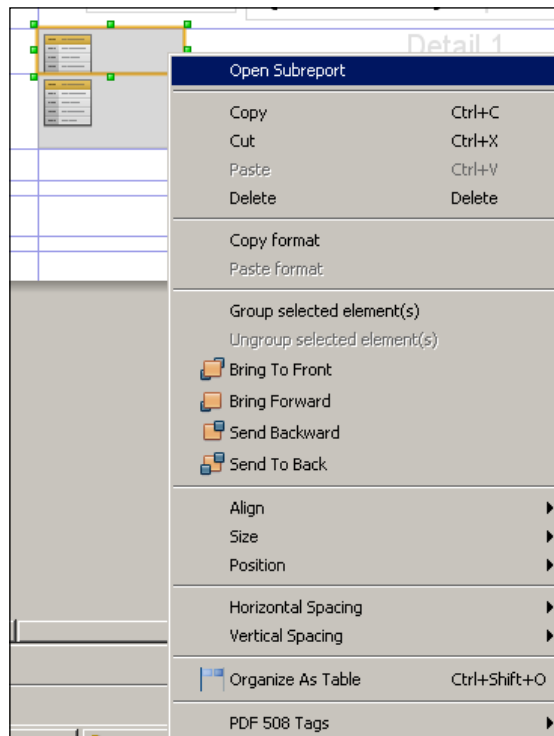
- Select the **Page break** component you just dropped and right-click on it. A pop-up menu will appear. Select **Align** from the pop-up menu, and a sub pop-up menu will appear. Select **Align To Bottom Margin** from the sub pop-up menu.



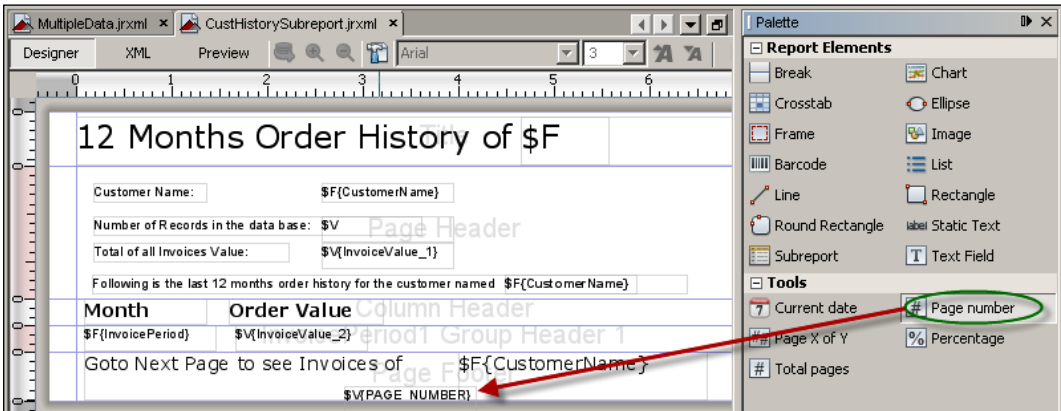
5. The **Page break** component will be positioned exactly below the **Subreport** component in the **Detail1** section.



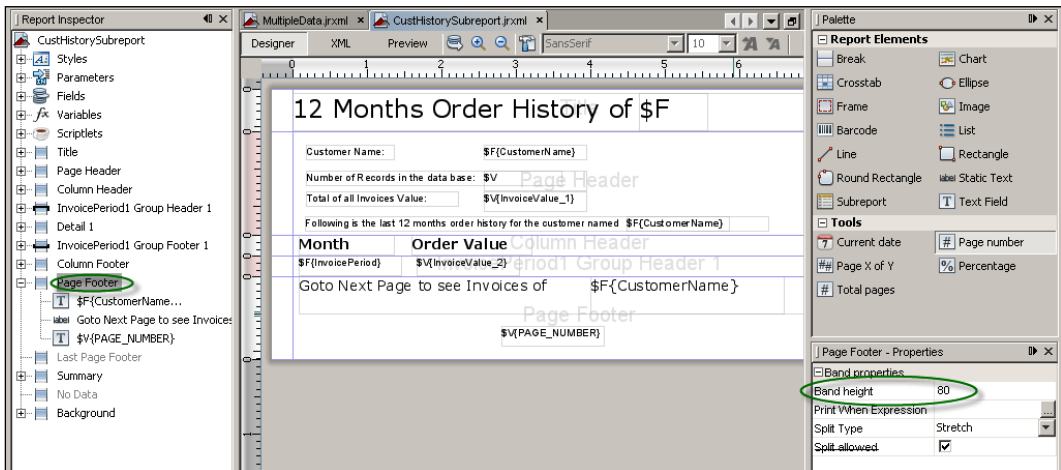
6. Select the **Subreport** component in the **Detail1** section and right-click on it. A pop-up menu will appear. Left-click on **Open Subreport** in the pop-up menu, as shown in the next screenshot:



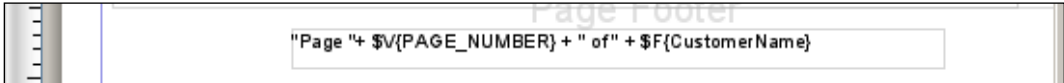
- A file named `CustHistorySubreport.jrxml` will open in a new tab. Drag-and-drop the **Page number** component from the **Palette** into the **Page Footer** section of the `CustHistorySubreport.jrxml` subreport. A **Text** field with the expression `#{PAGE_NUMBER}` will appear in the **Page Footer** section of the `CustHistorySubreport.jrxml` report.



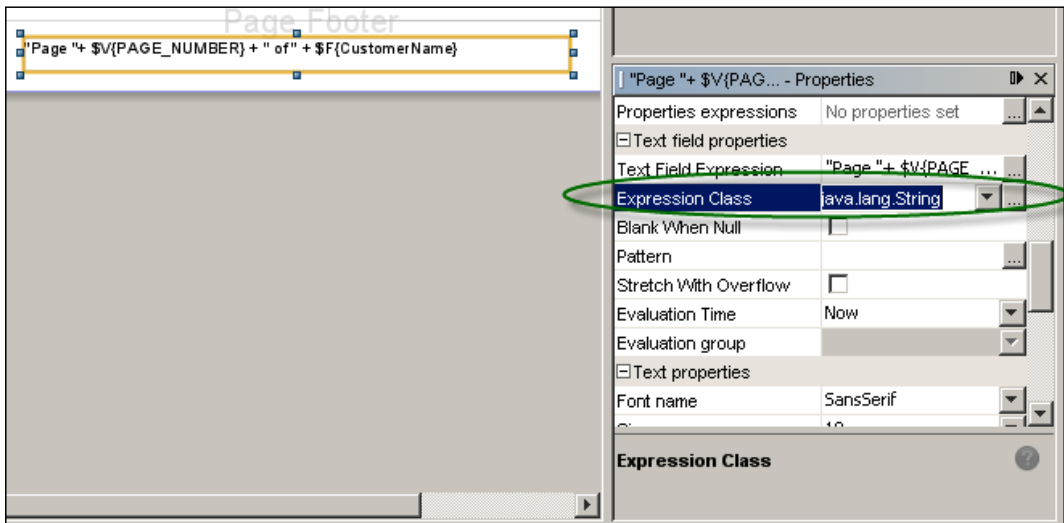
- In the **Report Inspector** select the **Page Footer** section and change the **Band height** property in the **Page Footer - Properties** section to 80. This will increase available space in the **Page Footer** section.



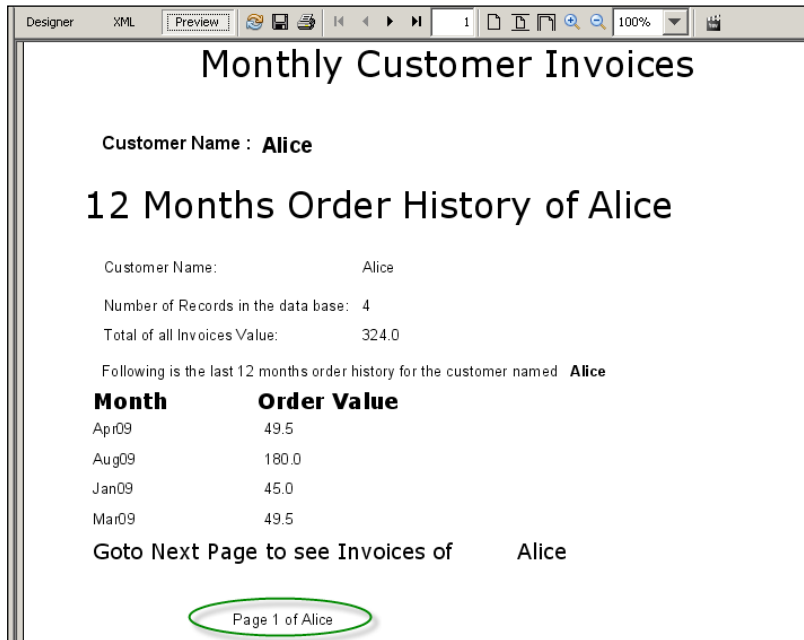
9. Increase the size of the **Text field** with expression `$V{PAGE_NUMBER}` in the **Page Footer** section by dragging its left edge to the left-side and the right edge to the right side. Double-click the **Text field** and its expression will be selected. Press the *Home* key on your keyboard and type "Page ". Then press the *End* key on your keyboard and type " of " + `$F{CustomerName}`. After typing, click anywhere outside the **Text field**. The expression will become "Page " + `$V{PAGE_NUMBER}` + of + `$F{CustomerName}`.



10. Select this **Text field** of the **Page Footer** section and change the **Expression Class** property in the **Properties** section to `java.lang.String`.

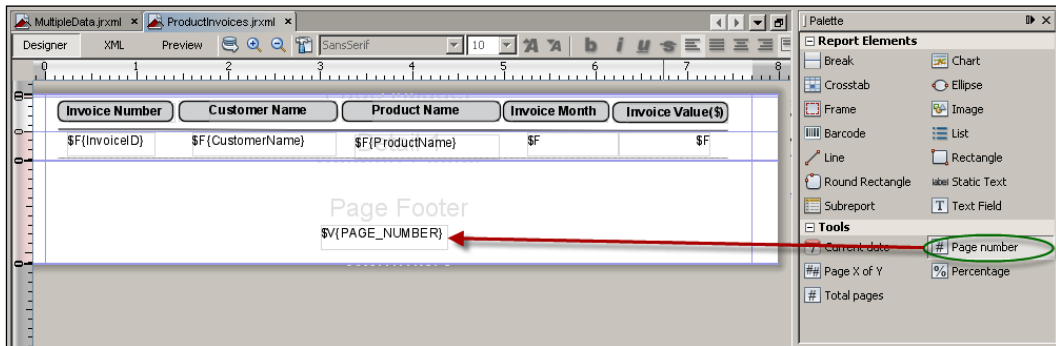


11. Save and close the `CustHistorySubreport.jrxml` subreport. Switch to the **Preview** tab of the master report, and you will see **Page1 of Alice** appearing at the bottom, as shown below. Check all the other pages as well. You will notice that only the first page of each customer is showing pagination; the subsequent pages don't carry any pagination.

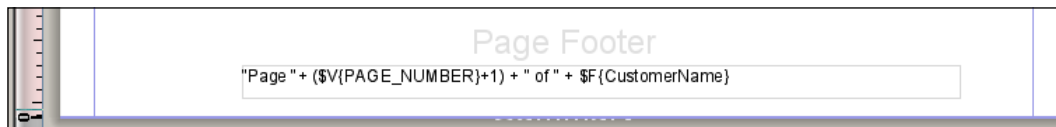


12. Switch back to the **Designer** tab. Select the **Subreport** component in the **Detail2** section and right-click on it. A pop-up menu will appear. Left-click on **Open Subreport** in the pop-up menu.
13. A file named `ProductInvoices.jrxml` will open in a new tab. In the **Report Inspector**, select the **Page Footer** section and change the **Band height** property in the **Page Footer-Properties** section to 80. This will increase available space in the **Page Footer** section.

14. Drag-and-drop the **Page number** component from the **Palette** into the **Page Footer** section of the `ProductInvoices.jrxml` report. A **Text Field** with the expression `$V{PAGE_NUMBER}` will appear in the **Page Footer** section of `ProductInvoices.jrxml` report, as shown below.

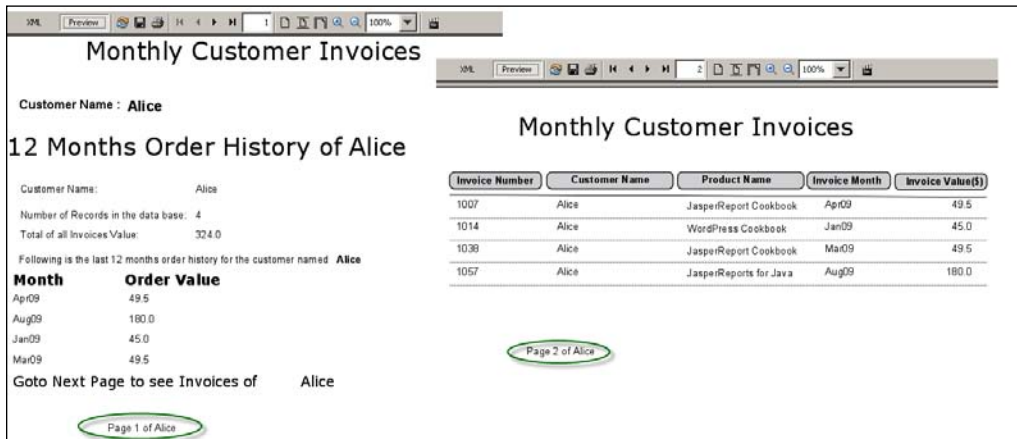


15. Increase the size of the **Text Field** with the expression `$V{PAGE_NUMBER}` in the **Page Footer** section by dragging its left edge to the left side and its right edge to the right side. Double-click the **Text Field** and the expression will be selected. Press the **Home** key on your keyboard and type `"Page " + (`. Then press the **End** key on your keyboard and type `+ 1) + " of " + $F{CustomerName}`. After typing, click anywhere outside the **Text Field**. The expression will become `"Page " + ($V{PAGE_NUMBER}+1) + " of " + $F{CustomerName}`, as shown below:

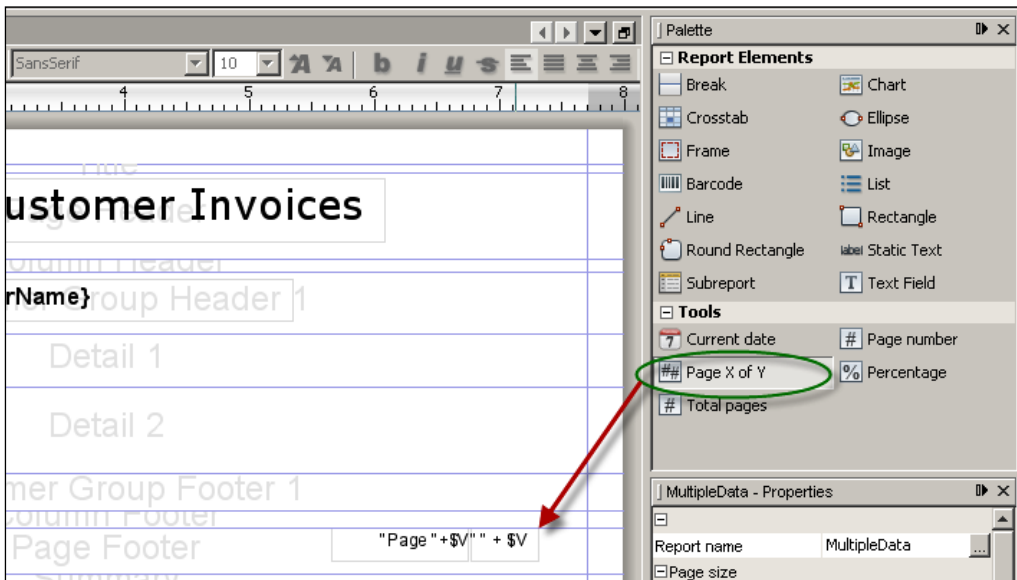


16. Select this **Text Field** that you just edited in the **Page Footer** section and change the **Expression Class** property from the **Properties** section to `java.lang.String`.

17. Save and close the `ProductInvoices.jrxml` subreport. Switch to the **Preview** tab of the master report, and you will see **Page 1 of Alice** is appearing at the bottom of page 1 and **Page 2 of Alice** is appearing at the bottom of page 2, as shown in the following screenshot. Check all the other pages as well. You will notice that each page of each customer is showing the correct pagination.



18. Switch back to the **Designer** tab. Drag-and-drop the **Page X of Y** component from the **Palette** into the **Page Footer** section of the master report. Two text fields with expressions `"Page " + $V{PAGE_NUMBER} + " of "` and `" " + $V{PAGE_NUMBER}` respectively will appear in the **Page Footer** section.



19. Switch to the **Preview** tab on the master report, and you will see that **Page 1 of 9** is appearing at the bottom of page 1 and **Page 2 of 9** is appearing at the bottom of page 2, and so on. Check all the other pages as well. You will notice that each page is now showing the correct pagination.

How it works...

In step 3 of the recipe you inserted a page break after `CustHistorySubreport.jrxml` subreport. Due to this page break, `CustHistorySubreport.jrxml` subreport, which is a single-page report, will display its data on a separate page. The `ProductInvoices.jrxml` subreport will fill multiple pages for each customer.

If each subreport contributes its natural pagination to the master report, the master will have Page 1 at two places (that is, the first page of each subreport). You corrected this problem by incrementing to the page numbers of the `ProductInvoices.jrxml` subreport in step 15 of this recipe by using the expression `($V{PAGE_NUMBER}+1)`.

6

Multi-column Reports

In this chapter, you will learn:

- ▶ Dividing the body of a report into multiple columns
- ▶ Displaying groups of data in separate columns
- ▶ Displaying data as name-value pairs in multiple columns
- ▶ Filling your report horizontally in multiple columns
- ▶ Using subreports to design a multi-column report

Introduction

You will learn to design multi-column reports in this chapter. The recipes cover the following aspects of multi-column reports:

- ▶ How to split your report into multiple columns
- ▶ How to present your data as name-value pairs in multiple columns
- ▶ How to fill your report horizontally in rows
- ▶ How to use subreports in a master report to design a multi-column report, especially useful when you are building a multi-column report whose columns completely differ from each other in size, data, and styles

Dividing the body of a report into multiple columns

Dividing a report body into multiple columns often gives a compact design with an improved look and feel.

JasperReports offers full flexibility of displaying your reports in multiple columns and provides a number of tools to enhance the look of multi-column reports.

In this simple recipe, you will learn how to convert your single-column report into a multi-column report.

Getting ready

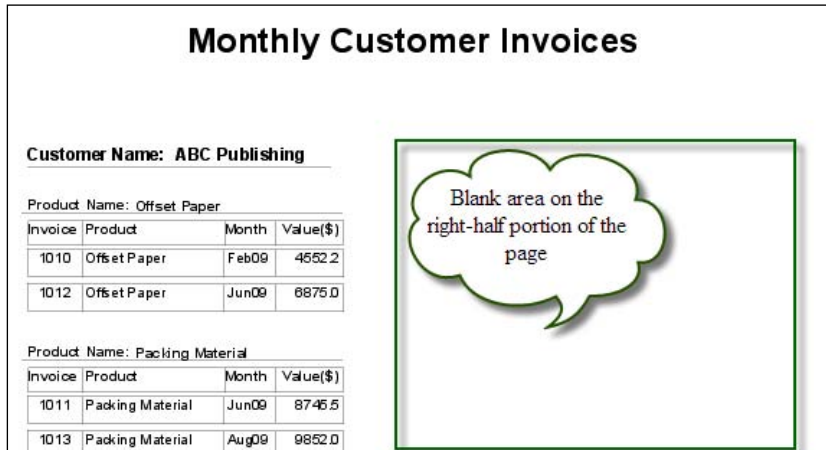
Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb7` and copy sample data for this recipe into the database.

How to do it...

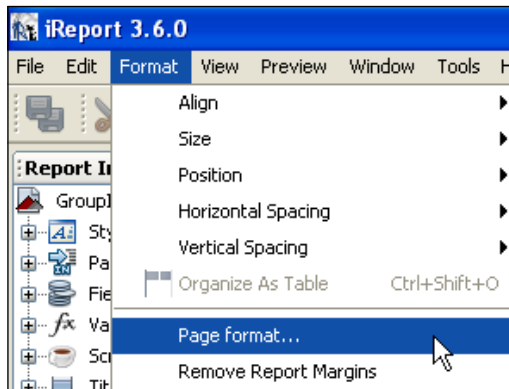
1. Open the `GroupInGroupMultiColumns.jrxml` file from the `Task1` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Customer Group Header 1**, **Product Group Header 1**, and **Detail 1** sections, as shown in the following screenshot:

Monthly Customer Invoices			
Page Header			
Customer Name: \${CustomerName}			
Product Name: \${ProductName}			
Invoice	Product	Month	Value(\$)
\$F	\${ProductName}	\$F	\$F

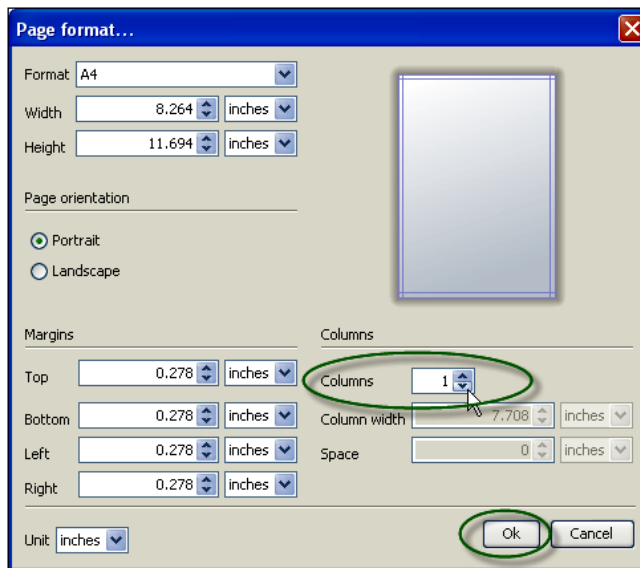
2. Switch to the **Preview** tab and you will see a report with data in tables spanning the left-half portion of the page. The right-half portion of the page is blank, as shown in the following screenshot:



3. Switch back to the **Designer** tab. In order to effectively populate the right-half portion of the page, you must use a two-column format for the report. In the main menu of **iReport**, click on the **Format** option. A pull-down menu will appear. Click on the **Page format...** option in the lower portion of the pull-down menu, as shown in the following screenshot:



4. A **Page format...** dialog will appear. You will notice that the value of the **Columns** textbox under the **Columns** section is **1**, as shown in the following screenshot. Set its value to **2** and press **Ok** button.



5. The **Page format...** dialog will disappear and a vertical bar will appear in your report. The bar will cut the **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, **Customer Group Footer 1**, **Product Group Footer 1**, and **Column Footer** sections vertically, thus splitting the report body into two halves. The vertical bar doesn't cut the **Title**, **Page Header**, **Page Footer**, and **Summary** sections, as shown in the following screenshot:

Monthly Customer Invoices			
Page Header			
Column Header			
Customer Name: \$F{CustomerName}			
Product Name: \$F{ProductName}			
Invoice	Product	Month	Value(\$)
\$F	\$F{ProductName}	\$F	\$F
Product Group Footer 1			
Customer Group Footer 1			
Column Footer			
Page Footer			

6. Switch to the **Preview** tab and you will see a report with data in two columns, as shown in the following screenshot:

Monthly Customer Invoices			
Customer Name: ABC Publishing			
Product Name: Offset Paper			
Invoice	Product	Month	Value(\$)
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	6875.0
Product Name: Packing Material			
Invoice	Product	Month	Value(\$)
1011	Packing Material	Jun09	8745.5
1013	Packing Material	Aug09	9852.0
Product Name: Printing Ink			
Invoice	Product	Month	Value(\$)
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
Customer Name: Alice			
Product Name: JasperReport			
Invoice	Product	Month	Value(\$)
1007	JasperReport	Apr09	49.5
1038	JasperReport	Mar09	49.5
Product Name: JasperReports for			
Invoice	Product	Month	Value(\$)
1057	JasperReports for	Aug09	180.0
Product Name: WordPress			
Invoice	Product	Month	Value(\$)
1014	WordPress	Jan09	45.0
Customer Name: Bob			
Product Name: Deployment of			
Invoice	Product	Month	Value(\$)
1058	Deployment of	Sep09	150.0
Product Name: Jasper Reports			
Invoice	Product	Month	Value(\$)
1037	JasperReports	Mar09	50.0
1006	JasperReports	Mar09	50.0
Product Name: JasperReports for			
Invoice	Product	Month	Value(\$)
1056	JasperReports for	Aug09	250.0
1059	JasperReports for	Sep09	190.0
Product Name: Printing Ink			
Invoice	Product	Month	Value(\$)
1018	Printing Ink	Sep09	150.0
Product Name: WordPress			
Invoice	Product	Month	Value(\$)
1015	WordPress	May09	45.0

How it works...

Notice the vertical bar you saw in step 5 of the recipe. The bar has no significance in report designing. This means you can place components on each side of the bar or even right on the bar.

However, the bar is actually a caution sign for you while designing multi-column reports. Notice that you placed all components on the left side of the bar. If you place components on the right of the bar, it will generate an ambiguous display. There will be two major ambiguities in the display:

- ▶ Data of the first column (that you put on the right of the bar) will overlap with the data of the second column.
- ▶ JasperReports will have no place to display data of the second column (that you put on the right of the bar). So it will just miss that particular bit of data and will not show it anywhere.

You can try dropping something on the right of the bar to see this ambiguity. If you take care and don't place anything on the right side of the bar, JasperReports will handle your multi-column report perfectly even if it spans multiple pages.

Displaying groups of data in separate columns

While working with multi-column reports, sometimes you need to display multiple groups or chunks of data in separate columns.

JasperReports provides full flexibility of organizing your data in multiple columns. It also provides a number of tools to introduce a column break into your report according to your needs. The requirement is sometimes static, which can be met by using a page break component, while sometimes it has to be decided at runtime.

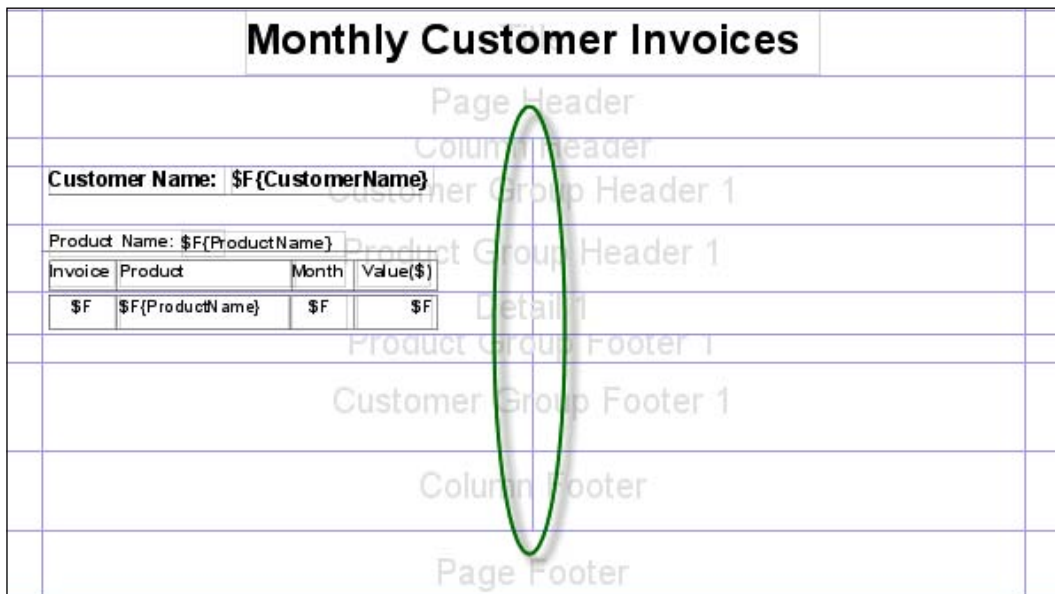
In this recipe, the JasperReports engine will dynamically decide the location of a column break. The report engine will look for the change of a customer name and as it finds the change, it inserts a column break, thus separating data of one customer from another's.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb7` and copy sample data for this recipe into the database.

How to do it...

1. Open the `GroupInGroupMultiColumns.jrxml` file from the `Task2` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Customer Group Header 1**, **Product Group Header 1**, and **Detail 1** sections. A vertical bar is also appearing in the report, which is dividing the **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, **Customer Group Footer 1**, **Product Group Footer 1**, and **Column Footer** sections into two equal halves, as shown in the following screenshot:

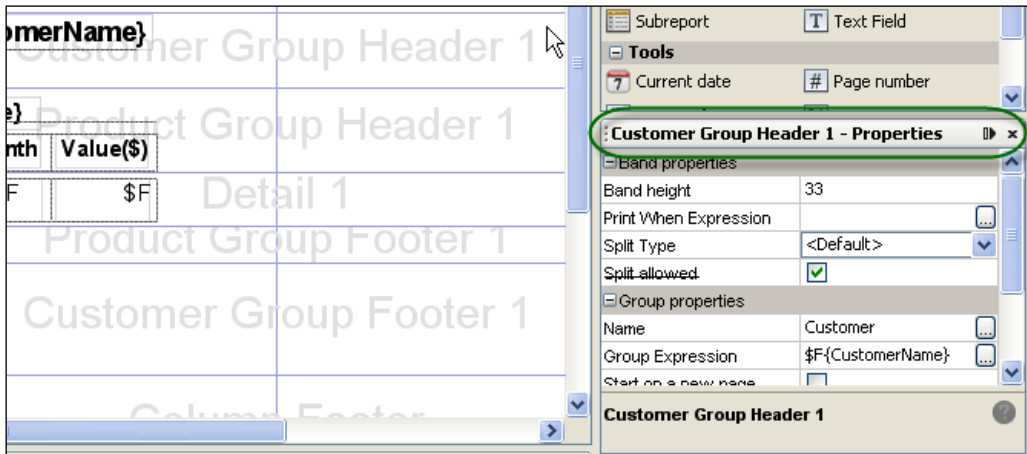


Monthly Customer Invoices			
Customer Name: \${CustomerName}			
Product Name: \${ProductName}			
Invoice	Product	Month	Value(\$)
\$F	\${ProductName}	\$F	\$F

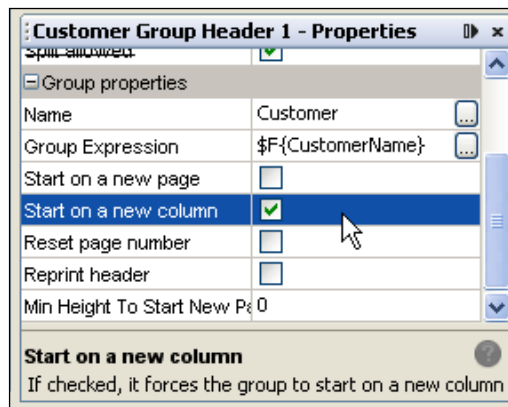
2. Switch to the **Preview** tab and you will see a report with data in two columns. Scroll down to see the complete page. You will notice that each column is displaying data of multiple customers, as shown in the following screenshot:

Monthly Customer Invoices			
Customer Name: ABC Publishing			
Product Name: Offset Paper			
Invoice	Product	Month	Value(\$)
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	6875.0
Product Name: Packing Material			
Invoice	Product	Month	Value(\$)
1011	Packing Material	Jun09	8745.5
1013	Packing Material	Aug09	9852.0
Product Name: Printing Ink			
Invoice	Product	Month	Value(\$)
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
Customer Name: Alice			
Product Name: JasperReport			
Invoice	Product	Month	Value(\$)
1007	JasperReport	Apr09	49.5
1038	JasperReport	Mar09	49.5
Product Name: JasperReports for			
Invoice	Product	Month	Value(\$)
1057	JasperReports for	Aug09	180.0
Product Name: WordPress			
Invoice	Product	Month	Value(\$)
1014	WordPress	Jan09	45.0
Customer Name: Bob			
Product Name: Deployment of			
Invoice	Product	Month	Value(\$)
1058	Deployment of	Sep09	150.0
Product Name: Jasper Reports			
Invoice	Product	Month	Value(\$)
1037	JasperReports	Mar09	50.0
1006	JasperReports	Mar09	50.0
Product Name: JasperReports for			
Invoice	Product	Month	Value(\$)
1056	JasperReports for	Aug09	250.0
1059	JasperReports for	Sep09	190.0
Product Name: Printing Ink			
Invoice	Product	Month	Value(\$)
1018	Printing Ink	Sep09	150.0
Product Name: WordPress			
Invoice	Product	Month	Value(\$)
1015	WordPress	May09	45.0

- Switch back to the **Designer** tab. Now you will learn how to show the data of each customer in a separate column. Click anywhere on white space inside the **Customer Group Header 1** section. **Customer Group Header 1 - Properties** will appear at the top of the **Properties** window below the **Palette**, as shown in the following screenshot:



- Scroll down the **Customer Group Header 1 - Properties** window to find the **Start on a new column** property. Tick this property's checkbox to change its value to true, as shown in the following screenshot:



- Switch to the **Preview** tab and you will see a report with data in two columns. Each column is showing data of a specific customer, as shown below. When the data of a customer ends, JasperReports dynamically inserts a column break and displays data of the next customer in a separate column. Even if the data of a customer spans multiple columns or multiple pages, JasperReports inserts a column break only when the data of a customer ends.

Monthly Customer Invoices			
Customer Name: ABC Publishing		Customer Name: Alice	
Product Name: Offset Paper		Product Name: JasperReport	
Invoice	Product	Month	Value(\$)
1010	Offset Paper	Feb09	4552.2
1060	Offset Paper	Aug09	8750.0
1012	Offset Paper	Jun09	6875.0
Product Name: Packing Material		Product Name: JasperReports for	
Invoice	Product	Month	Value(\$)
1013	Packing Material	Aug09	9852.0
1011	Packing Material	Jun09	8745.5
Product Name: Printing Ink		Product Name: WordPress	
Invoice	Product	Month	Value(\$)
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
		Product Name: JasperReports for	
Invoice	Product	Month	Value(\$)
1007	JasperReport	Apr09	49.5
1038	JasperReport	Mar09	49.5
		Product Name: JasperReports for	
Invoice	Product	Month	Value(\$)
1057	JasperReports for	Aug09	180.0
		Product Name: WordPress	
Invoice	Product	Month	Value(\$)
1014	WordPress	Jan09	45.0

Displaying data as name-value pairs in multiple columns

The normal tabular form of displaying data may not always be the most suitable presentation in a report. There are other options available. For example, you can display each record in a box, with all fields as name-value pairs. Such boxes of name-value pairs fit well in a multi-column report.

This recipe shows you how to display data as boxed name-value pairs in multiple columns.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb7` and copy sample data for this recipe into the database.

How to do it...

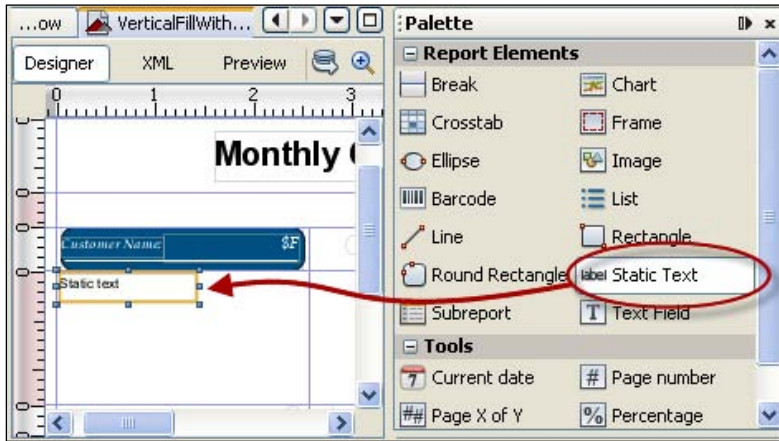
1. Open the `VerticalFillWithBox.jrxml` file from the `Task3` folder of the source code of this chapter. The **Designer** tab of iReport shows a three-column report containing data in the **Title** and **Column Header** sections, as shown below:

<div>Monthly Customer Invoices</div>		
Page Header		
Customer Name	\$F	Column Header

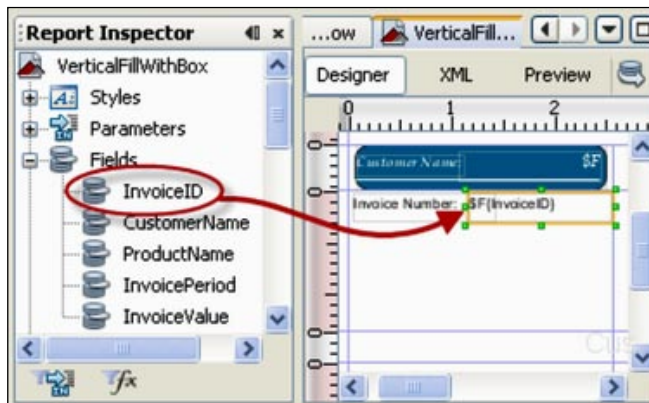
2. Switch to the **Preview** tab and you will see a report with a title, a header, and an empty body, as shown below:

<div>Monthly Customer Invoices</div>		
CustomerName: ABC Publishing	CustomerName: Alice	CustomerName: Bob

- Switch back to the **Designer** tab. Drag-and-drop a **Static Text** component from the **Palette** into the **Detail 1** section of your report, as shown below:



- Double-click on the **Static Text** component in the **Detail 1** section and type **Invoice Number:** as its value. While the **Static Text** component is selected, you will see its properties appear in the **Properties** window below the **Palette**. Set 70 as the value of the **Width** property of the **Static Text** component.
- Double-click the **Fields** node in the **Report Inspector** window on the left side of your report. The **Fields** node will expand to show **InvoiceID**, **CustomerName**, **ProductName**, **InvoicePeriod**, and **InvoiceValue** fields. Drag-and-drop a field named **InvoiceID** from the **Fields** node into the **Detail 1** section on the right exactly beside the **Invoice ID:** static text component, as shown below:



6. Switch to the **Preview** tab and you will see that the report shows invoice numbers under customer names, as shown below:

The screenshot shows a report titled "Monthly Customer Invoices". It displays a grid of customer names and invoice numbers. The first row shows "Customer Name: ABC Publishing" with "Invoice Number: 1010". The second row shows "Customer Name: Alice" with "Invoice Number: 1007". The third row shows "Customer Name: Bob" with "Invoice Number: 1006". The fourth row shows "Invoice Number: 1012", "Invoice Number: 1038", and "Invoice Number: 1056".

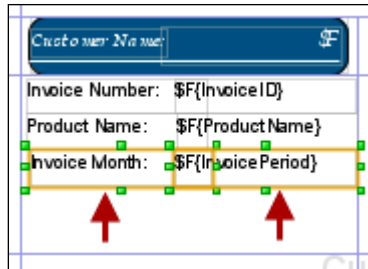
7. Drag-and-drop another **Static Text** component from the **Palette** into the **Detail 1** section just touching the bottom of the Invoice Number : static text component, as shown below. Double-click on it and type `Product Name :` as its value. Also set the **Width** property of the **Static Text** component to 70.

The screenshot shows the report design view. A blue header bar contains "Customer Name" and a currency symbol. Below it, the text "Invoice Number: \${InvoiceID}" is visible. A new static text component, "Product Name:", has been added below the invoice number, highlighted with a yellow border and a red arrow pointing to it.

8. Double-click the **Fields** node in the **Report Inspector** window. Drag-and-drop another field named `ProductName` from the **Fields** node into the **Detail 1** section right beside the `Product Name :` static text component, as shown below.

The screenshot shows the report design view. The static text component "Product Name:" now has a data field "\${ProductName}" added next to it, highlighted with a yellow border and a red arrow pointing to it.

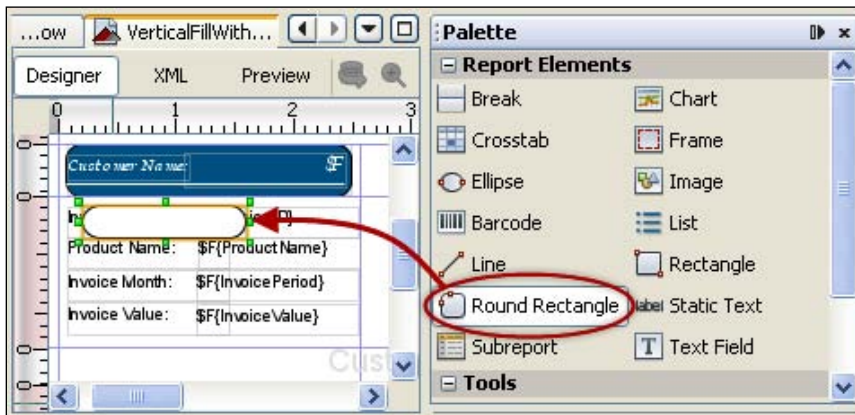
9. Repeat step 7 to drag-and-drop another **Static Text** component, double-click on it, and type `Invoice Month:` as its value. Also set the **Width** property of the **Static Text** component to 70.
10. Similarly, repeat step 8 to drag-and-drop the `InvoicePeriod` field from the **Fields** node in the **Report Inspector** window into the **Detail 1** section on the right exactly beside the `Invoice Month:` static text component, as shown below.



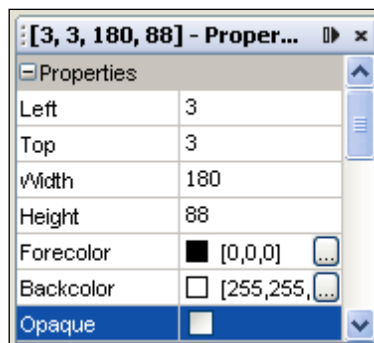
11. Repeat steps 7 and 8 for the fourth pair (that is, Invoice Value) of the **Static Text** and **Text Field** components. Double-click the **static text** component of this pair and type `Invoice Value:` as its value. Also set the **Width** property of the **Static Text** component to 70.
12. Switch to the **Preview** tab and you will see that your report now shows complete invoice records as name-value pairs.

Monthly Customer Invoices		
CustomerName: ABC Publishing	CustomerName: Alice	CustomerName: Bob
Invoice Number: 1010	Invoice Number: 1007	Invoice Number: 1006
Product Name: Offset Paper	Product Name: JasperReport	Product Name: JasperReports
Invoice Month: Feb09	Invoice Month: Apr09	Invoice Month: Mar09
Invoice Value: 4552.2	Invoice Value: 49.5	Invoice Value: 50.0

13. Switch back to the **Designer** tab. Drag-and-drop a **Round Rectangle** component from the **Palette** into the **Detail 1** section, as shown below. You will see that the **Round Rectangle** component is opaque, so it hides the components below it.



14. While **Round Rectangle** component is selected, its properties appear in the **Properties** window below the **Palette**. Set **3**, **3**, **180**, and **88** as values of the **Left**, **Top**, **Width**, and **Height** properties, respectively.
15. Select the **Opaque** property and uncheck the box beside it, as shown in the following screenshot. The components below the **Round Rectangle** will now become visible.



16. Switch to the **Preview** tab. You will see that each invoice record now has a border, as shown in the following screenshot:

Monthly Customer Invoices		
Customer Name: ABC Publishing	Customer Name: Alice	Customer Name: Bob
Invoice Number: 1010 Product Name: Offset Paper Invoice Month: Feb09 Invoice Value: 4552.2	Invoice Number: 1007 Product Name: JasperReport Invoice Month: Apr09 Invoice Value: 49.5	Invoice Number: 1006 Product Name: JapserReports Invoice Month: Mar09 Invoice Value: 50.0
Invoice Number: 1012 Product Name: Offset Paper Invoice Month: Jun09 Invoice Value: 6875.0	Invoice Number: 1038 Product Name: JasperReport Invoice Month: Mar09 Invoice Value: 49.5	Invoice Number: 1056 Product Name: JasperReports for Invoice Month: Aug09 Invoice Value: 250.0

Filling your report horizontally in multiple columns

You will sometimes require your reports to fill records horizontally in multiple columns instead of filling vertically downwards. This recipe gives you an example of when horizontal filling of report gives a better view and how to achieve this.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb7` and copy sample data for this recipe into the database.

How to do it...

1. Open the `ThreeColumnHorizontalFill.jrxml` file from the `Task4` folder of the source code of this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, **Customer Group Footer 1**, and **Page Footer** sections, as shown in the following screenshot:

Monthly Customer Invoices	
Three-Column Horizontal Fill Report	
Customer Name: <code>\$F{CustomerName}</code>	
\$F{ProductName}	Product Group Header 1
Invoice Number: <code>\$F{InvoiceID}</code> Product Name: <code>\$F{Product Name}</code> Invoice Month: <code>\$F{InvoicePeriod}</code> Invoice Value: <code>\$F{InvoiceValue}</code>	Detail 1
Invoice Total: <code>\$V{ColumnSum}</code>	
Page Footer	
<code>"Page "+\$V" " + \$V</code>	

2. Switch to the **Preview** tab, and you will see that the left side of the report contains customer invoices grouped by customer names. Approximately two-third of the page is empty, as shown in the following screenshot:

Monthly Customer Invoices

Three-Column Horizontal Fill Report

Customer Name: ABC Publishing

Offset Paper

Invoice Number: 1010
Product Name: Offset Paper
Invoice Month: Feb09
Invoice Value: 4952.2

Empty Area

Invoice Number: 1012
Product Name: Offset Paper
Invoice Month: Jun09
Invoice Value: 8875.0

Empty Area

Packing Material

Invoice Number: 1011
Product Name: Packing Material
Invoice Month: Jun09
Invoice Value: 8745.5

Empty Area

Invoice Number: 1013
Product Name: Packing Material
Invoice Month: Aug09
Invoice Value: 9852.0

Empty Area

Printing Ink

Invoice Number: 1018
Product Name: Printing Ink
Invoice Month: Jun09
Invoice Value: 2440.0

Empty Area

Invoice Number: 1020
Product Name: Printing Ink
Invoice Month: Mar09
Invoice Value: 3450.0

Empty Area

Invoice Total: null

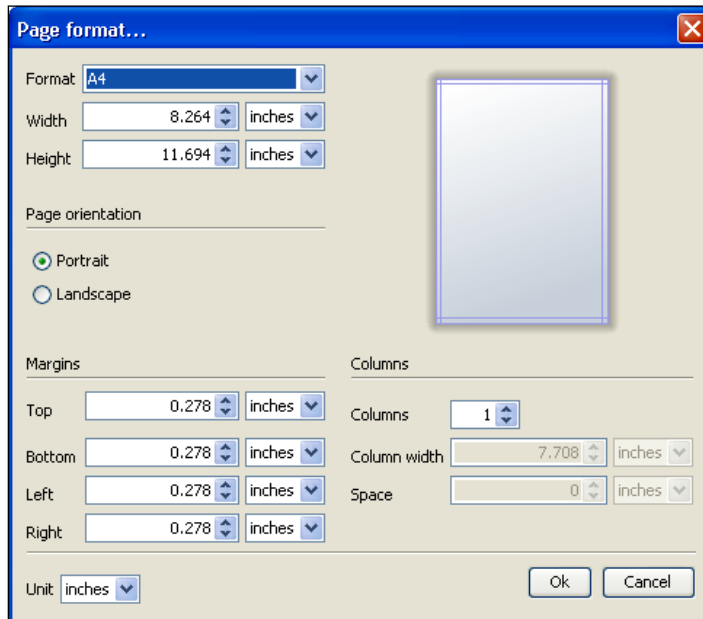
Customer Name: Alice

Page 1 of 9

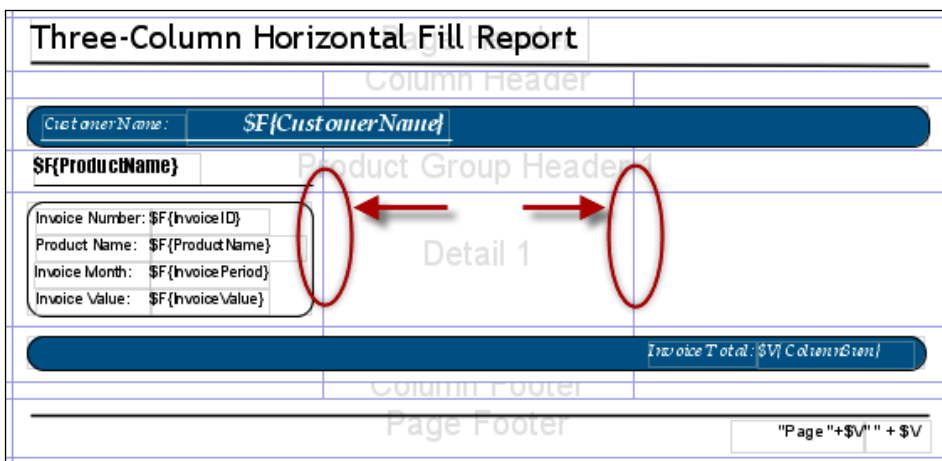
278

www.it-ebooks.info

- Switch back to the **Designer** tab. Click the **Format** menu and select the **Page format** option. A **Page format...** window will appear, as shown in the following screenshot:



- Click on the **Columns** field under the **Columns** section on the right of the **Page format...** window and set **3** as its value. Press the **Ok** button at the bottom to dismiss the window. You will see that now the report body is divided into three columns with two vertical blue lines starting from the **Page Header** section and ending at the **Page Footer** section, as shown encircled in the following screenshot:



- Switch to the **Preview** tab, and you will see that customer invoices are now shown in three columns. The left column is filled first, then the middle column, and finally the right column is filled with boxed invoice data. You will also notice that when JasperReports displays a group header at the beginning of a customer (or a group footer at the end of a customer) in the middle column, it does not care about records shown in the right column. And so the group header and footer overlap with boxed records, as shown encircled in the following screenshot:

Monthly Customer Invoices

Three-Column Horizontal Fill Report

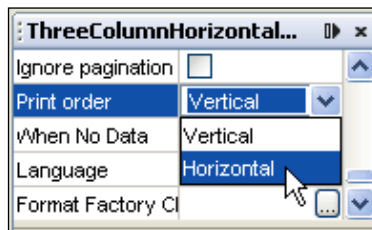
Customer Name: ABC Products Inc. Offset Paper Invoice Number: 1010 Product Name: Offset Paper Invoice Month: Feb09 Invoice Value: 4552.2 Invoice Number: 1012 Product Name: Offset Paper Invoice Month: Jun09 Invoice Value: 8875.0 Packing Material Invoice Number: 1011 Product Name: Packing Material Invoice Month: Jun09 Invoice Value: 8745.5 Invoice Number: 1013 Product Name: Packing Material Invoice Month: Aug09 Invoice Value: 9852.0 Printing Ink Invoice Number: 1016 Product Name: Printing Ink Invoice Month: Jun09 Invoice Value: 2440.0 Invoice Number: 1020 Product Name: Printing Ink Invoice Month: Mar09 Invoice Value: 3450.0	Invoice Number: 1007 Product Name: JasperReports Invoice Month: Mar09 Invoice Value: 50.0 Invoice Number: 1008 Product Name: JasperReports Invoice Month: Mar09 Invoice Value: 50.0 JasperReports for Invoice Number: 1056 Product Name: JasperReports for Java Invoice Month: Aug09 Invoice Value: 250.0 Invoice Number: 1059 Product Name: JasperReports for Java Invoice Month: Sep09 Invoice Value: 190.0 WordPress Invoice Number: 1014 Product Name: WordPress Cookbook Invoice Month: Jun09 Invoice Value: 45.0 Deployment of Invoice Number: 1028 Product Name: Deployment of Reports Invoice Month: Sep09 Invoice Value: 150.0 JasperReports	Invoice Number: 1017 Product Name: JasperReports Invoice Month: Mar09 Invoice Value: 50.0 Invoice Number: 1006 Product Name: JasperReports Invoice Month: Mar09 Invoice Value: 50.0 JasperReports for Invoice Number: 1056 Product Name: JasperReports for Java Invoice Month: Aug09 Invoice Value: 250.0 Invoice Number: 1059 Product Name: JasperReports for Java Invoice Month: Sep09 Invoice Value: 190.0 Printing Ink Invoice Number: 1016 Product Name: Printing Ink Invoice Month: Jun09 Invoice Value: 2440.0 WordPress Invoice Number: 1015 Product Name: WordPress Cookbook Invoice Month: May09 Invoice Value: 45.0
Customer Name: Alice		
Printing Material		

Page 1 of 4

- Switch back to the **Designer** tab. Click the name of your report in the **Report Inspector** window on the left side of your report, as shown in the following screenshot:



- Report properties will appear in the **Properties** window below the **Palette of Components** window. Click on the **Print order** property under the **More..** section in the **Properties** window. Select **Horizontal** as its value from the drop-down list beside it, as shown in the following screenshot:



8. Switch to the **Preview** tab and you will see this time that records are filled horizontally in rows instead of columns. The first row is filled, then the next row, and so on. You will also notice that while filling rows horizontally, JasperReports fixes the problem of data overlapping that you saw earlier in the screenshot of step 5.

Monthly Customer Invoices

Three-Column Horizontal Fill Report

Customer Name: **ABC Publishing**

Offset Paper

Invoice Number: 1010 Product Name: Offset Paper Invoice Month: Feb09 Invoice Value: 4552.2	Invoice Number: 1012 Product Name: Offset Paper Invoice Month: Jun09 Invoice Value: 6875.0
---	---

Packing Material

Invoice Number: 1011 Product Name: Packing Material Invoice Month: Jun09 Invoice Value: 8745.5	Invoice Number: 1013 Product Name: Packing Material Invoice Month: Aug09 Invoice Value: 9852.0
---	---

Printing Ink

Invoice Number: 1016 Product Name: Printing Ink Invoice Month: Jun09 Invoice Value: 2440.0	Invoice Number: 1020 Product Name: Printing Ink Invoice Month: Mar09 Invoice Value: 3450.0
---	---

Invoice Total: 278.00

Customer Name: **Alice**

JasperReport

Invoice Number: 1007 Product Name: JasperReport Invoice Month: Apr09 Invoice Value: 49.5	Invoice Number: 1038 Product Name: JasperReport Invoice Month: Mar09 Invoice Value: 49.5
---	---

JasperReports for

Invoice Number: 1057 Product Name: JasperReports for Java Invoice Month: Aug09 Invoice Value: 180.0
--

WordPress

 Page 1 of 5

How it works...

JasperReports fills data vertically downwards by default. In this recipe, you worked with a report that contained a header and footer designed for each customer. If you look at the customer group header in the designer view of step 1, you will see that it spans all across the page from left to right. This created the data overlapping problem highlighted in step 5, when you divided the report into three columns.

Later, when you horizontally filled the report in step 7, the data overlapping problem was automatically fixed. This shows that group headers and footers spanning all across the page from left to right are best suited to reports that fill data horizontally.

Using subreports to design a multi-column report

You can place subreports in your master report in such a way that the look and feel of the master report resembles that of a multi-column report. Therefore, you can use subreports to design multi-column reports.

However, doing this may cause minor problems in certain scenarios. This recipe demonstrates how to use subreports in multi-column reports and resolve problems that occur while doing this.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb7` and copy sample data for this recipe into the database.

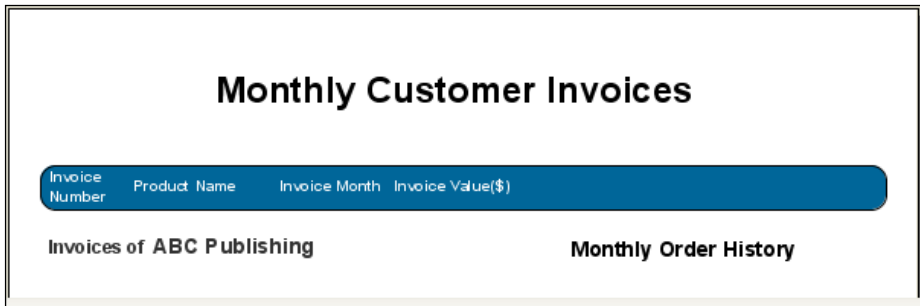
You will be using three JRXML files in this recipe: `SimulatingMultiColumnReport.jrxml`, `CustomerInvoicesSubreport.jrxml`, and `MonthlyOrderHistorySubreport.jrxml`. You will find these files in the `Task5` folder of the source code download of this chapter. The `SimulatingMultiColumnReport.jrxml` file is the master report, which uses the other two files as subreports. The master report has to refer to all its subreports using a complete path (you cannot use relative paths). This means you have to copy the three JRXML files to the `C:\JasperReportsCookBookSamples\` folder on your PC. I have hardcoded this complete path in the master report (`SimulatingMultiColumnReport.jrxml`).

How to do it...

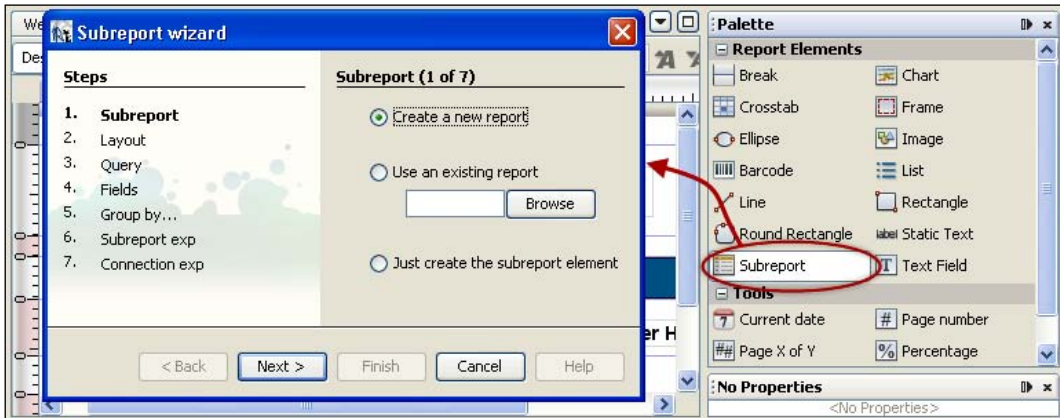
1. Open the `SimulatingMultiColumnReport.jrxml` file from the `C:\JasperReportsCookBookSamples` folder on your PC. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, and **Customer Group Header 1** sections, as shown in the following screenshot:



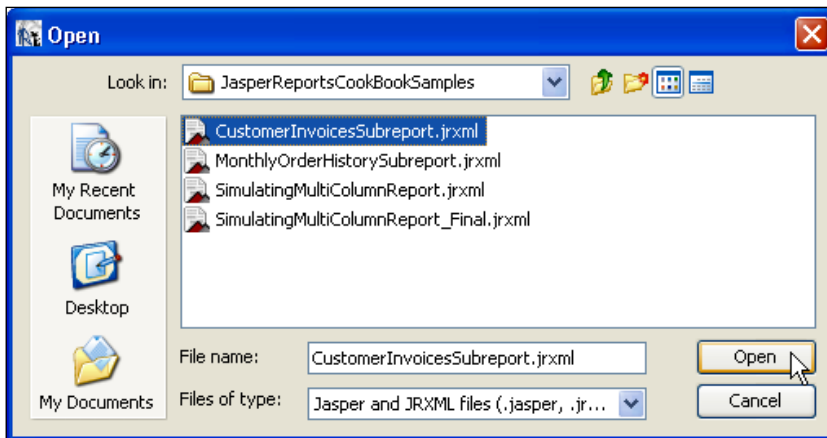
2. Switch to the **Preview** tab and you will see a report displaying a title, a column header, as well as customer names but no invoice data, as shown in the following screenshot:



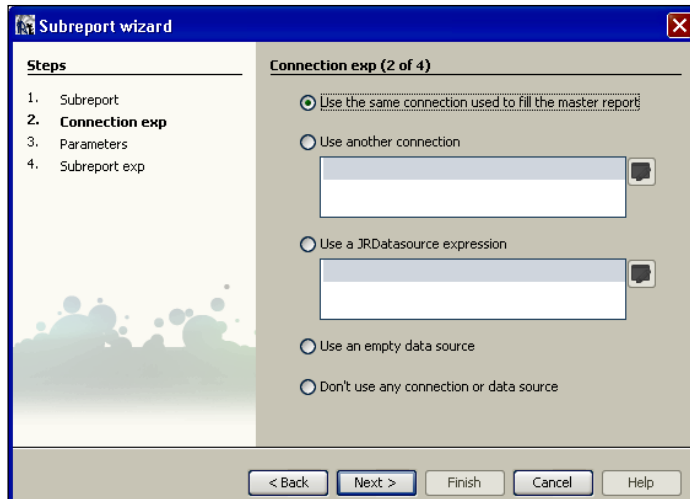
- Switch back to the **Designer** tab. Drag-and-drop a **Subreport** component from the **Palette** into the **Detail 1** section just below the static text component labeled **Invoice of**. A **Subreport wizard** dialog will appear, as shown in the following screenshot:



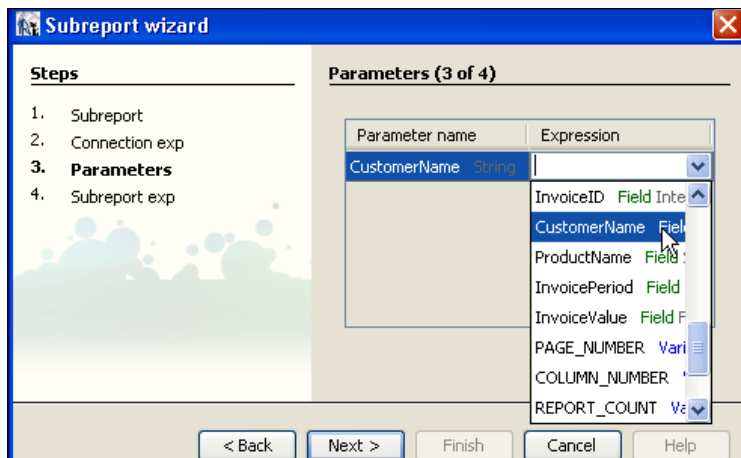
- Choose the **Use an existing report** option and press the **Browse** button. A browser dialog will appear. Browse to the `CustomerInvoicesSubreport.jrxml` file placed in the `C:\JasperReportsCookBookSamples` folder on your PC and press the **Open** button, as shown in the following screenshot. The browser dialog will disappear.



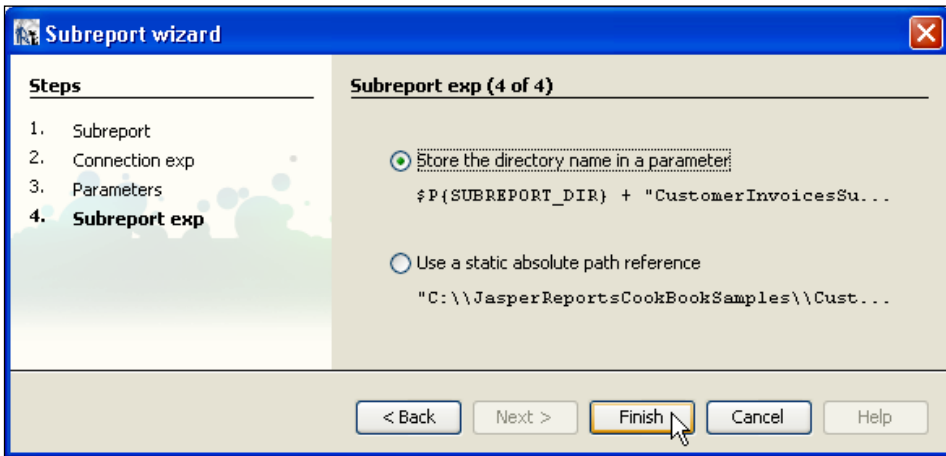
- Press the **Next** button in the **Subreport wizard** dialog. Notice that now the left side of the **Subreport wizard** shows four steps (**Subreport**, **Connection exp**, **Parameters**, **Subreport exp**). You are at step 2 (**Connection exp**), which is shown in bold. The **Connection exp (2 of 4)** dialog allows you to specify a database connection for your subreport. Press the **Next** button at the bottom to continue with the default **Use the same connection used to fill the master report** option, as shown in the following screenshot:



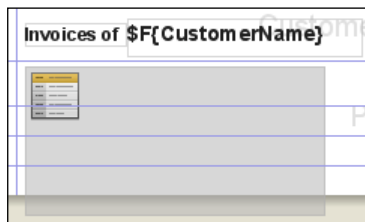
- A **Parameters (3 of 4)** dialog will appear. This shows the `CustomerName` parameter of `CustomerInvoicesSubreport.jrxml` and allows you to map it to elements (that is **Fields** or **Variables**) of the main `SimulatingMultiColumnReport.jrxml` report. Click the **Expression** field beside the `CustomerName` parameter. A drop-down list will open, which shows fields and variables of the master report. Select the `CustomerName` field from the list, as shown in the following screenshot. Press the **Next** button.



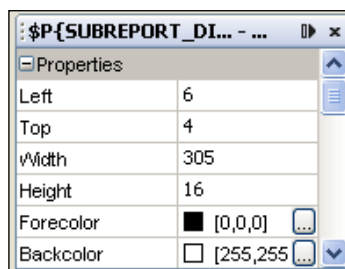
7. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected by default and press the **Finish** button.



8. The **Subreport** component will be placed in the **Detail 1** section, expanding to other sections as well, as shown in the following screenshot:



9. Click on the **Subreport** component. Its properties will appear in the **Properties** window below the **Palette**. Find the **Width** property and set **305** as its value. Similarly, set **16** as value of its **Height** property, as the shown in the following screenshot:



10. The size of the **Subreport** component will now fit inside the **Detail 1** section.



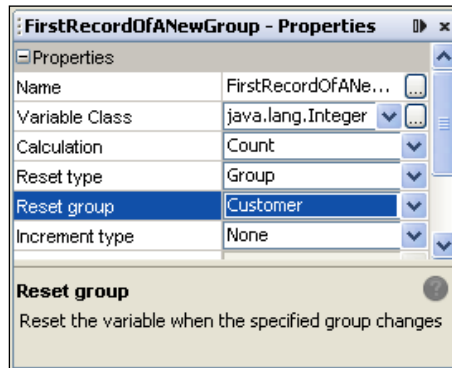
11. Switch to the **Preview** tab. You will see that the subreport (that is, `CustomerInvoicesSubreport.jrxml`) repeats for each record in the **Detail 1** section of the master report (that is, `SimulatingMultiColumnReport.jrxml`).

Monthly Customer Invoices			
Invoice Number	Product Name	Invoice Month	Invoice Value(\$)
Invoices of ABC Publishing			
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	6875.0
1011	Packing Material	Jun09	8745.5
1013	Packing Material	Aug09	9852.0
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
Monthly Order History			
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	6875.0
1011	Packing Material	Jun09	8745.5

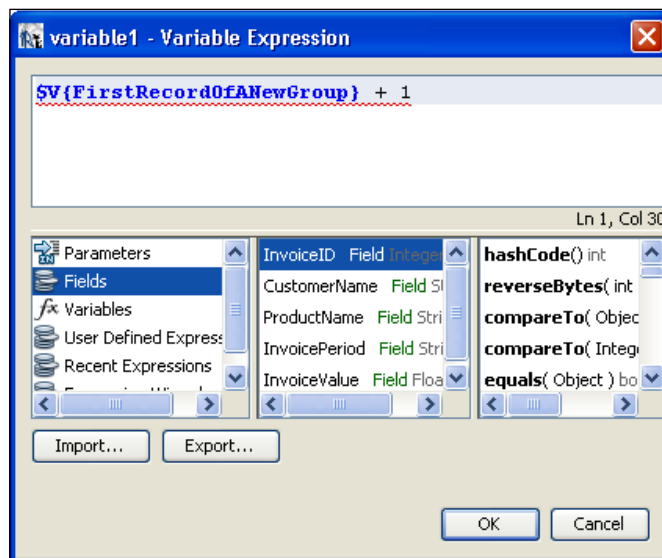
Customer invoices data
being report with each
record in Detail 1 section

12. Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Variable** option.
13. A new variable named `variable1` will be added at the end of the variables list.
14. While `variable1` is selected, find the **Name** property in the **Properties** window below the **Palette** and change its value to `FirstRecordOfANewGroup`. Now the name of the `variable1` variable will change to `FirstRecordOfANewGroup`.
15. Select the **Variable Class** property and change its value to `java.lang.Integer`.
16. Select the **Calculation** property and change its value to `Count`.
17. Select the **Reset type** property and change its value to `Group`.

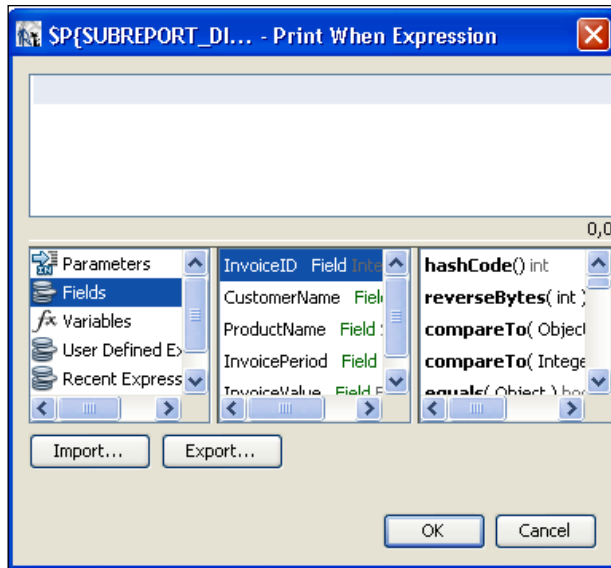
18. Select the **Reset group** property and change its value to `Customer`, as shown in the following screenshot:



19. Select the **Variable Expression** property and click on the button beside it. A **Variable Expression** window with no default expression will open. Type `$V{FirstRecordOfANewGroup} + 1` in the expression editor window, as shown in the following screenshot. Press the **OK** button.



20. Select the **Subreport** component in the **Detail 1** section. Its properties will appear in the **Properties** window below the **Palette**. Find the **Remove Line When Blank** property and check the box beside it.
21. Look for the **Print When Expression** property and click on the button beside it. A **Print When Expression** window with no default expression will open, as shown in the following screenshot:



22. Type the `$V{FirstRecordOfANewGroup} == 0` as value in the **Print When Expression** window, as shown in the following screenshot. Press the **OK** button to dismiss the window.

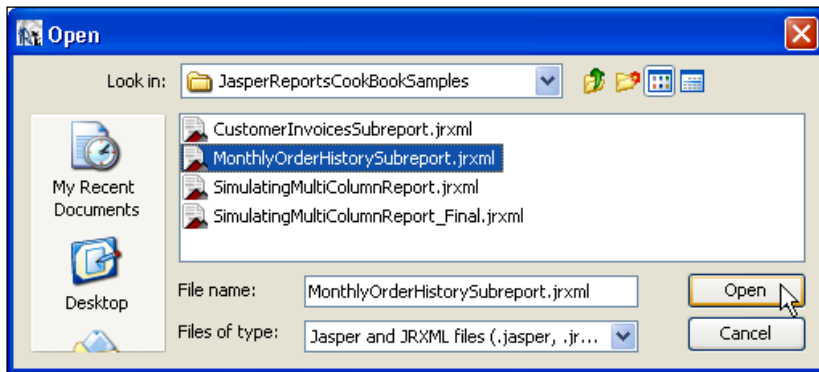


23. Switch to the **Preview** tab. This time you will see that the subreport is shown only once for each customer and does not repeat itself for every record.

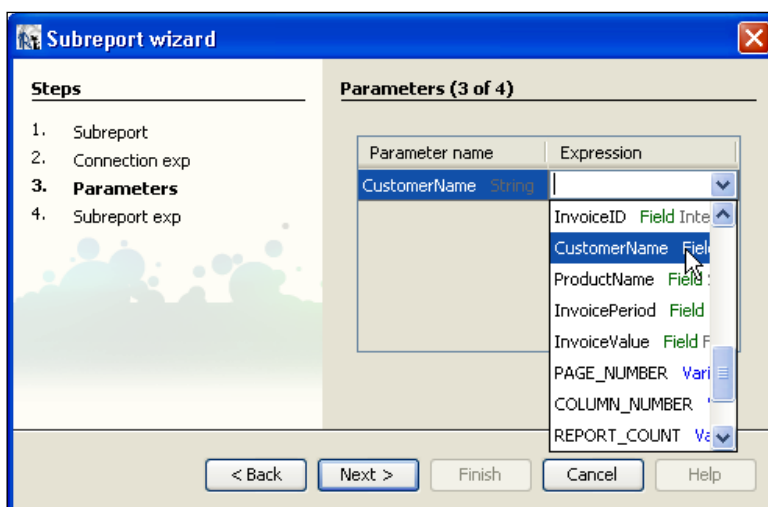
Monthly Customer Invoices			
Invoice Number	Product Name	Invoice Month	Invoice Value(\$)
Invoices of ABC Publishing		Monthly Order History	
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	6875.0
1011	Packing Material	Jun09	8745.5
1013	Packing Material	Aug09	9852.0
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
Invoices of Alice		Monthly Order History	
1007	JasperReport	Apr09	49.5
1038	JasperReport	Mar09	49.5
1057	JasperReports for	Aug09	180.0
1014	Word Press Cookbook	Jan09	45.0

24. Switch back to the **Designer** tab. Drag-and-drop another **Subreport** component from the **Palette** into the **Detail 1** section just to the right of the first subreport below the **Monthly Order History** static text component.

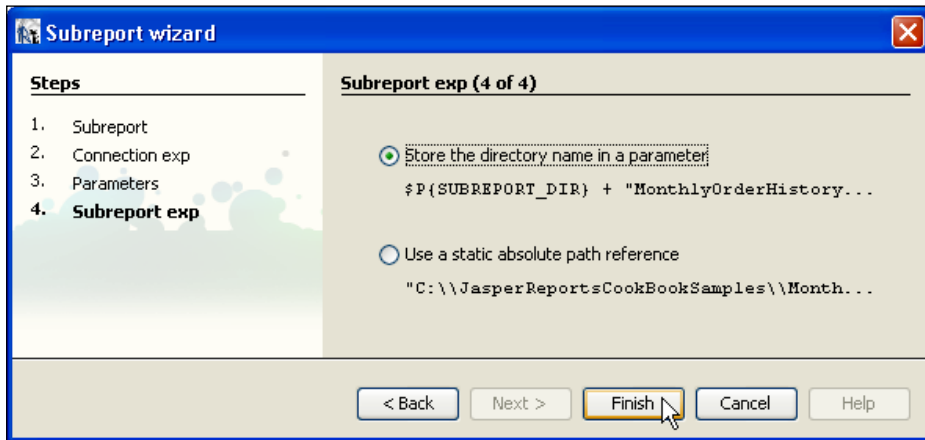
25. A **Subreport wizard** dialog will appear. Choose the **Use an existing report** option, press the **Browse** button to browse to the `MonthlyOrderHistorySubreport.jrxml` file located in the `C:\JasperReportsCookBookSamples\` folder on your PC, and press the **Open** button, as shown in the following screenshot:



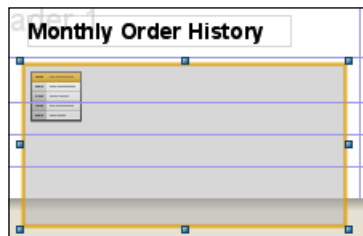
26. Press the **Next** button in the **Subreport wizard** dialog. A **Connection exp (2 of 4)** dialog will appear. Press the **Next** button at the bottom to continue with the default **Use the same connection used to fill the master report** option.
27. A **Parameters (3 of 4)** dialog will appear. This will show the `CustomerName` parameter of `MonthlyOrderHistorySubreport.jrxml` and allows you to map it to elements (that is, **Fields** or **Variables**) of the main `SimulatingMultiColumnReport.jrxml` report. Click on the **Expression** field beside the `CustomerName` parameter. A drop-down list will open, which shows the fields and variables of the master report. Select the **CustomerName** field from the list, as shown in the following screenshot. Click the **Next** button.



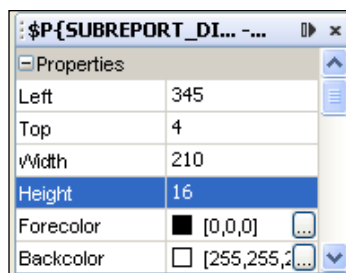
28. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected and click the **Finish** button.



29. The **Subreport** component will be placed in the **Detail 1** section, vertically expanding to other sections as well, as shown in the following screenshot:



30. Click on the **Subreport** component you just dropped. Its properties will appear in the **Properties** window below the **Palette**. Find the **Width** property and set **210** as its value. Similarly, set **16** as value of its **Height** property, as the shown in the following screenshot:



31. Select the **Remove Line When Blank** property and check the box against it.
32. Select the **Print When Expression** property and click on the button beside it. A **Print When Expression** window with no default expression will open.
33. Type `$V{FirstRecordOfANewGroup} == 0` as value in the **Print When Expression** window, as shown in the following screenshot. Press the **OK** button.



34. Switch to the **Preview** tab. You will see that the left part of the report body contains customer invoices, whereas the right part of the report contains monthly order history for customers, as shown in the following screenshot:

Monthly Customer Invoices

Invoice Number	Product Name	Invoice Month	Invoice Value(\$)
Invoices of ABC Publishing			
1010	Offset Paper	Feb09	4552.2
1012	Offset Paper	Jun09	8875.0
1011	Packing Material	Jun09	8745.5
1013	Packing Material	Aug09	9852.0
1016	Printing Ink	Jun09	2440.0
1020	Printing Ink	Mar09	3450.0
Monthly Order History			
Total number of invoices: 6			
Total of all Invoices Value: 35914.7			
Month		Order Value	
Aug09		9852.0	
Feb09		4552.2	
Jun09		18060.5	
Mar09		3450.0	

Invoice Number	Product Name	Invoice Month	Invoice Value(\$)
Invoices of Alice			
1007	JasperReport	Apr09	49.5
1038	JasperReport	Mar09	49.5
1057	JasperReports for	Aug09	180.0
1014	Word Press Cookbook	Jan09	45.0
Monthly Order History			
Total number of invoices: 4			
Total of all Invoices Value: 324.0			
Month		Order Value	
Apr09		49.5	
Aug09		180.0	
Jan09		45.0	
Mar09		49.5	

How it works...

In this recipe, you have used two subreports in a master report. You placed the two subreports in such a way that the master report looks like a multi-column report.

Notice from the recipe that when you inserted the first subreport in step 3 of the recipe, it started repeating itself for every record. This is a common problem that you may encounter when you insert a subreport in any detail section of a master report.

In order to stop the subreport from repeating itself, you configured a variable named `FirstRecordOfANewGroup` in steps 12 to 19. Then you configured the **Print When Expression** property of the two subreports in steps 22 and 33, which prints the subreports only on the first record of every customer, so that the subreports do not repeat for every record.

Also notice steps 20 and 31 of the recipe, where you configured the **Remove Line When Blank** property of the two subreports. This property removes the white space left empty when a subreport does not print due to the **Print When Expression** property.

You normally need to configure the **Remove Line When Blank** property whenever you wish to print a subreport conditionally. If you don't use this property, you will see a lot of unwanted white space in your report.

You can switch to the **XML** tab to see the following JRXML code:

```
<detail>
  <band height="28" splitType="Stretch">
    <subreport>
      <reportElement x="6" y="4" width="305" height="16"
        isRemoveLineWhenBlank="true">
        <printWhenExpression>
          <![CDATA[$V{FirstRecordOfANewGroup} == 0]]>
        </printWhenExpression>
      </reportElement>
      <subreportParameter name="CustomerName">
        <subreportParameterExpression>
          <![CDATA[$F{CustomerName}]]>
        </subreportParameterExpression>
      </subreportParameter>
      <connectionExpression>
        <![CDATA[$P{REPORT_CONNECTION}]]>
      </connectionExpression>
      <subreportExpression class="java.lang.String">
        <![CDATA[$P{SUBREPORT_DIR}
          + "CustomerInvoicesSubreport.jasper"]]>
      </subreportExpression>
    </subreport>
  </band>
</detail>
```

```
<subreport>
    <!--details of this subreport omitted for brevity-->
</subreport>
</band>
</detail>
```

In this recipe, you dragged-and-dropped two subreport components and so iReport authored two `<subreport>` tags as shown in the above JRXML code. In steps 3 to 10, iReport authored the first `<subreport>` tag, while in steps 24 to 30 it authored the second `<subreport>` tag.

The following is a mapping of recipe steps to JRXML code of the first `<subreport>` tag:

- ▶ In step 5 of the recipe, you told iReport to use the same database connection for the subreport as the master report. In response, iReport authored the `<connectionExpression>` tag shown highlighted.
- ▶ In step 6, you mapped a master report parameter to the subreport. This is when iReport authored the `<subreportParameter>` tag shown highlighted.
- ▶ Similarly, in step 7 of the recipe, you asked iReport to save the path to subreport file. In response to this, iReport authored the `<![CDATA[${SUBREPORT_DIR} + "CustomerInvoicesSubreport.jasper"]]>CDATA` section.
- ▶ iReport authored the `isRemoveLineWhenBlank="true"` attribute in response to your action of step 20, where you changed the **Remove Line When Blank** property of the subreport component.
- ▶ Similarly, in step 21, you changed the **Print When Expression** property of the subreport component and iReport authored the `<printWhenExpression>` tag in response.

7

Summary Report, Crosstabs, and Graphs

In this chapter, you will learn:

- ▶ Designing a simple summary report
- ▶ Designing a multi-level summary report
- ▶ Designing a crosstab—a table with dynamic rows and columns
- ▶ Displaying data trends as a graph in your report
- ▶ Embedding a bar graph inside a tabular view

Introduction

This chapter demonstrates summarizing your application data in the form of summary reports, crosstabs, bar graphs, and line graphs. This chapter also covers the use of subreports to produce a concise summary of large amounts of data.

Designing a simple summary report

Many times you need to display only a summary of your report data and not the actual nitty-gritty details of the data. JasperReports provides you with an easy-to-use variety of aggregation functions to use on your data. This helps you to quickly build robust summary reports.

This recipe teaches you how to use this feature of aggregation to build a simple summary report.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb8` and copy sample data for this recipe into the database.

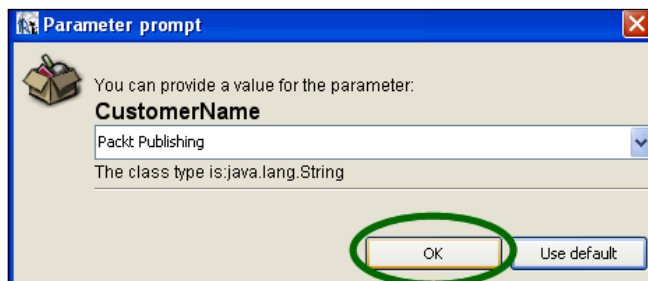
How to do it...

Now let's start designing a simple summary report by following the steps of this recipe:

1. Open the `CustSummary.jrxml` file from the `Task1` folder of the source code for this chapter. The **Designer** tab of **iReport** shows a report with a title, various fields in the **Page Header** section, and two static text components in the **Column Header** section. You will notice that all the other sections of the report, including the **Detail 1** section, are empty.

12 Months Order Summary of \$F	
Customer Name: \${CustomerName}	
Number of Invoices: \${REPORT_COUNT}	
Total of Invoice Values:	
Following is the last 12 months order history for the customer named \${CustomerName}	
Month	Order Value
InvoicePeriod1 Group Header 1	
Page Footer	

2. Switch to the **Preview** tab. A **Parameter prompt** dialog will appear, which will ask you to enter the name of the customer. Type `Packt Publishing` as the name of the customer and press the **OK** button.



3. A summary report for the particular customer will appear. You will see that the summary report shows the **Customer Name** and the **Number of Invoices**, but the report is missing **Total of Invoice Values** and actual monthly order values.

12 Months Order Summary of Packt Publishing

Customer Name: Packt Publishing

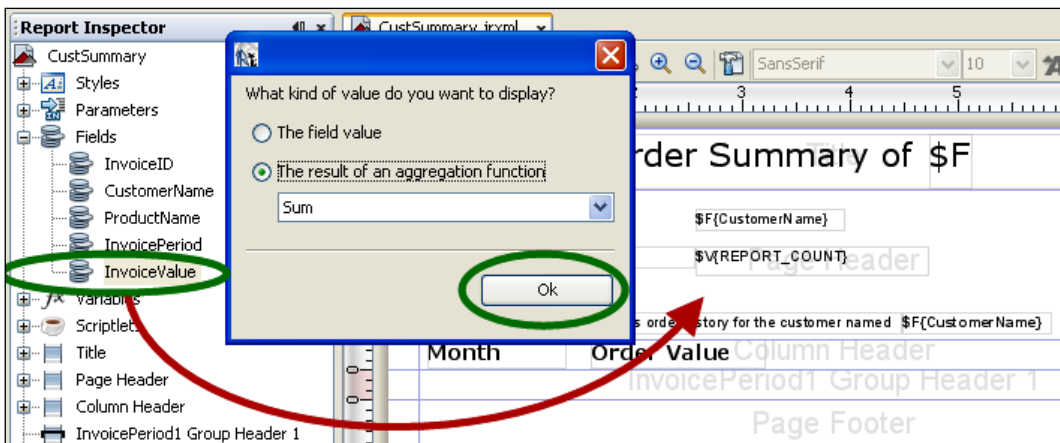
Number of Invoices: 42

Total of Invoice Values:

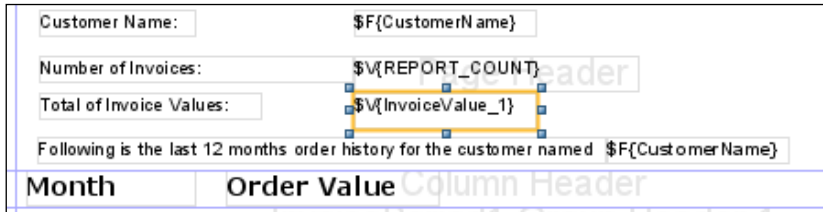
Following is the last 12 months order history for the customer named **Packt Publishing**

Month	Order Value
 	

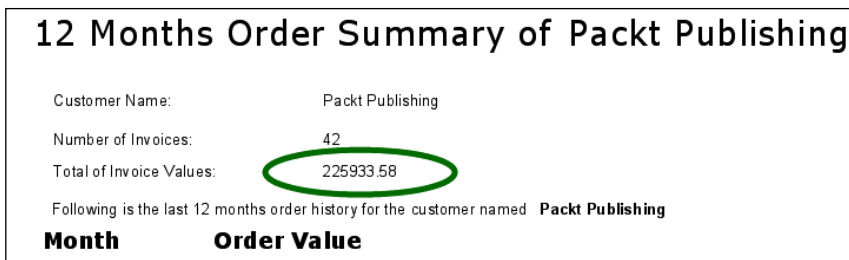
4. Switch back to the **Designer** tab. Expand the **Fields** node from the **Report Inspector** window on the left of the **Designer** tab. The **Fields** node will expand to show all the fields. You will find the **InvoiceValue** field at the end of the list of fields. Drag-and-drop the **InvoiceValue** field from the **Report Inspector** into the **Page Header** section. A dialog will appear. Select **The result of an aggregation function** radio button and the **Sum** option from the combo box in the dialog and click the **Ok** button, as shown in the following screenshot:



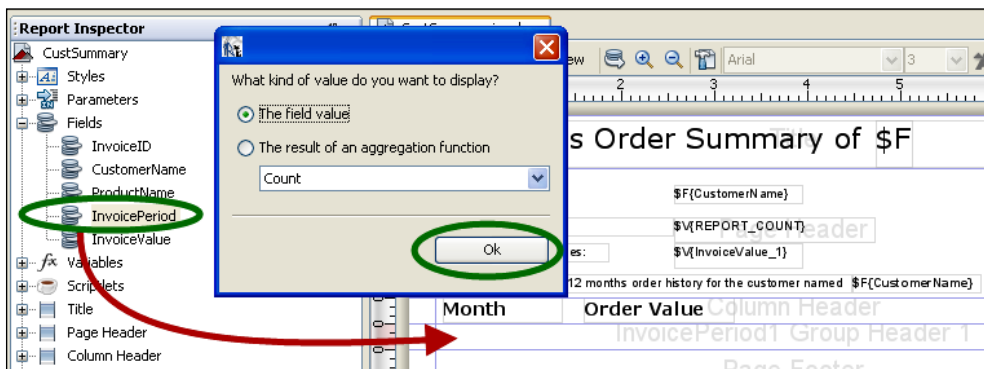
- A text field component with the expression `$V{InvoiceValue_1}` will appear in the **Page Header** section of your report. Using your mouse, align this text field component in front of the **Total of Invoices Value:** (for more clarification refer to the screenshot below this step) static text component, as shown in the following screenshot:



- Switch to the **Preview** tab. A **Parameter prompt** dialog will appear, which will ask you to enter the name of the customer. Type **Packt Publishing** as the name of the customer and click the **OK** button. You will see that the summary report this time shows the total of invoice values for Packt Publishing, as shown next. This **Total of Invoice Values** was missing in the previous preview of step 3.



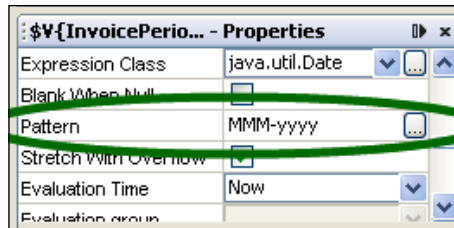
- Switch back to the **Designer** tab. Expand the **Fields** node from the **Report Inspector** window. The **Fields** node will expand to show all the fields. You will find the **InvoicePeriod** field at the second last position, just above the **InvoiceValue**. Drag-and-drop the **InvoicePeriod** field from the **Report Inspector** into the **InvoicePeriod1 Group Header 1** section. A dialog will appear. Select **The field value** radio button in the dialog and click the **Ok** button, as shown in the following screenshot:



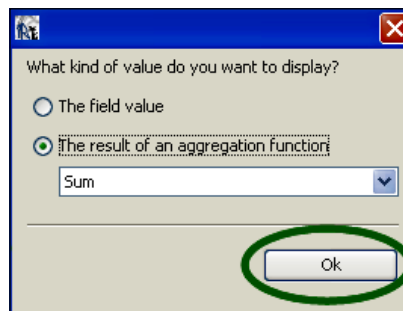
8. A text field component with the expression `$F{InvoicePeriod}` will appear in the **InvoicePeriod1 Group Header 1** section of your report. Using your mouse, align this text field component just below the **Month** static text component, in the **Column Header** section, as shown in the following screenshot:



9. While the text field component of the **InvoicePeriod1 Group Header 1** section is selected, the **Properties** window below the **Palette** of components will show the properties of the **Text Field** component. Find the **Pattern** property in the **Properties** window and type `MMM-yyyy` as its value as shown in the following screenshot:



10. Similarly, drag-and-drop the **InvoiceValue** from the **Report Inspector** into the **InvoicePeriod1 Group Header** section of your report. A dialog will appear. Select the **The result of an aggregation function** radio button in the dialog and the **Sum** option from the combo box in the dialog. Then click the **Ok** button, as shown in the following screenshot:



11. A **Text Field** component with the expression `$V{InvoiceValue_2}` will appear in the **InvoicePeriod1 Group Header 1** section of your report. Using your mouse, align this **Text Field** component just below the **Order Value** static text component, in the **Column Header** section, as shown in the following screenshot:



12. Switch to the **Preview** tab. A **Parameter prompt** dialog will appear, which will ask you to enter the name of the customer. Type **Packt Publishing** as the name of the customer and click the **OK** button. You will see a preview of your report, as shown next. This time you will notice that actual monthly order values are shown in the summary.

12 Months Order Summary of Packt Publishing	
Customer Name:	Packt Publishing
Number of Invoices:	42
Total of Invoice Values:	225933.58
Following is the last 12 months order history for the customer named Packt Publishing	
Month	Order Value
Jan-2010	50141.07
Feb-2010	13635.0
Mar-2010	84098.05
Apr-2010	3000.5
May-2010	17320.0
Jun-2010	30325.95
Aug-2010	15422.5
Sep-2010	7550.5
Oct-2010	4440.0

How it works...

Note that the **Detail 1** section remained empty throughout this recipe. That's because you want to process the **InvoiceValue** records to generate a summary. You don't want to display individual records.

If you dropped anything into the **Detail 1** section, it would display it for every record. So instead of dropping the **InvoiceValue** field into the **Detail 1** section, in step 11, you dropped the **InvoiceValue** field into the **InvoicePeriod1 Group Header 1** section. This section displays its data only when the **InvoicePeriod** (that is month) changes. As a result, only monthly records are shown instead of each individual record.

Now notice from the screenshot of step 11 that the sum of all invoices is shown in the **Page Header** section of your report. On the other hand, the sum of the monthly invoice values is shown in the **InvoicePeriod1 Group Header 1** section in the same screenshot. How does **JasperReports** handle this differentiation?

iReport intelligently writes JRXML code for you. It watches which component you are dropping and into which section. Depending upon the component and section, it asks you some questions through dialogs and writes specific JRXML code for you, which meets your exact requirements.

Refer to step 4 of this recipe, where you dragged-and-dropped the **InvoiceValue** field into the **Page Header** section of your report. **iReport** asked you to use the actual **InvoiceValue** field or to perform some aggregation process on the value. You chose to sum the invoice values. As a result, **iReport** generated the JRXML code to sum all the invoice values for the particular customer.

Compare it with step 10 of this recipe, where you dragged-and-dropped the same **InvoiceValue** field into the **InvoicePeriod1 Group Header 1** section and again chose to sum the invoice values. This time, **iReport** generated the JRXML code to sum the invoice values for the particular customer and for a particular month. That's because **iReport** knows that you are dropping the same **InvoiceValue** field into a group that is based on **InvoicePeriod**.

Designing a multi-level summary report

You may need to generate a summary report that collects, processes, and summarizes large amounts of data into a single document. JasperReports allows you to do that.

In this recipe, you will design a monthly billing report for an electricity supplying company. The company has many feeders to supply electricity; each feeder feeds into many transformers, and each transformer serves many electricity consumers. The report of this recipe calculates the sum of monthly bills of all consumers connected to a transformer. Then it gathers the billing data of all transformers of a feeder into a single monthly feeder billing summary.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

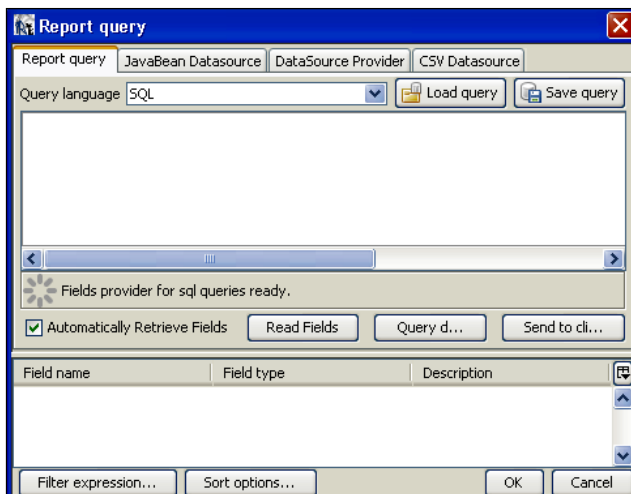
The source code for this chapter also includes a file named `CopyFeederDataIntoPG.txt`, which helps you to create a database named `jasperdb8` and two tables named `Feeder` and `Transformer`. The text file also helps you to copy sample data for this recipe into the tables.

You will be using two JRXML files, `MonthlyTransformerBill.jrxml` and `FeederSummaryReport.jrxml`, in this recipe. You will find these files in the `Task2` folder of the source code download for this chapter. The `FeederSummaryReport.jrxml` file is the main report, which uses the subreport named `MonthlyTransformerBill.jrxml`. The main report has to refer to its subreport using a complete path (you cannot use relative paths). This means you have to copy the main and subreport JRXML files to the `C:\JasperReportsCookBookSamples\` folder on your PC. I have hardcoded this complete path in the master report (`FeederSummaryReport.jrxml`).

How to do it...

Let's discover how to work with a multi-level summary report by going through the steps of the recipe:

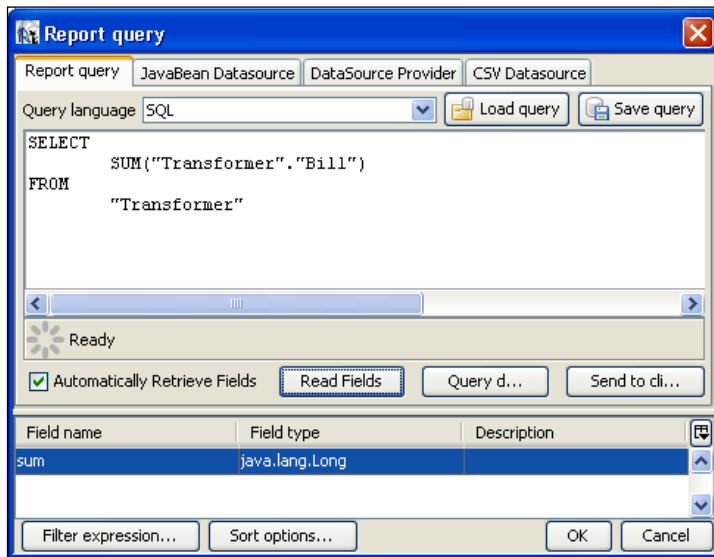
1. Open the `MonthlyTransformerBill.jrxml` file from the `C:\JasperReportsCookBookSamples` folder on your PC. The **Designer** tab of iReport shows an empty report.
2. Open the **Report query** window by clicking the **Report query** button on the right side of the **Preview** tab button. A **Report query** window will appear, as shown in the following screenshot:



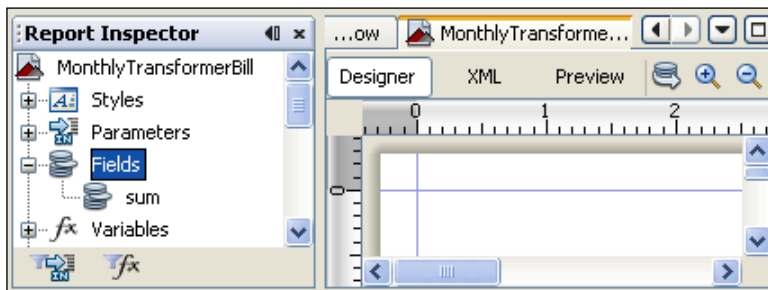
3. Enter the following SQL query into the **Report query** window:

```
SELECT SUM("Transformer"."Bill") FROM "Transformer"
```

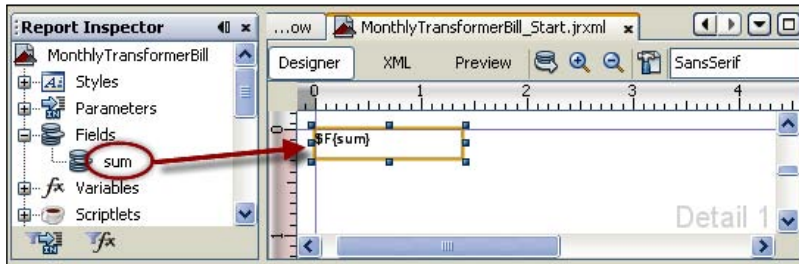
The lower part of the **Report query** window will show a **sum** field. Click **OK**.



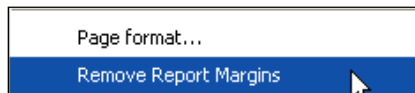
4. From the **Report Inspector** window on the left-side of the **Designer** tab, double-click the **Fields** node to expand it. You will notice a single field named **sum** in the expanded fields list, as shown in the following screenshot:



5. Drag-and-drop the **sum** field from the **Fields** list into the **Detail 1** section at the top-left corner of your report, as shown in the following screenshot:



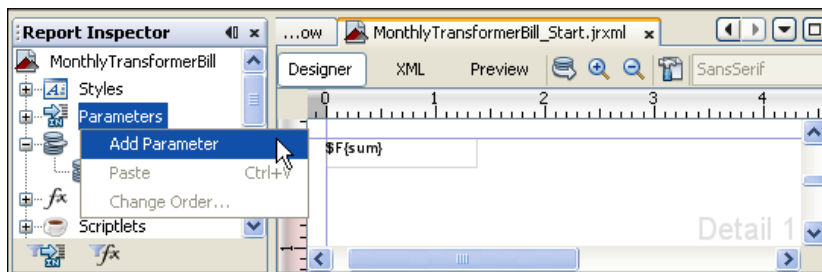
6. Click **Format** from the main iReport menu and select the **Remove Report Margins** option, as shown in the following screenshot:



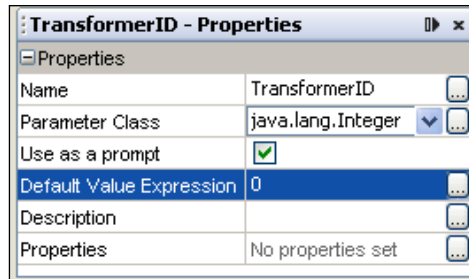
7. Double-click the blue line at the bottom of the **Detail 1** section. The height of the **Detail 1** section will become equal to the height of the **sum** field you dropped into the **Detail 1** section in step 5.
8. Click anywhere in the **Title** section; its properties will appear in the **Properties** window below the **Palette**. Select the **Band height** property and set 0 as its value.
9. Similarly, repeat step 8 for the **Page Header**, **Column Header**, **Column Footer**, **Page Footer**, and **Summary** sections and set 0 as the value for their **Band height** properties. The report in the **Designer** tab will look as shown:



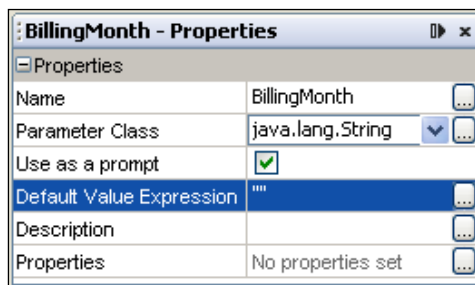
10. Right-click the **Parameters** node in the **Report Inspector** window on the left of the **Designer** tab. A pop-up menu will appear. Choose the **Add Parameter** option from the pop-up menu.



11. The **Parameters** node will expand to show the newly added parameter named **parameter1** at the end of the parameters list. Select **parameter1** from the list; its properties will appear in the **Properties** window below the **Palette**.
12. Click on the **Name** property of the parameter and type `TransformerID` as its value. The name of the **parameter1** parameter will change to **TransformerID**.
13. Click on the **Parameter Class** property and select `java.lang.Integer` as its value.
14. Click on the **Default Value Expression** property and enter `0` as its value. Leave the rest of the parameter properties at their default values.



15. Now right-click on the **Parameters** node in the **Report Inspector** window. A pop-up menu will appear. Choose the **Add Parameter** option. The **Parameters** node will expand to show the newly added parameter named **parameter1** at the end of the parameters list. Select **parameter1** from the list; its properties will appear in the **Properties** window below the **Palette**. Click on the **Name** property of the parameter in the **Properties** window and type `BillingMonth` as its value. The name of the parameter will change to **BillingMonth**.
16. Click on the **Default Value Expression** property and enter `""` as its value.



17. Open the **Report query** window by clicking the **Report query** button on the right of the **Preview** tab button. A **Report query** window will appear.

18. Type the following SQL query at the end of the existing text into the **Report query** window that you previously entered in step 3:

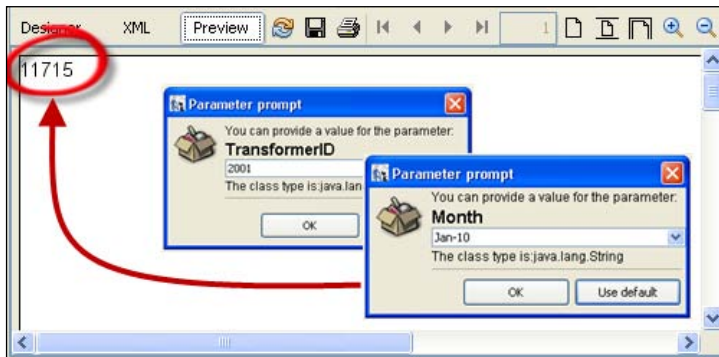
```
WHERE "Transformer"."TransformerID" = $P{TransformerID}
AND "Transformer"."Month" = $P{BillingMonth}
```

The complete expression in the editor will become:

```
SELECT SUM("Transformer"."Bill") FROM "Transformer"
WHERE "Transformer"."TransformerID" = $P{TransformerID}
AND "Transformer"."Month" = $P{BillingMonth}
```

Click **OK**.

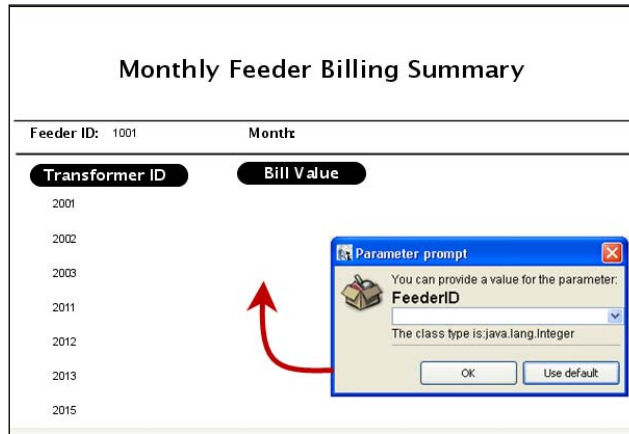
19. Switch to the **Preview** tab. You will see a **Parameter prompt** dialog, which will ask you to enter a **TransformerID**. Type 2001 as the value and click **OK**. Another parameter dialog will appear, which will ask you to enter the value of the billing month. Type Jan-10 as the value and click **OK**. The preview will appear, which will show just a figure, as shown next. The figure is actually the sum of the bills of all consumers connected to a transformer. Your `MonthlyTransformerBill.jrxml` subreport is now ready.



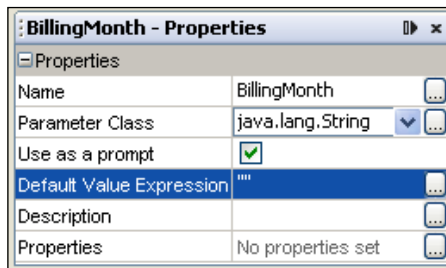
20. Switch back to the **Designer** tab. Open the `FeederSummaryReport.jrxml` file from the `C:\JasperReportsCookBookSamples\` folder on your PC. The **Designer** tab of iReport shows a report containing data in the **Title**, **Page Header**, **Column Header**, and **Detail 1** sections, as shown in the following screenshot:

Monthly Feeder Billing Summary	
Feeder ID: \$F{FeederID}	Month: \$F{Month}
Transformer ID	Bill Value
\$F{TransformerID}	

21. Switch to the **Preview** tab. You will see a **Parameter prompt** dialog, enter 1001 as the value of the **FeederID** parameter. It will display a list for transformers of the specified feeder:

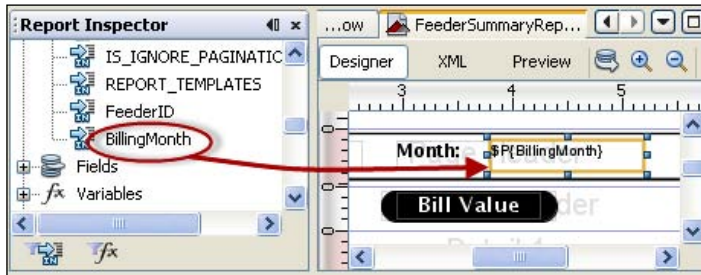


22. Switch back to the **Designer** tab. Right-click on the **Parameters** node in the **Report Inspector** window. A pop-up menu will appear. Choose the **Add Parameter** option. The **Parameters** node will expand to show the newly added parameter named `parameter1` at the end of the parameters list. Select `parameter1` from the list; its properties will appear in the **Properties** window below the **Palette**. Click on the **Name** property of the parameter in the **Properties** window and type `BillingMonth` as its value. The name of the parameter will change to `BillingMonth`.
23. Click on the **Default Value Expression** property and enter `""` as its value.

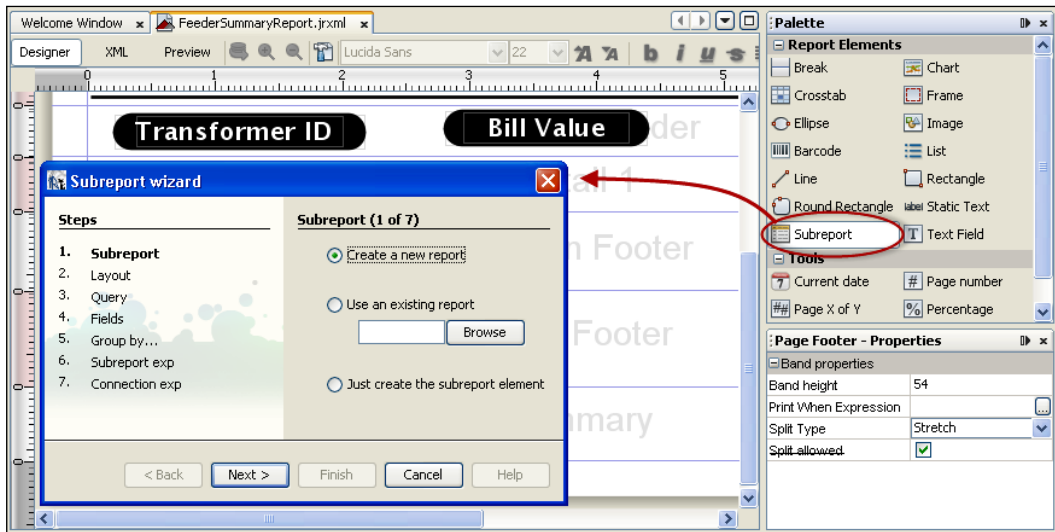


24. Double-click the **Parameters** node in the **Report Inspector** window on the left side of the **Designer** tab. The **Parameters** node will expand to show a list of parameters.

25. Drag-and-drop the **BillingMonth** parameter from the **Parameters** node into the **Page Header** section on the right side of the static text component labeled **Month**:

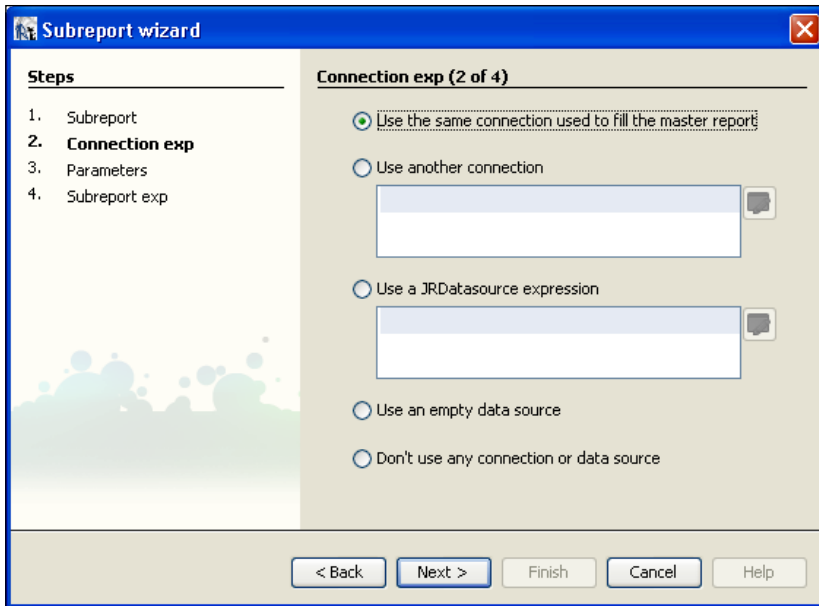


26. Drag-and-drop a **Subreport** component from the **Palette** into the **Detail 1** section just below the **Bill Value** column header. A **Subreport wizard** dialog will appear, as shown in the following screenshot:

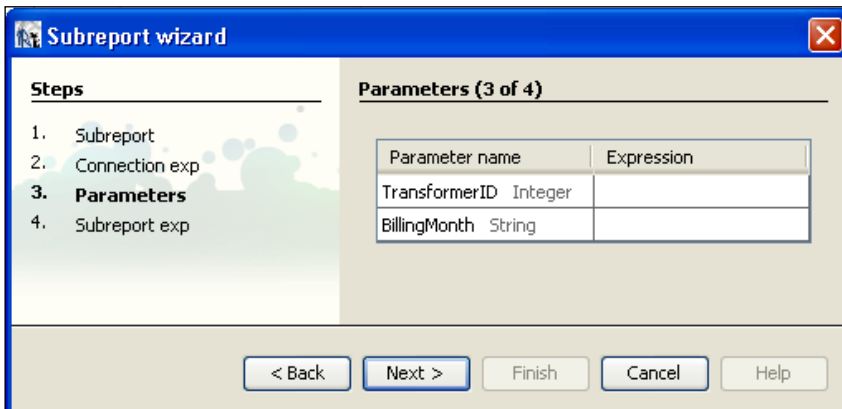


27. Select the **Use an existing report** option and click the **Browse** button to browse to the `MonthlyTransformerBill.jrxml` subreport file located in the `C:\JasperReportsCookBookSamples` folder and click the **Open** button. The browser dialog will disappear. Click the **Next** button in the **Subreport wizard** dialog.

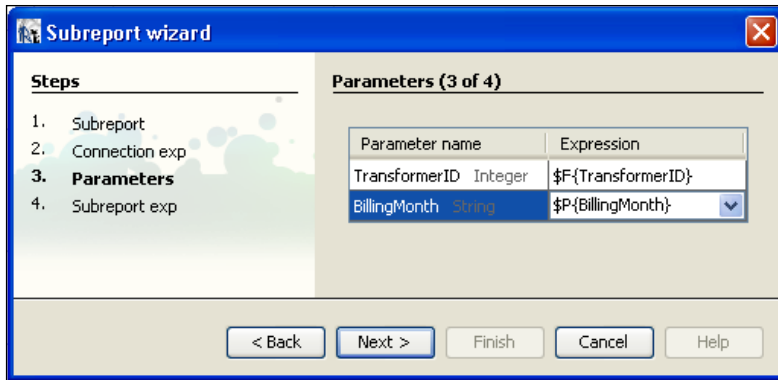
28. A **Connection exp (2 of 4)** dialog will appear, as shown next. Notice that now the left side of the **Subreport wizard** shows four steps (**Subreport**, **Connection exp**, **Parameters**, **Subreport exp**). You are at step 2 (**Connection exp**), which is shown in bold. Click the **Next** button.



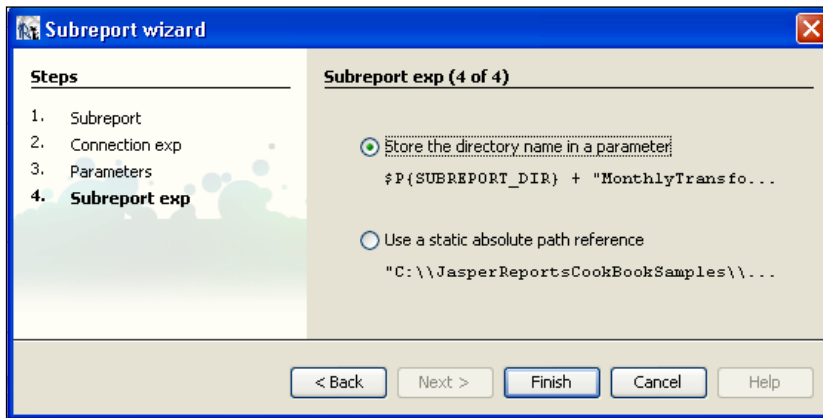
29. A **Parameters (3 of 4)** dialog will appear. This will show the TransformerID and BillingMonth parameters of the MonthlyTransformerBill .jrxml subreport and allow you to enter an expression to map these parameters to elements (that is, **Fields** or **Variables**) of the main FeederSummaryReport .jrxml report.



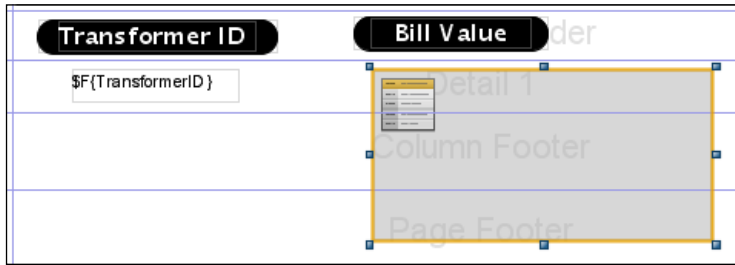
30. Click on the **Expression** field beside the **TransformerID** parameter. A drop-down list will open. This will show all the elements of the main `FeederSummaryReport.jrxml` report in the form of a drop-down list. Select the **TransformerID** field from the list.
31. Now click on the expression field beside the **BillingMonth** parameter. A drop-down list will open; select **BillingMonth** parameter from the list. Click the **Next** button.



32. A **Subreport exp (4 of 4)** dialog will appear. Continue with the **Store the directory name in a parameter** option selected and click the **Finish** button.



33. The subreport element will be placed in the **Detail 1** section.



34. Click on the subreport element; its properties will appear in the **Properties** window below the **Palette**. Find the **Height** property and set 19 as its value. The size of the subreport element will become appropriate according to the size of the other field placed in the **Detail 1** section. Use your mouse to align this subreport element in the **Detail 1** section, as shown in the following screenshot:



35. Click the **Preview** button. You will see the **Parameter prompt** dialogs. Enter 1001 and Jan-10 as value of feeder ID and billing month parameters. The report will show bill values for all transformers of the specified feeder for the particular month, as shown in the following screenshot:

Monthly Feeder Billing Summary	
Feeder ID	1001
Month	Jan-10
Transformer ID	Bill Value
2001	11715
2002	16170
2003	12120
2011	9007
2012	8458
2013	14642
2015	7898

Bill values (for each transformer) coming from the subreport

How it works...

You have done two main things in this recipe. First you designed a subreport in steps 1 to 19. The subreport shows only one figure in its view, as shown in the screenshot for step 19. The figure represents the total bill of all customers connected to a particular transformer of the electricity supply company.

Notice from the SQL query of step 3 that it has a `SUM` function. The subreport uses the `SUM` function of SQL to calculate the sum of all bills.

Then in step 20, you opened a main report named `FeederSummaryReport.jrxml` and later in steps 26 to 33 you inserted the `MonthlySummaryReport.jrxml` subreport in the **Detail 1** section of the main report. Anything that resides inside the **Detail 1** section gets processed once with every record. Each record in the main report represents a transformer of the electric supply company. Therefore, the main `FeederSummaryReport.jrxml` report gets a copy of the subreport for each transformer.

Recall from steps 30 and 31 of the recipe that you mapped `TransformerID` and `BillingMonth` parameters of the main `FeederSummaryReport.jrxml` report to the same parameters of the `MonthlyTransformerBill.jrxml` subreport. This means the main `FeederSummaryReport.jrxml` report will pass the transformer ID and billing month to the subreport. The subreport will provide the total bill of all consumers connected to that transformer for the particular month. This way, the main `FeederSummaryReport.jrxml` report passes all transformer IDs one by one to the subreport, with the same billing month every time, and gathers the bills of all transformers of the particular feeder into the monthly feeder billing summary.

You selected the **Store the directory name in a parameter** option in step 32. As a result, **iReport** stored the path to the `MonthlyTransformerBill.jrxml` subreport file in a parameter named `SUBREPORT_DIR`, which is useful whenever you need to relocate your subreport files (especially if there are multiple subreports in a master report).

Relocating subreports simply requires changing the value of the `SUBREPORT_DIR` parameter, which you do directly from the **Report Inspector** window. For this purpose, you will first click on the `SUBREPORT_DIR` parameter in the **Parameters** node from the **Report Inspector** window. Then you will find the **Default Value Expression** property from the **Properties** window and provide a path to the directory where your subreport files are located. For example, if your files are located in a directory named `JasperReportsCookBookSamples` on your `C:` drive, the value of the **Default Value Expression** property should be `C:\JasperReportsCookBookSamples\`.

Especially notice that the expression is given in double-quotes ("") and it uses double backslashes (\\) in the directory path notation.

Designing a crosstab—a table with dynamic rows and columns

Crosstabs are special tables, in which both columns and rows have names. Crosstabs are very helpful in presenting summaries of large amounts of data. For example, a crosstab can be used to design a 12-month summary report to show monthly invoice totals for each of your customers.

Such a summary report will have customer names in the left-most column (meaning each row will belong to a unique customer) and month names in the top-most row (meaning each column will belong to a specific month). This way each cell of the crosstab will show the invoice total for a customer and for a specific month.

This recipe shows how you will use JasperReports to design such a crosstab.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

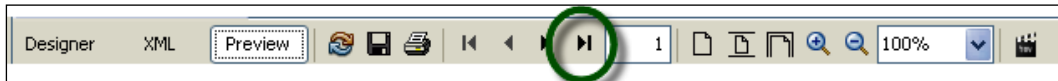
The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb8` and copy sample data for this recipe into the database.

How to do it...

1. Open the `CrossTab.jrxml` file from the `Task3` folder of the source code for this chapter. The **Designer** tab of iReport shows a report with data in the **Title**, **Customer Group Header 1**, **Product Group Header 1**, and **Detail 1** sections. Notice that the **Summary** section at the end is empty.

Monthly Customer Invoices				
Customer Name: \$F{CustomerName}				
\$F{ProductName}				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
\$F{InvoiceID}	\$F{CustomerName}	\$F{ProductName}	\$F{InvoicePeriod}	\$F{InvoiceValue}

- Switch to the **Preview** tab and click the last page button, as shown encircled in the following screenshot:

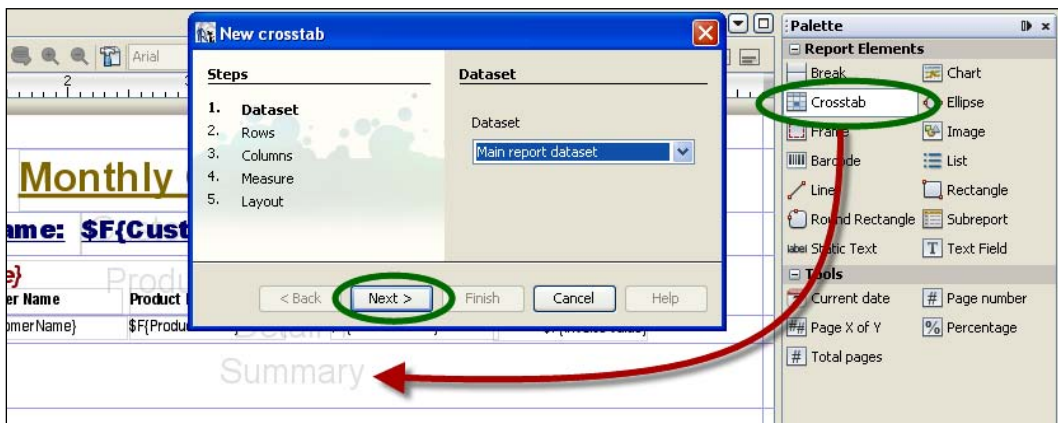


- You will see the last page of your report, as shown next. You will notice that there is no data in the **Summary** section.

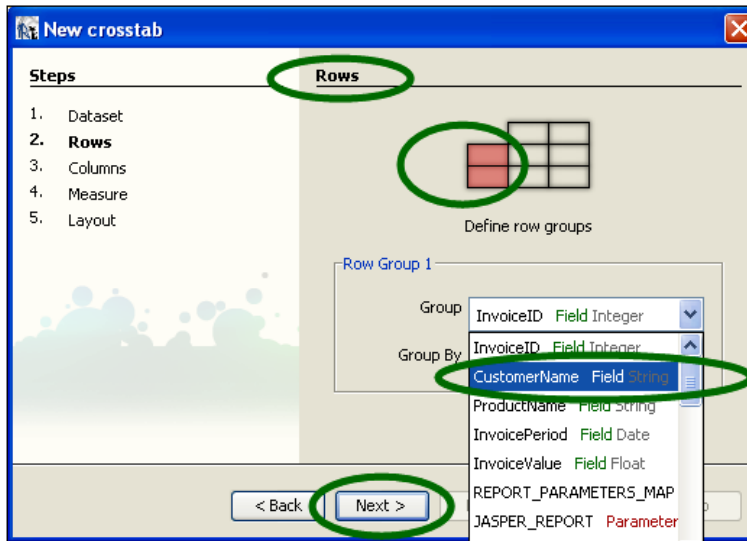
1046	Packt Publishing	Printing Film	Jan-2010	2558.0
1045	Packt Publishing	Printing Film	Mar-2010	8750.0
Printing Ink				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1017	Packt Publishing	Printing Ink	Aug-2010	6540.0
1031	Packt Publishing	Printing Ink	Jan-2010	5240.0
1030	Packt Publishing	Printing Ink	Jan-2010	5240.0
1027	Packt Publishing	Printing Ink	Oct-2010	4440.0
1025	Packt Publishing	Printing Ink	Aug-2010	5882.0
1019	Packt Publishing	Printing Ink	Jan-2010	5240.0
1042	Packt Publishing	Printing Ink	Mar-2010	8500.0

There is no data in the summary section

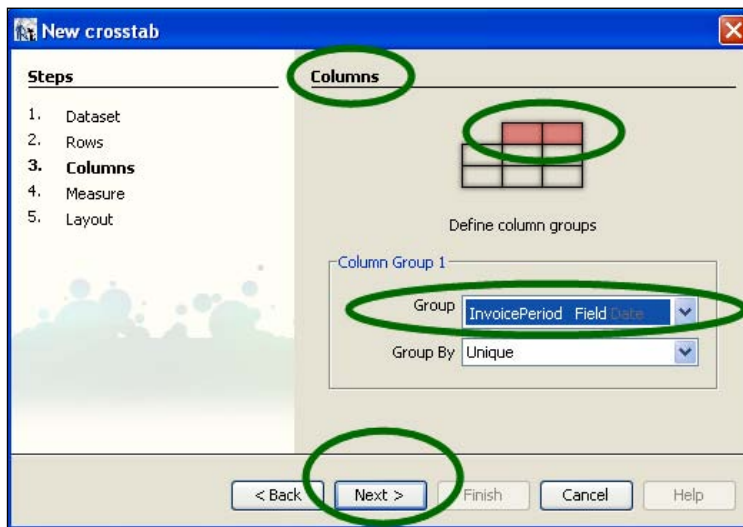
- Switch back to the **Designer** tab. Drag-and-drop a **Crosstab** component from the **Palette** of components into the **Summary** section of your report. A **New crosstab** dialog will appear. Click the **Next** button.



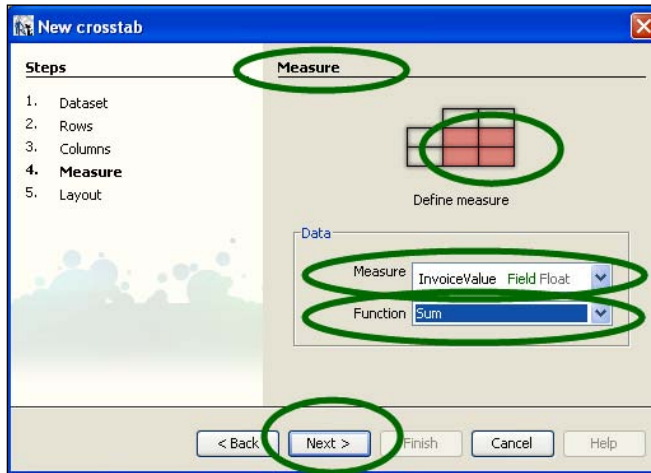
5. The **New crosstab** dialog will change its view. The right-half of the dialog is labeled as **Rows**. Click on the **Group** combo box under **Rows** heading. The combo box will expand. Select the **CustomerName Field** option, then click the **Next** button, as shown in the following screenshot:



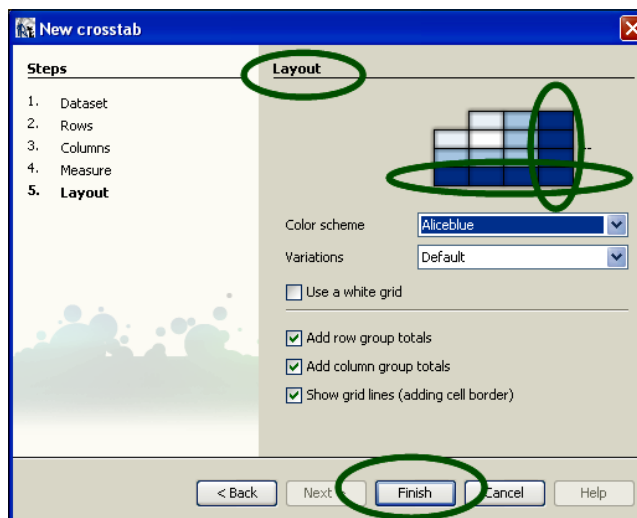
6. The **New crosstab** dialog will again change its view. The right-half of the dialog is now labeled as **Columns**. Click on the **Group** combo box under the **Columns** heading. The combo box will expand. Select the **InvoicePeriod Field** option, then click the **Next** button, as shown in the following screenshot:



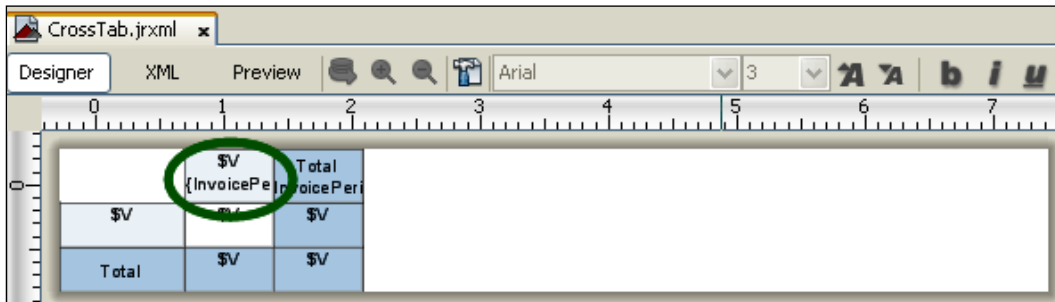
7. The **New crosstab** dialog will again change its view. The right half of the dialog is now labeled as **Measure**. Click on the **Measure** combo box under the **Measure** heading. The combo box will expand. Select the `InvoiceValue` Field option. Similarly, click on the **Function** combo box, below the **Measure** combo box, and the **Function** combo box will expand. Select the `Sum` option. Then click the **Next** button, as shown in the following screenshot:



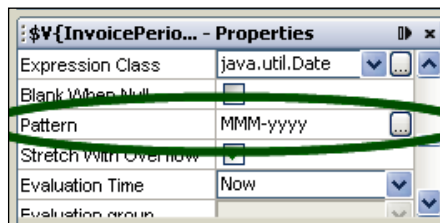
8. The **New crosstab** dialog will again change its view. The right half of the dialog is now labeled as **Layout**. This screen allows you to select a color scheme for your crosstab and whether you want to show column and row totals in your crosstab. This screen also allows you to show or hide grid lines in your crosstab. Leave everything as default and click the **Finish** button. The dialog will disappear.



9. The **Designer** tab of iReport will show the following view. Click on the text field component with the expression `$V{InvoicePeriod}` present in the middle position of the top row in the crosstab designing area. The **Properties** window below the **Palette** window will show properties of the text field component.



10. Find the **Pattern** property in the **Properties** window and type `MMM-yyyy` as its value, as shown in the following screenshot:



11. Switch to the **Preview** tab and click the last page button, as shown encircled in the following screenshot:



12. You will see that the last page of your report shows a crosstab, as shown in the following screenshot:

1046	Packt Publishing	Printing Film	Jan-2010	2558.0
1045	Packt Publishing	Printing Film	Mar-2010	8750.0

Printing Ink

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1017	Packt Publishing	Printing Ink	Aug-2010	6540.0
1031	Packt Publishing	Printing Ink	Jan-2010	5240.0
1030	Packt Publishing	Printing Ink	Jan-2010	5240.0
1027	Packt Publishing	Printing Ink	Oct-2010	4440.0
1025	Packt Publishing	Printing Ink	Aug-2010	5882.0
1019	Packt Publishing	Printing Ink	Jan-2010	5240.0
1042	Packt Publishing	Printing Ink	Mar-2010	8500.0

	Jan-2010	Feb-2010	Mar-2010	Apr-2010	May-2010	Jun-2010	Aug-2010	Sep-2010	Oct-2010
ABC Publishing	0.0	4552.2	3450.0	0.0	0.0	18060.5	9852.0	0.0	0.0
Alice	45.0	0.0	49.5	49.5	0.0	0.0	180.0	0.0	0.0
Bob	0.0	0.0	100.0	0.0	45.0	0.0	250.0	490.0	0.0
Packt	50141.07	13635.0	84098.05	3000.5	17320.0	30325.95	15422.5	7550.5	4440.0
Total	50186.07	18187.2	87697.55	3050.0	17365.0	48386.45	25704.5	8040.5	4440.0

	Total Invoice Period
ABC Publishing	35914.7
Alice	324.0
Bob	885.0
Packt	225933.58
Total	263057.28

How it works...

You have designed a crosstab in this recipe. While designing a crosstab, you need to specify four things:

1. First you have to insert a crosstab component into your report. This is what you did in steps 4 to 8 of this recipe.

2. Then you decide what data you need to show in the left-most column. Whatever you show in the left-most column acts as titles for rows. In this case, you selected customer names to be shown in the left-most column (refer to step 5 of this recipe). Now each row of the crosstab will be associated with a single customer.
3. Next you decide what data you need to display in the top row. Whatever you show in the top row acts as titles for the columns. In this case, you selected the invoice period (month) to be shown in the top row (refer to step 6 of this recipe). Now each column of the crosstab will be associated with a single month.
4. Now you know what is the meaning of rows and columns in your crosstab. As rows are associated with customers and columns are associated with months, each cell of the crosstab will represent *something* related to a particular customer and a particular month. So the next step is to specify what *something* you want to show in the individual cells of the crosstab. This is what you chose in step 7 of the recipe where you selected to display the sum of invoice values.

The result of these four steps is a crosstab that shows monthly sum totals of invoice values for each customer.

Displaying data trends as a graph in your report

You will often need to display your report data in the form of a graph or a chart. JasperReports allows you to use graphs and charts in your report.

In this recipe, you will learn how to present a summary of your data as a graph.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes a file named `copyFeederDataIntoPGS.txt`, which helps you to create a database named `jasperdb8` and two tables named `Feeder` and `Transformer`. The text file will also tell you how to copy sample data for this recipe into the tables.

How to do it...

The following steps will help you learn how to present data trends by using a graph at the end of your report:

1. Open the `Feeder_Trend.jrxml` file from the `Task4` folder of the source code for this chapter. The **Designer** tab of **iReport** shows that the report contains two groups named **Feeder** and **Month**. You will find a set of four components in the **Feeder Group Header 1** and **Month Group Header 1** sections. These four components together form a two-column table, as shown in the following screenshot:

Monthly Feeder Billing Summary of the Electricity Supply Company	
Feeder ID: \$F(FeederID)	
Month ID	Bill Value
\$F(Date)	\$V(Bill_1)

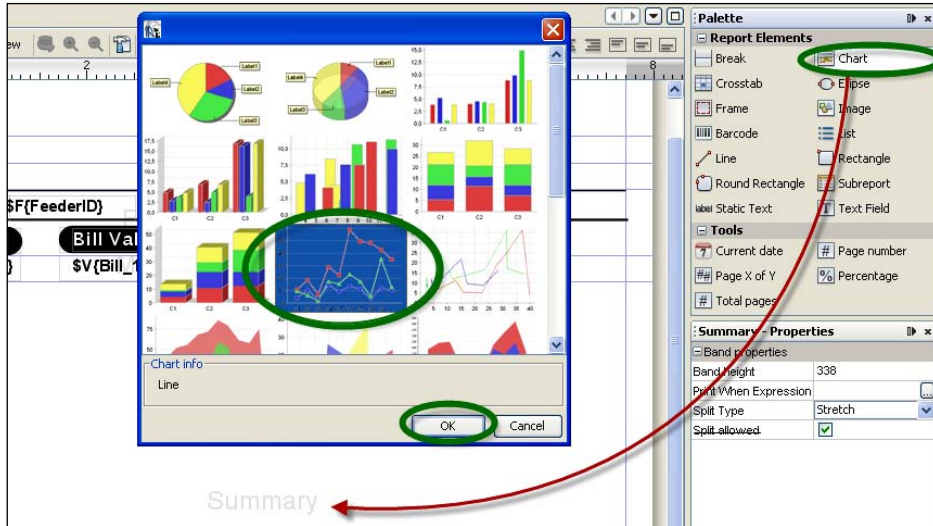
2. Switch to the **Preview** tab and click the last page button, as shown in the following screenshot:



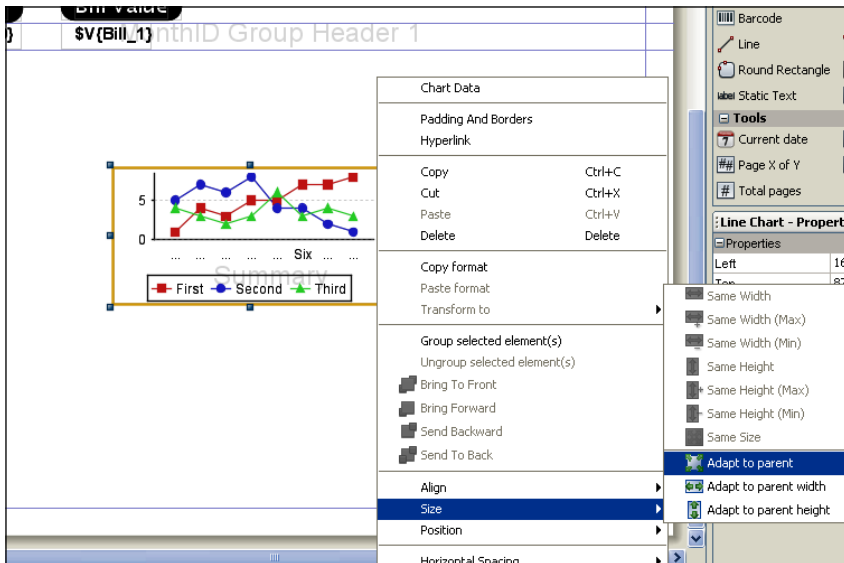
3. You will see that the last page of your report shows the monthly bills of different feeders of the electricity supply company.

Apr-10	89500
May-10	28780
Feeder ID: 1006	
Month ID	Bill Value
Jan-10	211540
Feb-10	47200
Mar-10	51900
Apr-10	89500
May-10	28780
Feeder ID: 1007	
Month ID	Bill Value
Jan-10	211540
Feb-10	47200
Mar-10	51900
Apr-10	89500
May-10	28780

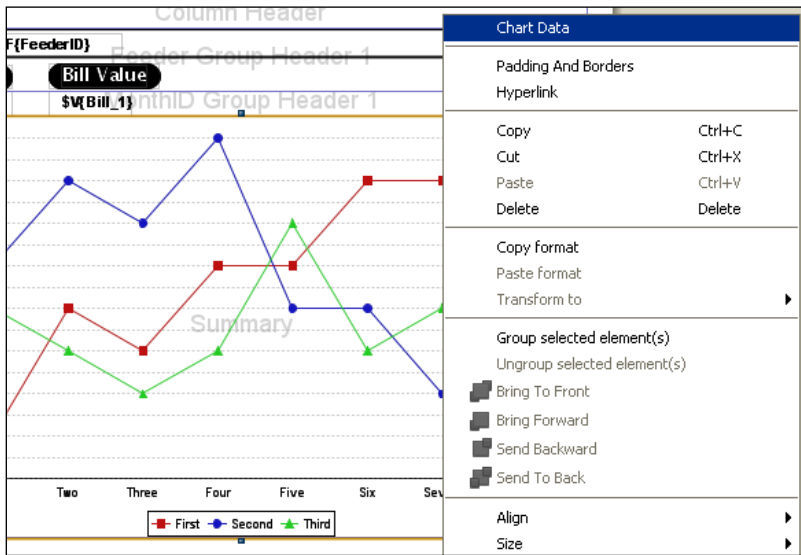
- Switch back to the **Designer** tab. Drag-and-drop a **Chart** component from the **Palette** into the **Summary** section of your report. A chart-type selection dialog will appear. Select **Line** as chart-type and click the **OK** button, as shown in the following screenshot:



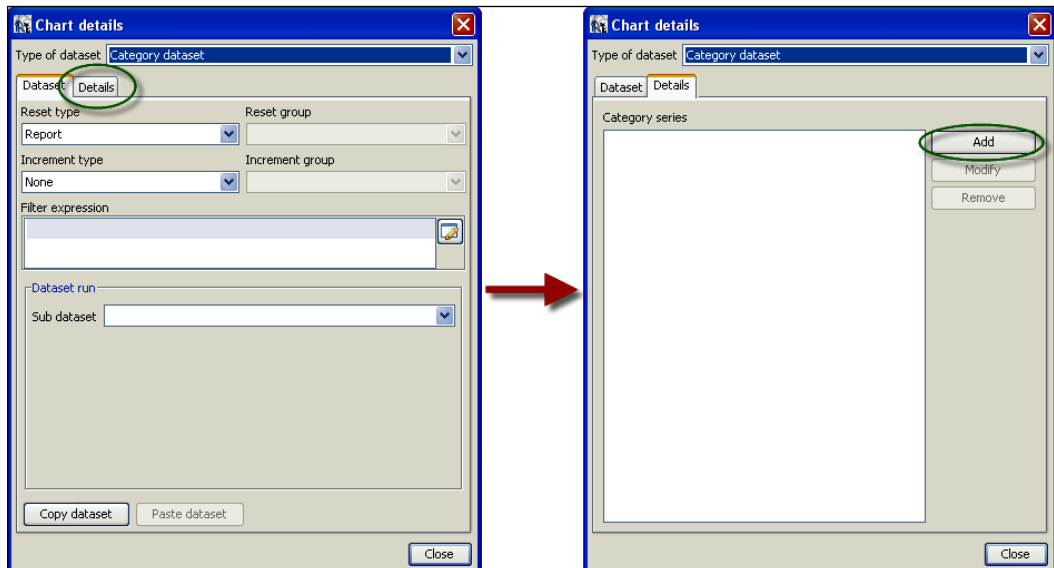
- The chart-type selection dialog will disappear and a **Chart** component will appear in the **Summary** section of your report. The **Chart** component looks like a graph. Right-click on this component, a pop-up menu will appear. Select the **Size** option from the pop-up menu, and a sub pop-up menu will appear. Select the **Adapt to parent** option from the sub pop-up menu, as shown next. The **Chart** component will expand to occupy all the available space in the **Summary** section.



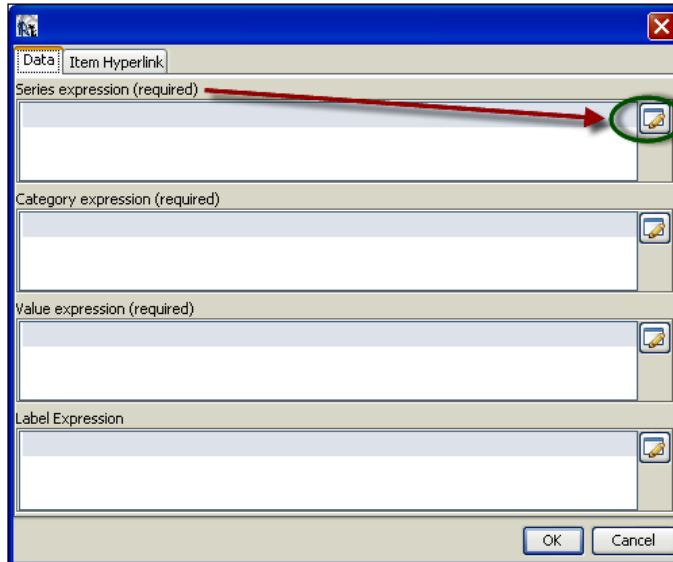
- Right-click on the **Chart** component again, and a pop-up menu will appear. Select the **Chart Data** option from the pop-up menu. A **Chart details** dialog will appear, which contains **Dataset** and **Details** tabs.



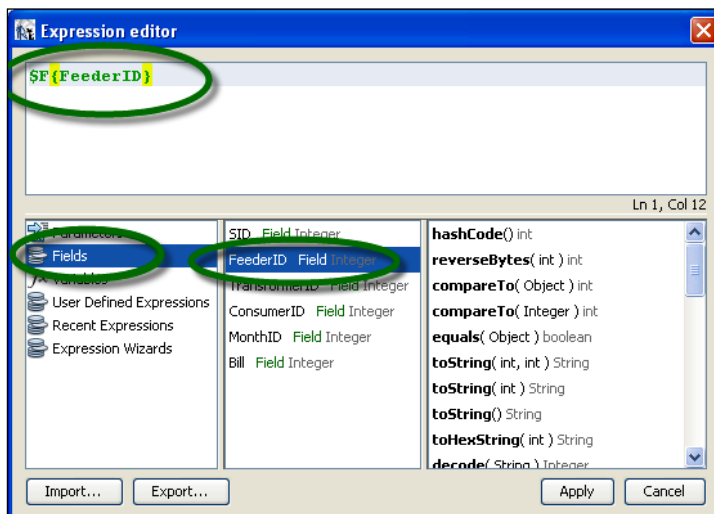
- Switch to the **Details** tab in the **Chart details** dialog and click the **Add** button, as shown in the following screenshot:



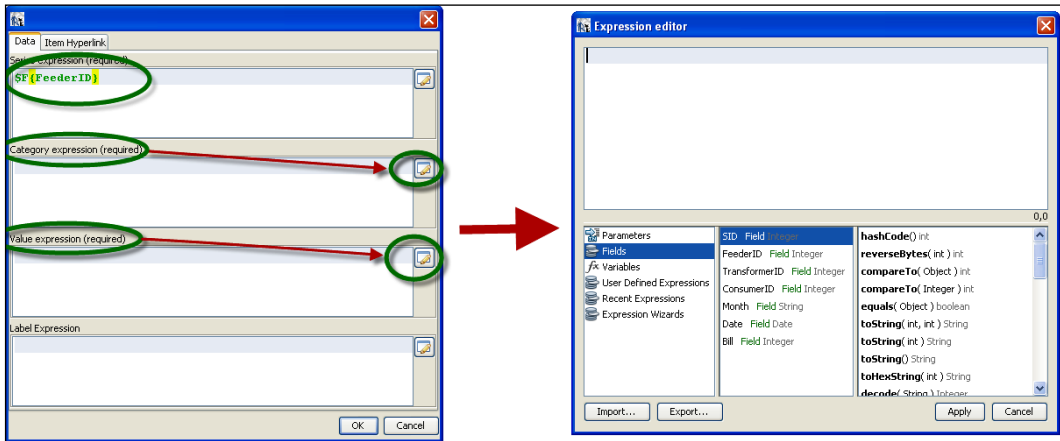
8. A dialog will appear, which allows you to enter various expressions (**Series expression**, **Category expression**, **Value expression**, and **Label Expression**). In order to add a **Series expression**, click the button beside it, as shown. An **Expression editor** dialog will appear.



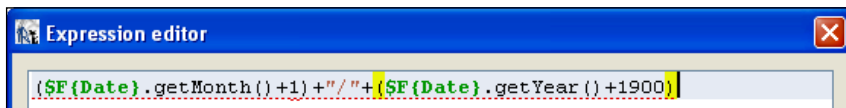
9. Click on the **Fields** option in the first column of the **Expression editor** and double-click on **FeederID Field** in the second column; a `$F{FeederID}` expression will appear in the upper portion of the dialog, as shown. Click the **Apply** button, and the **Expression editor** dialog will disappear.



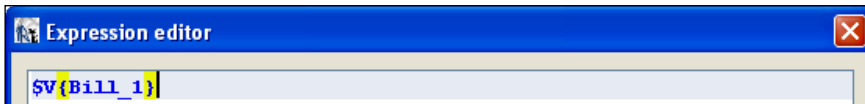
10. Notice that the `$F{FeederID}` expression will appear under **Series expression**. Now click the button beside **Category expression**, as shown next. Another **Expression editor** dialog will appear.



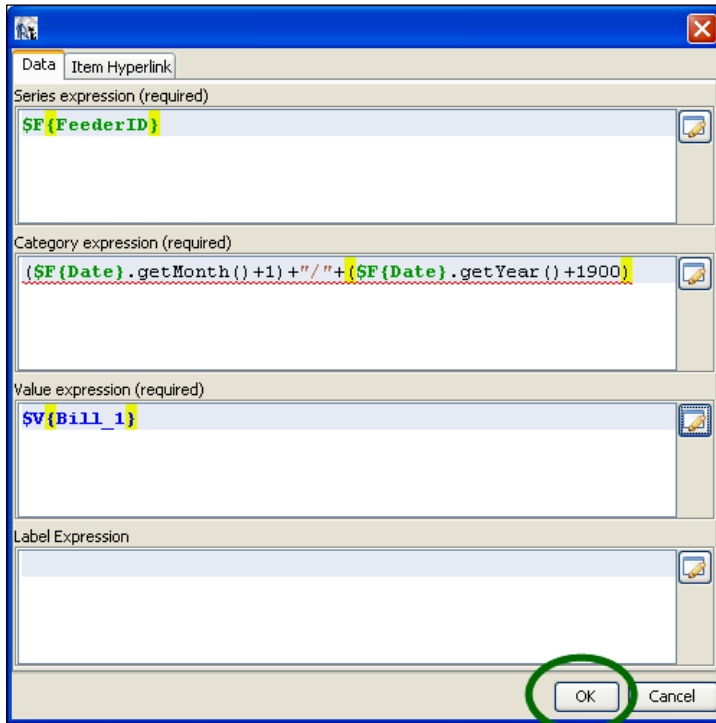
11. Type a `($F{Date}.getMonth()+1)+"/"+($F{Date}.getYear()+1900)` expression in the upper portion of the **Expression editor** dialog. Click the **Apply** button; the **Expression editor** dialog will disappear and you will be back to the previous dialog. You will notice the `$F{Date}.getMonth()+"/"+$F{Date}.getYear()` expression appearing under **Category expression**.



12. Now click the button beside the **Value expression**. Another **Expression editor** dialog will appear. Click on the **Variables** option in the first column of the **Expression editor** and double-click on **Bill_1 Variable** in the second column. The `$V{Bill_1}` expression will appear in the upper portion of the dialog. Click the **Apply** button; the **Expression editor** dialog will disappear and you will be back to the previous dialog.



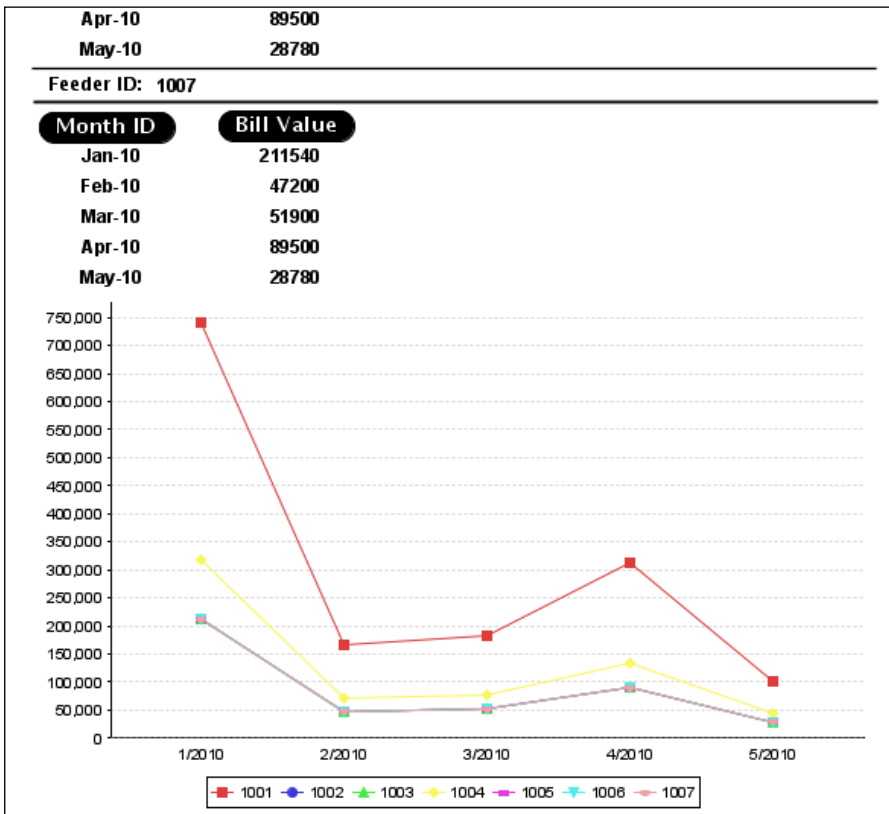
13. Now this dialog will look as shown in the following screenshot. Click the **OK** button, and the dialog will disappear. Now you will be back to the **Chart details** dialog of step 6.



14. Click the **Close** button at the bottom of **Chart details** dialog. This dialog will disappear.
15. Switch to the **Preview** tab and click the last-page button, as shown in the following screenshot:



16. You will see the last page of your report as shown next. You will notice that there is a graph that uses colored lines to show the trends of monthly bills for individual feeders.



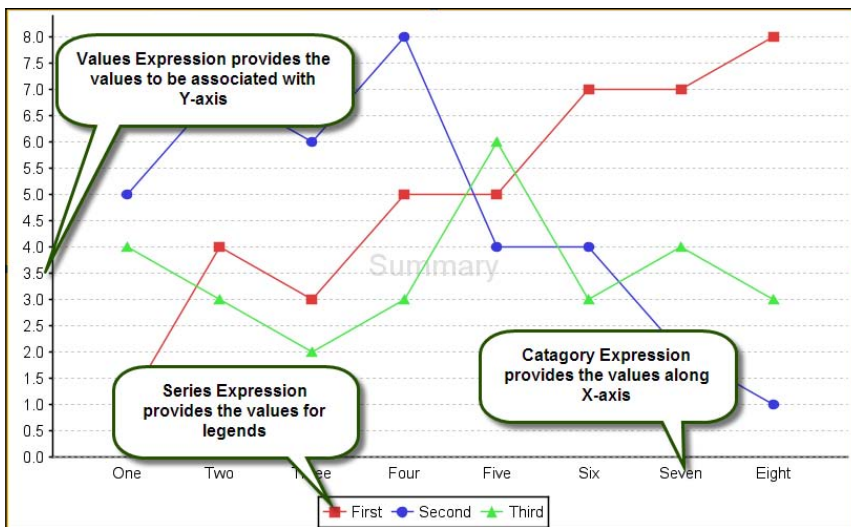
How it works...

In this recipe, you have designed a graph to summarize and show trends in the data of your report. In order to draw this graph, you have done the following four tasks:

1. In step 4 of the recipe, you dragged-and-dropped a **Chart** component into your report.
2. In steps 8 and 9 of the recipe, you provided a **Series expression** to the **Chart** component. Recall from the **Expression editor** of step 9 that you used `$F{FeederID}` as your **Series expression**. This specifies that you want to use **FeederID** to draw one complete trend (each feeder with a different colored line). JasperReports automatically included a legend to describe which color corresponds to which feeder. The legend is shown in the following screenshot:

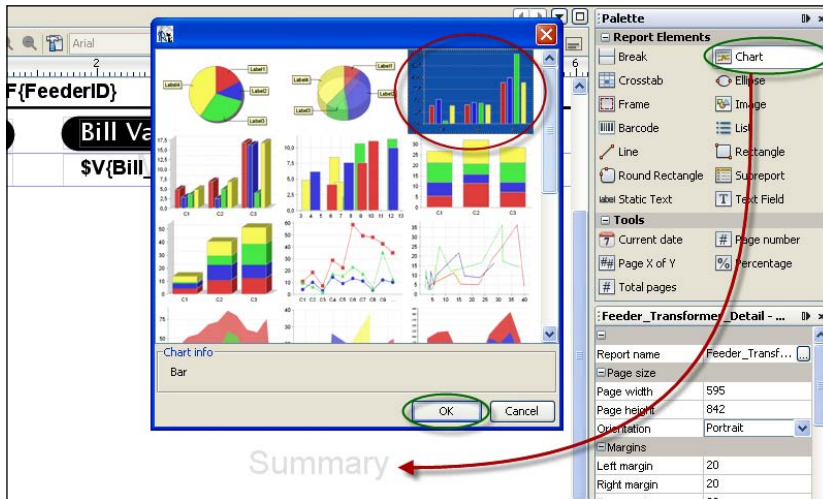
3. In steps 10 and 11 of the recipe, you provided a **Category expression** to the **Chart** component. Recall the **Expression editor** of step 11 where you used `($F{Date}.getMonth()+1)+"/"+($F{Date}.getYear()+1900)` as **Category expression**. This specifies what should be displayed along the x-axis of the graph. In your case, month/year are shown along the x-axis of the graph.
4. In step 12 of the recipe, you provided a **Value expression** to the **Chart** component. Recall from the **Expression editor** of this step that you provided `$V{Bill_1}` as **Value expression**. The **Bill_1** variable stores the sum of monthly bills of all transformers connected to a feeder. **JasperReports** automatically displays a scale for values along the y-axis of the graph.

The following figure shows all components of the graph mapped to their respective expressions:

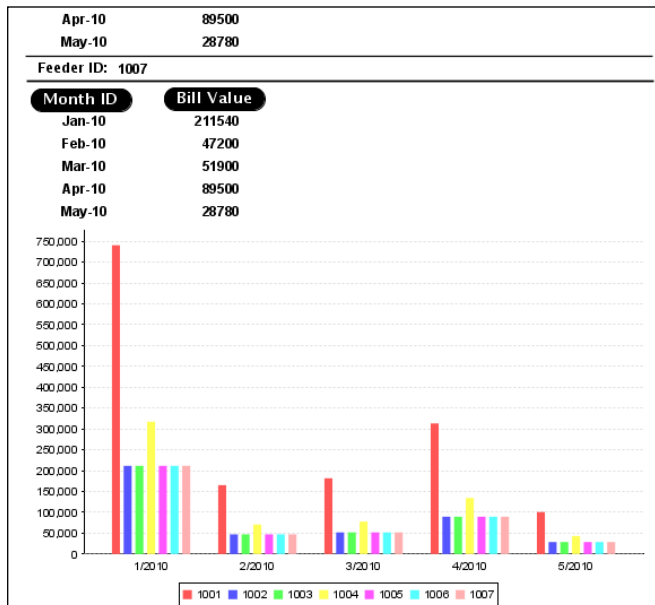


There's more...

JasperReports allows you to use different types of charts. In this recipe, you have designed a line graph using the **Chart** component of JasperReports. You can use the same **Chart** component in a very similar fashion to build a bar graph. You just have to select a bar graph (as shown encircled next) instead of the line chart in step 4 of the recipe.



All the other steps of the recipe will be the same, and you will end up with the following preview in the last step:



Embedding a bar graph inside a tabular view

You can use the flexibility of JasperReports graphs in many creative ways. This recipe teaches you a very interesting application, in which you can show colored bars of scaled size within a table. The size of each bar will represent a numeric value.

Getting ready

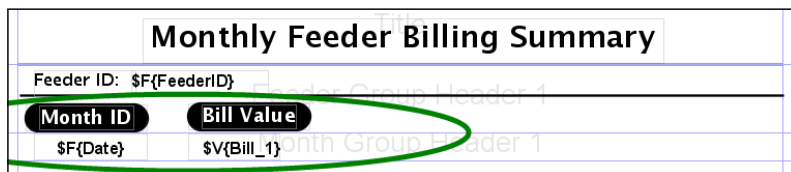
Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes a file named `copyFeederDataIntoPGS.txt`, which helps you to create a database named `jasperdb8` and two tables named `Feeder` and `Transformer`. The text file will also tell you how to copy sample data for this recipe into the tables.

How to do it...

The following steps demonstrate how to display a scaled bar beside each record in a table.

1. Open the `Feeder_BarChart.jrxml` file from the `Task5` folder of the source code for this chapter. The **Designer** tab of iReport shows that the report contains two groups, named **Feeder** and **Month**. You will find a set of four components in the **Feeder Group Header 1** and **Month Group Header 1** sections. These four components together form a two-column table, as shown in the following screenshot:



Monthly Feeder Billing Summary	
Feeder ID: \$F{FeederID}	
Month ID	Bill Value
\$F{Date}	\$V{Bill_1}

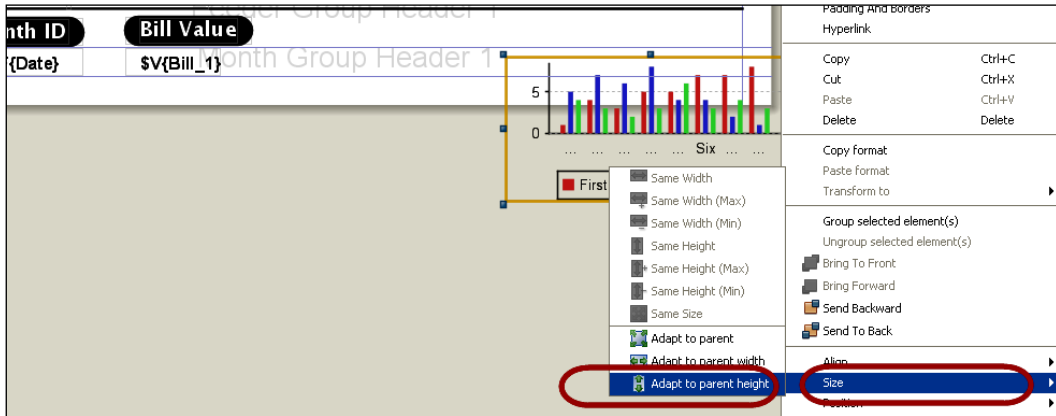
- Switch to the **Preview** tab. The preview of your report shows the monthly bills for different feeders of the electricity supply company. Notice that the right half of the report area is unused, as shown encircled in the following screenshot:

Monthly Feeder Billing Summary	
Feeder ID: 1001	
Month ID	Bill Value
Jan-10	740390
Feb-10	165200
Mar-10	181650
Apr-10	313250
May-10	100730
Feeder ID: 1002	
Month ID	Bill Value
Jan-10	211540
Feb-10	47200
Mar-10	51900
Apr-10	89500
May-10	28780
Feeder ID: 1003	
Month ID	Bill Value
Jan-10	211540

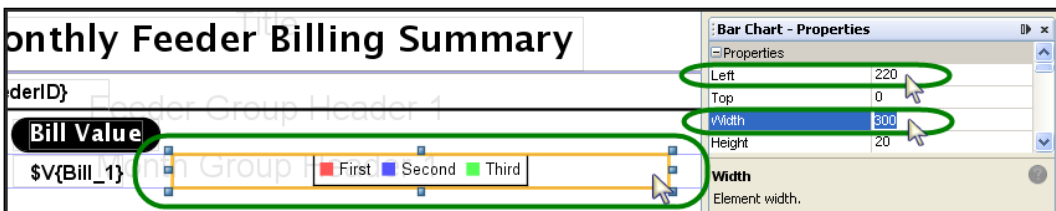
- Switch back to the **Designer** tab. Drag-and-drop a **Chart** component from the **Palette** into the **Month Group Header 1** section of your report. A chart type selection dialog will appear. Select **Bar** as chart type and click the **OK** button, as shown in the following screenshot:



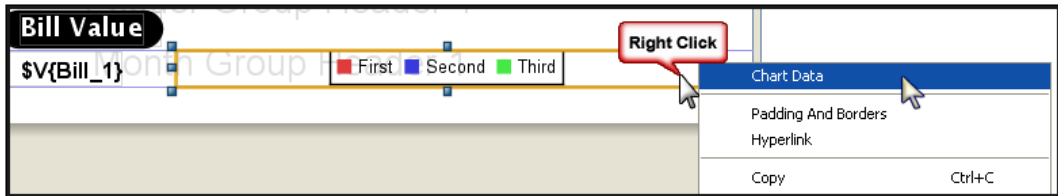
- The chart-type selection dialog will disappear and a large **Chart** component will appear in the **Month Group Header 1** section of your report. You will observe that the chart component is larger in height than the **Month Group Header 1** section. In order to match the height of this component with the height of the **Month Group Header 1** section, right-click on this component; a pop-up menu will appear. Select the **Size** option from the pop-up menu; a sub pop-up menu will appear. Select the **Adapt to parent height** option from the sub pop-up menu, as shown next. The **Chart** component will shrink vertically to adjust itself within the **Month Group Header 1** section.



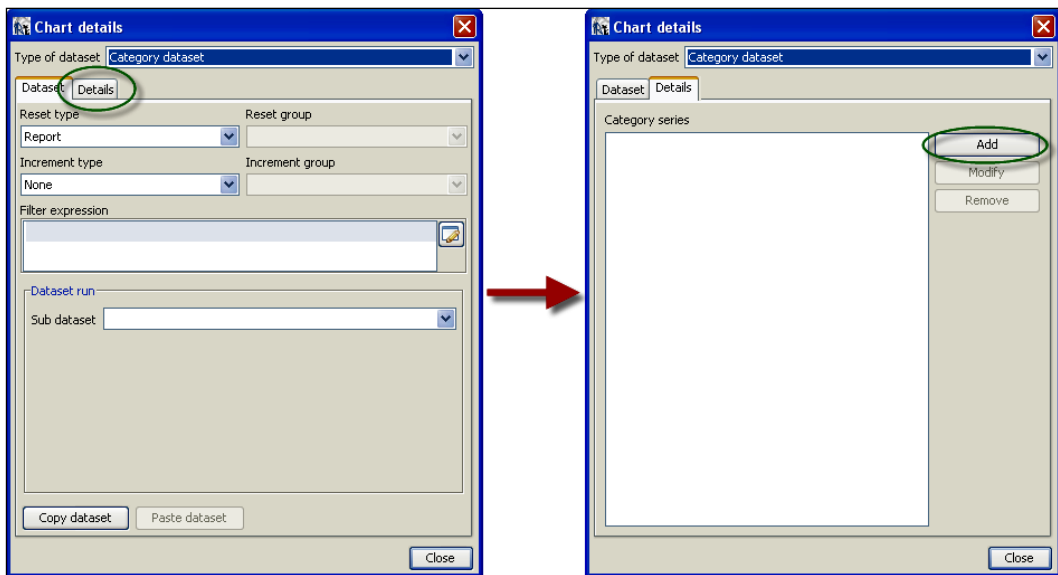
- While the **Chart** component is selected, its properties will appear in the **Properties** window below the **Palette**. Find the **Left** property and set 220 as its value. Find the **Width** property and set 300 as its value. The 220 value for the left property will position the chart correctly and the 300 value for the width property will size it appropriately in the **Month Group Header 1** section. The **Chart** component will expand and position itself horizontally, as shown in the following screenshot. Similarly, find the **Orientation** property and set **Horizontal** as its value.



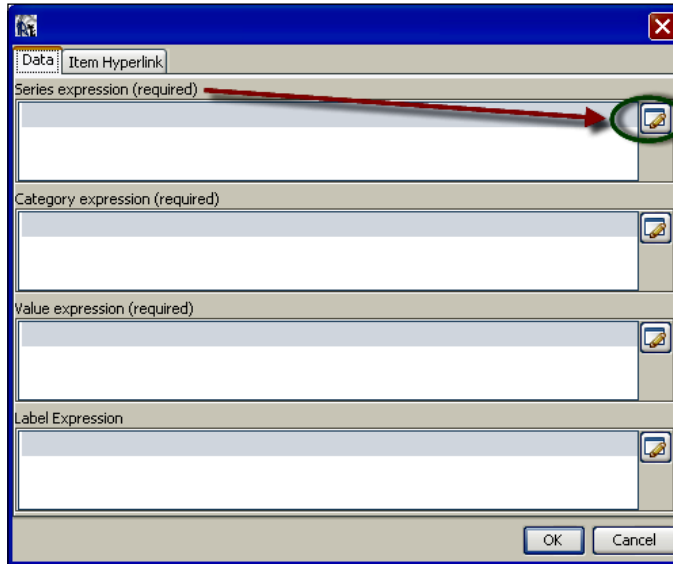
- Right-click on the **Chart** component again; a pop-up menu will appear. Select the **Chart Data** option from the pop-up menu, as shown in the following screenshot:



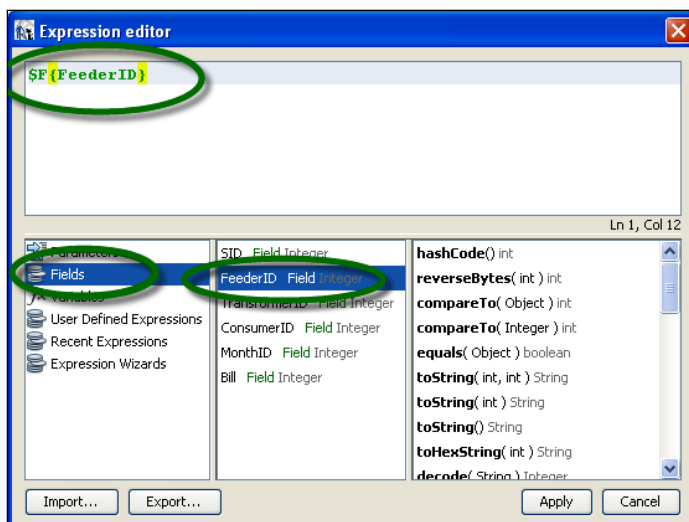
- A **Chart details** dialog will appear, containing **Dataset** and **Details** tabs. Switch to the **Details** tab in the **Chart details** dialog and click the **Add** button, as shown in the following screenshot:



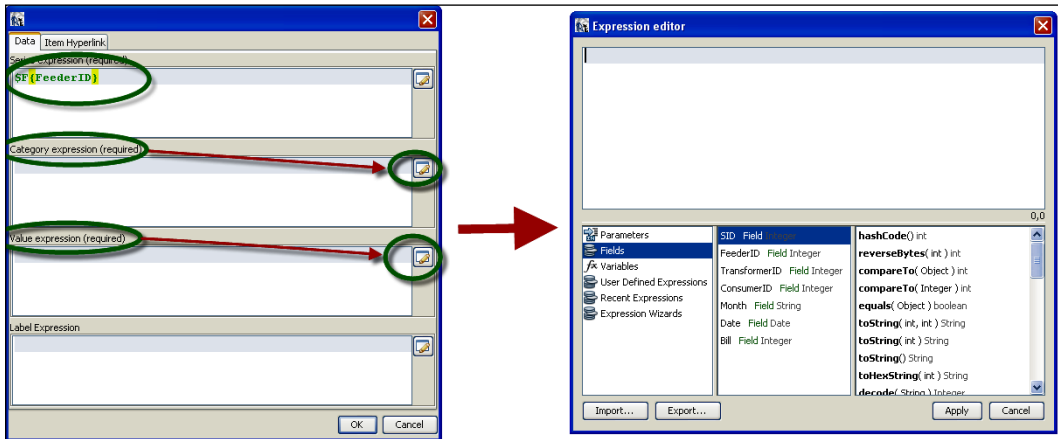
8. A dialog will appear, which will allow you to enter various expressions (**Series expression**, **Category expression**, **Value expression**, and **Label Expression**). In order to add a **Series expression**, click the button beside it, as shown in the following screenshot:



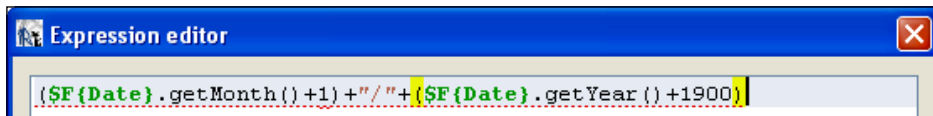
9. An **Expression editor** dialog will appear. Click on the **Fields** option in the first column of the **Expression editor** and double-click on **FeederID Field** in the second column. A `$F{FeederID}` expression will appear in the upper portion of the dialog, as shown next. Click the **Apply** button, and the **Expression editor** dialog will disappear.



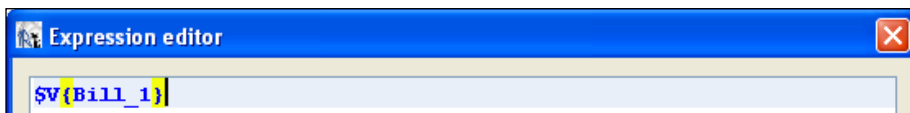
10. Notice that the `$F{FeederID}` expression will appear under **Series expression**. Now click the button beside **Category expression**, as shown next. Another **Expression editor** dialog will appear.



11. Type a `($F{Date}.getMonth()+1)+" / "+($F{Date}.getYear()+1900)` expression in the upper portion of the **Expression editor** dialog, as shown below. Click the **Apply** button; the **Expression editor** dialog will disappear and you will be back to the previous dialog. You will notice the `$F{Date}.getMonth()+"/"+$F{Date}.getYear()` expression appearing under **Category expression**.



12. Now click the button beside **Value expression**. Another **Expression editor** dialog will appear. Click on the **Variables** option in the first column of the **Expression editor** and double-click on **Bill_1 Variable** in the second column. The `$V{Bill_1}` expression will appear in the upper portion of the dialog, as shown below. Click the **Apply** button, the **Expression editor** dialog will disappear and you will be back to the previous dialog.



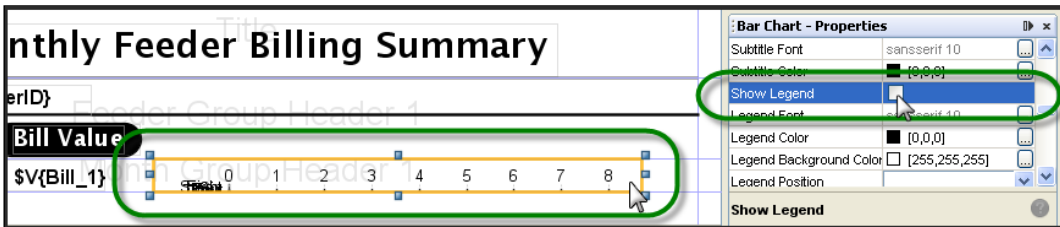
13. Now this dialog will look as shown next. Click the **OK** button; the dialog will disappear. Now you will be back to the **Chart details** dialog of step 6.

The dialog box has a blue title bar with a close button. It contains four text input fields with labels: 'Series expression (required)', 'Category expression (required)', 'Value expression (required)', and 'Label Expression'. Each field has a small icon on the right. The 'OK' button at the bottom right is circled in green.

14. Click the **Close** button at the bottom of the **Chart details** dialog. This dialog will disappear. Switch to the **Preview** tab. You will see a preview of your report as shown below. You will notice that instead of a bar graph, only legends are appearing against each record. The legends are quite meaningless.

Monthly Feeder Billing Summary			
Feeder ID: 1001			
Month ID	Bill Value		
Jan-10	740390	0.0	
Feb-10	165200	<div><div>1001</div><div>1002</div><div>1003</div><div>1004</div><div>1005</div><div>1006</div><div>1007</div></div>	
Mar-10	181650	<div><div>1001</div><div>1002</div><div>1003</div><div>1004</div><div>1005</div><div>1006</div><div>1007</div></div>	
Apr-10	313250	<div><div>1001</div><div>1002</div><div>1003</div><div>1004</div><div>1005</div><div>1006</div><div>1007</div></div>	
May-10	100730	<div><div>1001</div><div>1002</div><div>1003</div><div>1004</div><div>1005</div><div>1006</div><div>1007</div></div>	
Feeder ID: 1002			

15. Switch back to the **Designer** tab. While the **Chart** component is selected, switch to the Properties window below the **Palette**. Find the **Show Legend** property and uncheck the checkbox corresponding to it, as shown next.



16. Switch to the **Preview** tab. You will notice that the legends have gone but the bar graph is not yet appearing in the preview. Although, some odd data is appearing against each record.

Monthly Feeder Billing Summary		
Feeder ID: 1001		
Month ID	Bill Value	
Jan-10	740390	0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Feb-10	165200	0 250,000 500,000 750,000
Mar-10	181650	0 250,000 500,000 750,000
Apr-10	313250	0 250,000 500,000 750,000
May-10	100730	0 250,000 500,000 750,000
Feeder ID: 1002		

17. Switch back to the **Designer** tab. While the **Chart** component is selected, switch to the **Properties** window below the **Palette**. Find the **Show Labels**, the **Show Tick Marks**, and the **Show Tick Labels** properties and uncheck the checkbox corresponding to them.



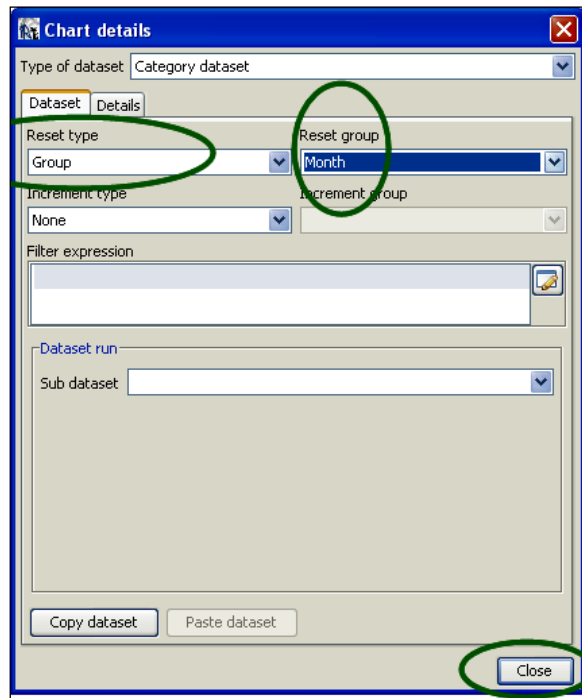
18. Switch to the **Preview** tab. You will notice this time that a horizontal graph is appearing against each record. However, instead of representing a single value, the graph consists of many colored lines whose meaning is difficult to understand.

Monthly Feeder Billing Summary		
Feeder ID: 1001		
Month ID	Bill Value	
Jan-10	740390	
Feb-10	165200	
Mar-10	181650	
Apr-10	313250	
May-10	100730	
Feeder ID: 1002		

19. Switch back to the **Designer** tab. While the **Chart** component is selected, switch to the **Properties** window below the **Palette**. Find the **Evaluation Time** property and set **Group** as its value. Find the **Evaluation group** property and set **Month** as its value, as shown in the following screenshot:

The screenshot shows the Jaspersoft Designer interface. On the left, a preview of the report titled "Monthly Feeder Billing Summary" is visible, showing a table with columns "Month ID" and "Bill Value". The "Month ID" column contains values like "Jan-10", "Feb-10", etc. The "Bill Value" column contains numerical values. A horizontal bar chart is displayed below the table, with each bar representing a month's bill value. The bars are colored in a gradient from blue to red. On the right, the "Bar Chart - Properties" window is open. The "Evaluation Time" property is set to "Group" and the "Evaluation group" property is set to "Month". The "Title Expression" is set to "\$V{Bill_1}". The "Title Font" is set to "sansserif 10". The "Title Color" is set to "[0,0,0]".

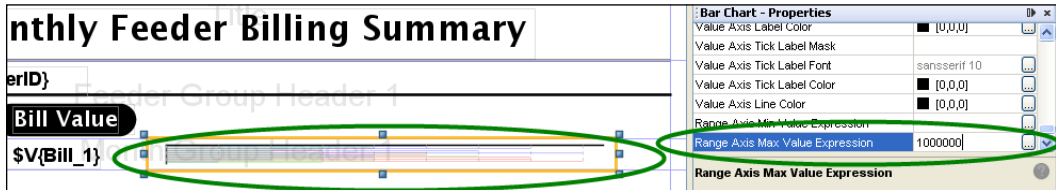
20. Right-click on the **Chart** component; a pop-up menu will appear. Select the **Chart Data** option from the pop-up menu, as shown next. A **Chart details** dialog will appear, containing **Dataset** and **Details** tabs. Expand the **Reset type** combo box from the **Dataset** tab and select the **Group** option. Similarly, expand the **Reset group** combo box and select the **Month** option. Then click the **Close** button, as shown next. The **Chart details** dialog will disappear.



21. Switch to the **Preview** tab. You will notice that a colored horizontal bar is appearing against each record. However, all the bars are of the same length and, therefore, do not tell anything about bill values.

Monthly Feeder Billing Summary		
Feeder ID: 1001		
Month ID	Bill Value	
Jan-10	740390	
Feb-10	165200	
Mar-10	181650	
Apr-10	313250	
May-10	100730	
Feeder ID: 1002		

22. Switch back to the **Designer** tab. While the **Chart** component is selected, switch to the **Properties** window below the **Palette**. Find the **Range Axis Max Value Expression** property and set 1000000 as its value, as shown in the following screenshot:



23. Switch to the **Preview** tab. You will notice this time that a horizontal bar of scaled length is appearing against each record. The length of each bar is now representing the bill value of each record.

Monthly Feeder Billing Summary		
Feeder ID: 1001		
Month ID	Bill Value	
Jan-10	740390	<div></div>
Feb-10	165200	<div></div>
Mar-10	181650	<div></div>
Apr-10	313250	<div></div>
May-10	100730	<div></div>
Feeder ID: 1002		

8

Java Wrappers for your JasperReports

In this chapter, you will learn:

- ▶ Creating a Java wrapper for your report
- ▶ Compiling and viewing your report in a Java Swing application
- ▶ Printing the hardcopy of your report using a Java Swing application
- ▶ Creating an Excel report from a Java Swing application
- ▶ Creating a JasperReport on the fly in a Java web application

Introduction

This chapter is about developing Java applications that use the functionality of JasperReports. The recipes in this chapter demonstrate how you can integrate JasperReports as a reporting tool directly into your Java applications.

The recipes of this chapter teach you the following aspects of integrating JasperReports:

- ▶ Developing a wrapper Java class that can wrap JasperReports and expose its common functionality using a simple interface.
- ▶ Integrating JasperReports into desktop as well as web applications.
- ▶ Demonstrating how to write Java code to compile your JRXML files, how to supply application data to JasperReports, and how to save reports in PDF and Excel format.

Creating a Java wrapper for your report

This recipe teaches you how to design a simple wrapper class in Java that wraps JasperReports functionality to process and compile an existing JRXML file, fetch application data, and combine the processed JRXML with application data to generate and save your JasperReport in PDF format.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb9` and copy sample data for this recipe into the database.

You will need **Java Development Kit (JDK)** version 1.6.X to compile and run the Java application of this recipe. You will find all the necessary downloads and instructions on Sun's official website. After installing JDK, you will also set the `JAVA_HOME` environment variable to point to the main folder of your JDK installation (`.. \jdk1.6.X`).

You will be using a JRXML file named `MonthlyCustomerReport.jrxml` and two Java files named `ConnectionManager.java` and `JasperReportsWrapper.java` in this recipe. You will find these three files in the `Task1` folder of the source code download of this chapter. In addition, you will also find two `.bat` files named `compile.bat` and `run.bat` and two folders named `com` and `lib` in the same `Task1` folder. Copy all five files along with the `com` and `lib` folders (with their contents) to the `C:\JasperReportsCookBookSamples\` folder on your PC.

In order to compile `JasperReportsWrapper.java`, you will need the following seven JAR files:

- ▶ `jasperreports-3.6.0.jar`
- ▶ `commons-beanutils-1.8.0.jar`
- ▶ `commons-collections-3.2.1.jar`
- ▶ `commons-digester-1.7.jar`
- ▶ `commons-logging-1.1.jar`
- ▶ `groovy-all-1.5.5.jar`
- ▶ `iText-2.1.0.jar`

You will find all seven JAR files in the `.. \Jaspersoft\iReport-nb-3.X\ireport\modules\ext` folder of your iReport installation.

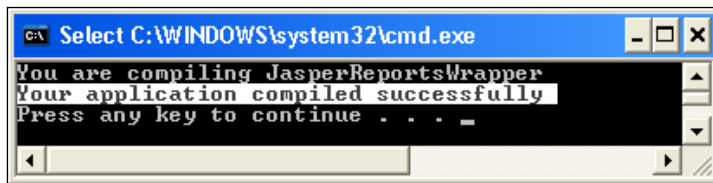
Although your `JasperReportsWrapper.java` will compile with these seven JAR files, your complete application will not run without the JDBC JAR file that contains the driver for PostgreSQL. The name of the JAR file is `postgresql-8.2-506.jdbc3.jar`. You can download the PostgreSQL JDBC driver from <http://jdbc.postgresql.org/download.html>. You can also find this `postgresql-8.2-506.jdbc3.jar` file in the `.. \Jaspersoft\iReport-nb-3.X\ide8\modules\ext` folder of your iReport installation.

Once you have all the JAR files ready, you will put them in the `C:\JasperReportsCookBookSamples\lib\` folder on your PC. The `compile.bat` and `run.bat` files assume that all JAR files are located in the `C:\JasperReportsCookBookSamples\lib\` folder.

How to do it...

The following steps demonstrate how to incrementally design a wrapper Java class to process your JasperReport and generate its PDF.

1. Browse to the `C:\JasperReportsCookBookSamples` folder on your PC into which you copied files from the `Task1` folder from the source code for this chapter.
2. Double-click the `compile.bat` file, and a command prompt will appear. It will compile your `JasperReportsWrapper.java` file and show the compilation results. After successful compilation, dismiss the command prompt by clicking the **Close** button on its top-right corner. Now you will see that a `JasperReportsWrapper.class` file will appear in the `C:\JasperReportsCookBookSamples\com` folder.



3. Now double-click the `run.bat` file to execute the compiled `JasperReportsWrapper` class. You will see that your Java application displays a **Connection with database established successfully** message on command prompt. Dismiss the command prompt by clicking the **Close** button.



4. Open the `JasperReportsWrapper.java` file from the `C:\JasperReportsCookBookSamples` folder on your PC in a text editor. The file contains code for your `JasperReportsWrapper` class. You will find five methods named `connect2DB()`, `compileJRXMLFile()`, `fillReport()`, `saveReportInPDF()`, and `main()` in the `JasperReportsWrapper` class. The first method named `connect2DB()` is ready to be used in the recipe. The `connect2DB()` method connects to your PostgreSQL database and returns the connection in the form of a `Connection` object. You will incrementally build the rest of the four methods in this recipe to compile, fill, and save your `JasperReport`.
5. Now you will use the JasperReports API to compile your `MonthlyCustomerInvoices.jrxml` file. For this purpose, type the following code in the try block of the `compileJRXMLFile()` method in the `JasperReportsWrapper.java` file.

```
JasperReport jr =  
    JasperCompileManager.compileReport(jasperXMLFileName);  
if (jr != null)  
    System.out.println ("Your report compiled successfully");  
return jr;
```

After you have typed the code, the complete `compileJRXMLFile()` method will look as follows:

```
// Compile JasperReports XML file  
public JasperReport compileJRXMLFile(String jasperXMLFileName)  
{  
    try  
    {  
        JasperReport jr =  
            JasperCompileManager.compileReport(jasperXMLFileName);  
        if (jr != null)  
            System.out.println ("Your report compiled successfully");  
        return jr;  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    return null;  
}
```

Save your `JasperReportsWrapper` class.

6. Now add a call to the `compileJRXMLFile()` method at the end of the existing code in the `main()` method of your `JasperReportsWrapper` class. The call to the `compileJRXMLFile()` file will look as follows:

```
JasperReport compiledJasperReport =  
    wrapper.compileJRXMLFile(path2JRXMLFile);
```

After adding the call, the `main()` method will become:

```
public static void main (String[] args)
{
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    Connection connection = wrapper.connect2DB(
        dbServerAdd, dbServerPort, dbName, dbUser, dbPass);
    // Compile JRXML file
    JasperReport compiledJasperReport =
        wrapper.compileJRXMLFile(path2JRXMLFile);
}
```

Save your `JasperReportsWrapper` class.

7. Double-click the `compile.bat` file to compile the `JasperReportsWrapper` class. A command prompt will appear. It will compile your application. After successful compilation, dismiss the command prompt by clicking on the **Close** button.
8. Double-click the `run.bat` file to execute the compiled `JasperReportsWrapper.class` file. You will see that your Java application displays **Connection with database established successfully** and **Your report compiled successfully** messages. Dismiss the command prompt by clicking the **Close** button.
9. Next you will fill your compiled `JasperReport` with your application data. You will use the `Connection` object that you saw earlier in step 4 to connect to your PostgreSQL database. Recall from step 4 that `connect2DB()` method returned this `Connection` object. You will pass the `Connection` object to the `FillManager` class of `JasperReports` by typing the following code in the `try` block of the `fillReport()` method of the `JasperReportsWrapper` class:

```
if (connection != null)
{
    JasperPrint jp =
        JasperFillManager.fillReport(jr, params, con);
    if (jp != null)
        System.out.println ("
            Your report filled with application data");
    return jp;
}
```

After typing the code, the `fillReport()` method will look as follows:

```
// Fill data in compiled JasperReports file
public JasperPrint fillReport (
    JasperReport jr, Map params, Connection con)
{
    try
    {
```

```
        if (connection != null)
        {
            JasperPrint jp =
                JasperFillManager.fillReport(jr, params, con);
            if (jp != null)
                System.out.println ("
                    Your report filled with application data");
            return jp;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
```

Save your JasperReportsWrapper class.

10. Now add a call to the `fillReport()` method at the end of the existing code in the `main()` method of your `JasperReportsWrapper` class, as shown below:

```
JasperPrint jasperPrint =
    wrapper.fillReport(compiledJasperReport, null, connection);
```

After adding the call, the `main()` method will become:

```
public static void main(String[] args)
{
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    Connection connection = wrapper.connect2DB(
        dbServerAdd, dbServerPort, dbName, dbUser, dbPass);
    // Compile JRXML file
    JasperReport compiledJasperReport =
        wrapper.compileJRXMLFile(path2JRXMLFile);
    // Fill data in compiled JRXML file
    JasperPrint jasperPrint = wrapper.fillReport(
        compiledJasperReport, null, connection);
}
```

Save your `JasperReportsWrapper` class.

11. Double-click the `compile.bat` file to compile the `JasperReportsWrapper` class. A command prompt will appear that will compile your Java file. After successful compilation, dismiss the command prompt by pressing the **Close** button.

12. Now double-click the `run.bat` file to execute the compiled `JasperReportsWrapper.class` file. You will see that your Java application displays **Connection with database established successfully, Your report compiled successfully**, and **Your report filled with application data** messages. Click the **Close** button to dismiss the command prompt.
13. Next you will add code to save your report in PDF format. For this purpose, you will fill the `saveReportInPDF()` method of the `JasperReportsWrapper` class with Java code in its `try` block as follows:

```
JasperExportManager.exportReportToPdfFile(
    jasperPrint, pdfFileName);
System.out.println ("Your report successfully saved in PDF");
```

After typing the code, the `saveReportInPDF()` method will look like as follows:

```
// Compile, fill and save report in JasperReport
public void saveReportInPDF (JasperPrint jasperPrint,
    String pdfFileName)
{
    try
    {
        JasperExportManager.exportReportToPdfFile(
            jasperPrint, pdfFileName);
        System.out.println ("
            Your report successfully saved in PDF");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Save your `JasperReportsWrapper` class.

14. Now add a call to the `saveReportInPDF()` at the end of the existing code in the `main()` method of your `JasperReportsWrapper` class as follows:

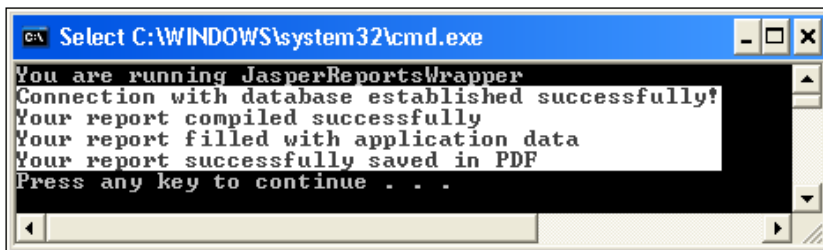
```
wrapper.saveReportInPDF(jasperPrint, pdfFileName);
```

After adding the call, the `main()` method will become:

```
public static void main(String[] args)
{
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    Connection connection = wrapper.connect2DB(
        dbServerAdd, dbServerPort, dbName, dbUser, dbPass);
    // Compile JRXML file
    JasperReport compiledJasperReport= wrapper.compileJRXMLFile();
    // Fill data in compiled jasper file
    JasperPrint jasperPrint = wrapper.fillReport(
```

```
        compiledJasperReport, null, connection);  
    // Save compiled and filled Jasper report in PDF  
    wrapper.saveReportInPDF(jasperPrint, pdfFileName);  
}
```

15. Double-click the `compile.bat` file to compile the `JasperReportsWrapper` class. A command prompt will appear that will compile your Java file. After successful compilation, click the **Close** button to dismiss the command prompt.
16. Now double-click the `run.bat` file to execute the compiled `JasperReportsWrapper.class` file. You will see that your Java application displays **Connection with database established successfully, Your report compiled successfully, Your report filled with application data, and Your report successfully saved in PDF** messages. Click the **Close** button to dismiss the command prompt. Now you can browse to the `C:\JasperReportsCookBookSamples\` folder on your PC, where you will find a `MonthlyCustomerInvoices.pdf` file. This is the JasperReport, which your `JasperReportsWrapper` Java application has generated and saved for you.



How it works...

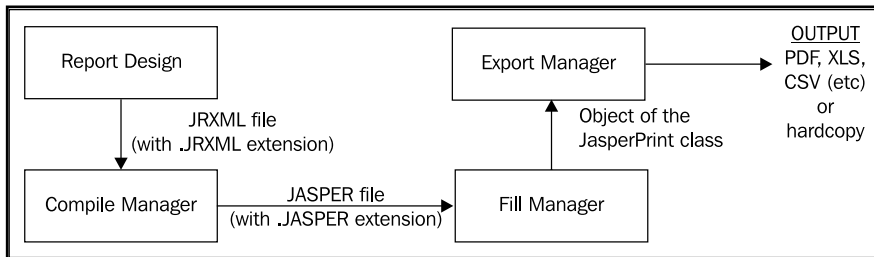
Your Java application has compiled, generated, and saved your JasperReport in this recipe. In order to do all this, you used a wrapper class named `JasperReportsWrapper`, which wraps the functionality of JasperReports.

When you started, the `JasperReportsWrapper` class only had the functionality of connecting to a database. You incrementally built it to provide functionality to fetch application data, compile a JRXML file, supply application data to the compiled JRXML file, generate your JasperReport, and finally save it as a PDF file.

In step 5 of the recipe, you used a manager class of JasperReports named `JasperCompileManager` to compile your JRXML file. Then, in step 9, you used another manager class named `JasperFillManager` to fill the compiled report with application data. Note that the `JasperFillManager` class takes both a compiled JasperReport and a database `Connection` object. It internally fetches application data and provides the data to the compiled report to generate your actual JasperReport.

Finally, in step 13, you used another manager class named `JasperExportManager` to export your filled `JasperReport` into a PDF file, which you could browse and find in the `C:\JasperReportsCookBookSamples\` folder on your PC.

The following diagram shows the steps that are followed to generate a report:



You can see four steps in the diagram; each step is represented by a box:

1. **Report Design:** This step produces a JRXML file. You will normally design your report using iReport, as you have learned in the recipes of the first eight chapters of this cookbook.
2. **Compile Manager:** The compile manager compiles the JRXML file into a JASPER file. You used the compile manager in step 5 of this recipe.
3. **Fill Manager:** It takes a JASPER file and combines with actual report data to produce your report. You used Jasper's fill manager in step 9.
4. **Export Manager:** Finally the last box named Export Manager takes your report and generates a PDF file for the report. This is what you did in step 13.

Compiling and viewing your report in a Java Swing application

This recipe teaches you how to embed JasperReports functionality in a Java swing application.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb9` and copy sample data for this recipe into the database.

You will need Java Development Kit (JDK) version 1.6.X to compile and run the Java application of this recipe. You will find all the necessary downloads and instructions on Sun's official website . After installing JDK, you will also set the `JAVA_HOME` environment variable to point to the main folder of your JDK installation (`.. \jdk1.6.X`).

You will be using a JRXML file named `MonthlyCustomerReport.jrxml` and three Java files named `ConnectionManager.java`, `JasperReportsWrapper.java`, and `JasperReportsViewer.java` in this recipe. You will find these three files in the `Task2` folder of the source code download for this chapter. In addition, you will also find two `.bat` files named `compile.bat` and `run.bat`, along with two folders named `com` and `lib` in the same `Task2` folder. Copy all five files and the `com` and `lib` folders (along with their contents) to the `C:\JasperReportsCookBookSamples\` folder on your PC.

In order to compile `JasperReportsViewer.java`, you will need the following seven JAR files:

- ▶ `jasperreports-3.6.0.jar`
- ▶ `commons-beanutils-1.8.0.jar`
- ▶ `commons-collections-3.2.1.jar`
- ▶ `commons-digester-1.7.jar`
- ▶ `commons-logging-1.1.jar`
- ▶ `groovy-all-1.5.5.jar`
- ▶ `iText-2.1.0.jar`

You will find all seven JAR files in the `.. \Jaspersoft\iReport-nb-3.X\ireport\modules\ext` folder of your iReport installation.

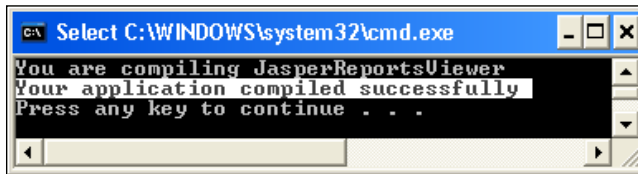
Although your `JasperReportsViewer.java` will compile with these seven JAR files, your complete application will not run without the JDBC JAR file that contains the driver for PostgreSQL. The name of the JAR file is `postgresql-8.2-506.jdbc3.jar`. You can download PostgreSQL JDBC driver from <http://jdbc.postgresql.org/download.html>. You can also find this `postgresql-8.2-506.jdbc3.jar` file in the `.. \Jaspersoft\iReport-nb-3.X\ide8\modules\ext` folder of your iReport installation.

Once you have all the JAR files ready, you will put them in the `C:\JasperReportsCookBookSamples\lib\` folder on your PC. The `compile.bat` and `run.bat` files assume that all JAR files are located in the `C:\JasperReportsCookBookSamples\lib\` folder.

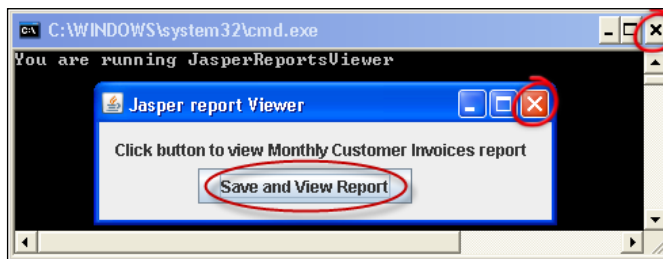
How to do it...

The following steps show you how to work with JasperReports in a Java Swing application:

1. Browse to the `C:\JasperReportsCookBookSamples` folder on your PC into which you copied files from the `Task2` folder of the source code for this chapter.
2. Double-click the `compile.bat` file. A command prompt will appear. It will compile `JasperReportsViewer.java` and show the compilation results. After successful compilation, dismiss the command prompt by clicking the **Close** button on its top-right corner. Now you will see that a `JasperReportsViewer.class` file will appear in the `C:\JasperReportsCookBookSamples\com` folder.



3. Now double-click the `run.bat` file to execute the compiled `JasperReportsViewer.class` file. A command prompt will appear, which will later show a GUI window containing a **Save and View Report** button. When you click the **Save and View Report** button, nothing will happen. Click the **Close** button on the top-right corner of GUI as well as the command prompt to dismiss both.



4. Open the `JasperReportsViewer.java` file from the `C:\JasperReportsCookBookSamples` folder on your PC in a text editor. The `JasperReportsViewer.java` file contains the code for a class named `JasperReportsViewer`. You will find three methods named `initComponents()`, `jButtonActionPerformed()`, and `viewReport()` in the `JasperReportsViewer` class. The `initComponents()` method initializes all the required components and the `jButtonActionPerformed()` method handles the action when the **Save and View Report** button is clicked. In this recipe you will add code to the third method named `viewReport()`. This method will display your `JasperReport` in your Swing application.

5. Edit `JasperReportsViewer.java` by typing following code in the `viewReport()` method:

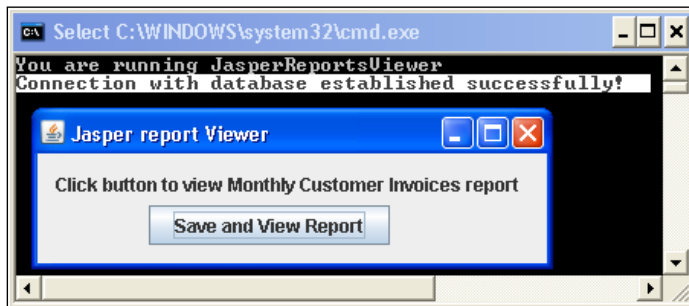
```
JasperReportsWrapper wrapper = new JasperReportsWrapper();  
  
    Connection connection =  
        wrapper.connect2DB(wrapper.dbServerAdd,  
            wrapper.dbServerPort, wrapper.dbName,  
            wrapper.dbUser, wrapper.dbPass);
```

After adding the code, the `viewReport()` method will become:

```
public void viewReport ()  
{  
    // Create wrapper instance and connect to DB  
    JasperReportsWrapper wrapper = new  
        JasperReportsWrapper();  
    Connection connection =  
        wrapper.connect2DB(wrapper.dbServerAdd,  
            wrapper.dbServerPort, wrapper.dbName,  
            wrapper.dbUser, wrapper.dbPass);  
}
```

Save the `JasperReportsViewer` class.

6. Double-click the `compile.bat` file to compile the `JasperReportsViewer` class. A command prompt will appear. It will compile `JasperReportsViewer.java` and show the compilation results. Click the **Close** button to dismiss the command prompt.
7. Now double-click the `run.bat` file to execute your Java application. A command prompt will appear, which will show a GUI window containing a **Save and View Report** button. Click the **Save and View Report** button. You will see a **Connection with database established successfully** message on command prompt. Dismiss the GUI as well as the command prompt by clicking the **Close** button.



8. Open the `JasperReportsViewer.java` file again and type the following code at the end to the `viewReport()` method as follows:

```
JasperPrint jasperPrint =
    wrapper.fillReport(wrapper.compileJRXMLFile (
        wrapper.path2JRXMLFile), null, connection);
wrapper.saveReportInPDF(jasperPrint, wrapper.pdfFileName);
```

After adding the code, the `viewReport()` method will become:

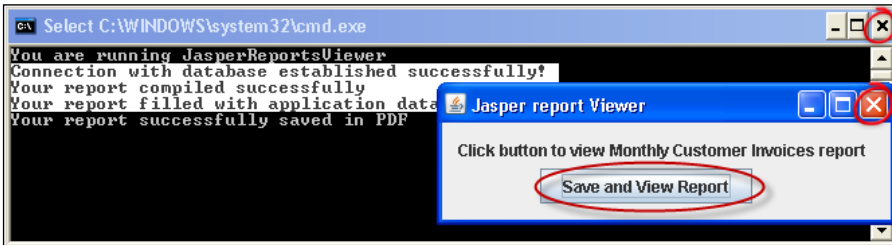
```
public void viewReport ()
{
    // Create wrapper instance and connect to DB
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    Connection connection =
        wrapper.connect2DB(wrapper.dbServerAdd,
            wrapper.dbServerPort, wrapper.dbName,
            wrapper.dbUser, wrapper.dbPass);

    // Compile and fill Jasper report with application data
    JasperPrint jasperPrint =
        wrapper.fillReport(wrapper.compileJRXMLFile
            (wrapper.path2JRXMLFile), null, connection);

    // Save report in PDF
    wrapper.saveReportInPDF(jasperPrint, wrapper.pdfFileName);
}
```

Save the `JasperReportsViewer` class.

9. Double-click the `compile.bat` file to compile the `JasperReportsViewer` class. A command prompt will appear. It will compile `JasperReportsViewer.java` and show the compilation results. Click the **Close** button to dismiss the command prompt.
10. Double-click the `run.bat` file to execute your Java application. A command prompt will appear, which will show a GUI window containing a **Save and View Report** button. Click the **Save and View Report** button. You will see **Your report filled with application data** and **Your report successfully saved in PDF** messages on command prompt. Dismiss the GUI as well as the command prompt by clicking the **Close** button. Now you can browse to the `C:\JasperReportsCookBookSamples\` folder on your PC, where you will find a `MonthlyCustomerInvoices.pdf` file. This is the JasperReport, which your `JasperReportsViewer` Swing application has generated and saved for you.



11. Now you have your JasperReport which your Java application has generated and saved for you. Next you will add code to view your JasperReport in a PDF viewer utility named `JasperViewer`, which is part of JasperReports. Type the following code at the end of the `viewReport()` method:

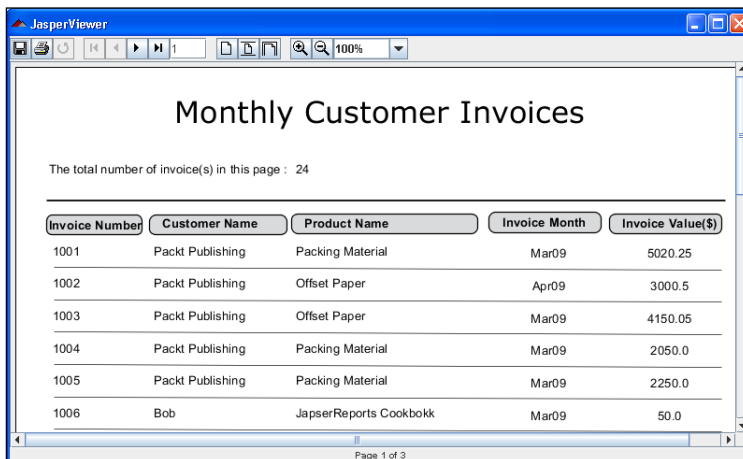
```
JasperViewer.viewReport(jasperPrint);
```

After adding the code, the `viewReport()` method will become:

```
public void viewReport ()
{
    // Create wrapper instance and connect to DB
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    wrapper.connect2DB(
        dbServerAdd, dbServerPort, dbName, dbUser, dbPass);
    // Compile and fill Jasper report with application data
    JasperPrint jasperPrint =
        wrapper.fillReport(wrapper.compileJRXMLEFile
            (wrapper.path2JRXMLEFile), null, connection);
    // Save report in PDF
    wrapper.saveReportInPDF(jasperPrint, wrapper.pdfFileName);
    // View PDF of your Jasper report
    JasperViewer.viewReport(jasperPrint);
}
```

Save the `JasperReportsViewer` class.

12. Double-click the `compile.bat` file to compile the `JasperReportsViewer` class. A command prompt will appear. It will compile `JasperReportsViewer.java` and show the compilation results. Click on the **Close** button to dismiss it.
13. Double-click the `run.bat`, it will execute your compiled Java application to show the same GUI window that you saw earlier in step 3. Click the **Save and View Report** button. A window will be displayed to show the JasperReport, as shown below:



How it works...

In this recipe, you have embedded JasperReports functionality inside a Swing application. You have used the same wrapper class named `JasperReportsWrapper` that you used in the previous recipe *Creating a Java wrapper for your report*. The `JasperReportsWrapper` class provides all the functionality to fetch application data, compile a JRXML file, supply application data to the compiled JRXML file, generate your JasperReport, and finally save it as a PDF file.

Refer to the *How it works* section of the *Creating a Java wrapper for your report* recipe to learn how the wrapper class works.

You also used a utility class named `JasperViewer` in step 11 of the recipe. The `JasperViewer` class provides a useful static method named `viewReport()`, which simply displays a JasperReport inside a Java container. The `viewReport()` method is simple to use. You just call the method, pass a report (already filled with data), and the method will internally do everything to display the report.

Printing the hardcopy of your report using a Java Swing application

This recipe teaches you how to use JasperReports to print a hardcopy of your Jasper report from within a Java Swing application.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb9` and copy sample data for this recipe into the database.

You will need Java Development Kit (JDK) version 1.6.X to compile and run the Java Swing application of this recipe. You will find all the necessary downloads and instructions on Sun's official website. After installing JDK, you will also set the `JAVA_HOME` environment variable to point to the main folder of your JDK installation (`.. \jdk1.6.X`).

You will be using a JRXML file named `MonthlyCustomerReport.jrxml` and three Java files named `ConnectionManager.java`, `JasperReportsWrapper.java`, and `JasperReportsPrinter.java` in this recipe. You will find these three files in the `Task3` folder of the source code download for this chapter. In addition, you will also find two `.bat` files named `compile.bat` and `run.bat`, along with two folders named `com` and `lib` in the same `Task3` folder. Copy all five files and the `com` and `lib` folders (along with their contents) to the `C:\JasperReportsCookBookSamples\` folder on your PC.

In order to compile `JasperReportsPrinter.java`, you will need the following seven JAR files:

- ▶ `jasperreports-3.6.0.jar`
- ▶ `commons-beanutils-1.8.0.jar`
- ▶ `commons-collections-3.2.1.jar`
- ▶ `commons-digester-1.7.jar`
- ▶ `commons-logging-1.1.jar`
- ▶ `groovy-all-1.5.5.jar`

Your `JasperReportsPrinter.java` will compile with these seven JAR files, but your complete application will not run without the JDBC JAR file that contains the driver for PostgreSQL. The name of the JAR file is `postgresql-8.2-506.jdbc3.jar`. You can download PostgreSQL JDBC driver from <http://jdbc.postgresql.org/download.html>. You can also find this `postgresql-8.2-506.jdbc3.jar` file in the `...\Jaspersoft\iReport-nb-3.X\ide8\modules\ext` folder of your iReport installation.

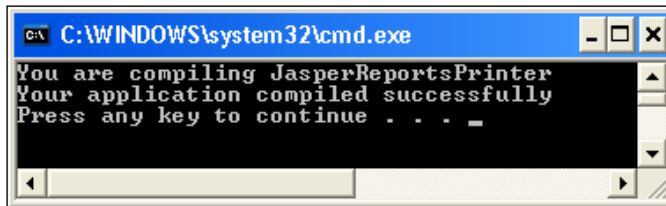
Once you have all the JAR files ready, you will put them in the `C:\JasperReportsCookBookSamples\lib\` folder on your PC. The `compile.bat` and `run.bat` files assume that all JAR files are located in `C:\JasperReportsCookBookSamples\lib\` folder.

How to do it...

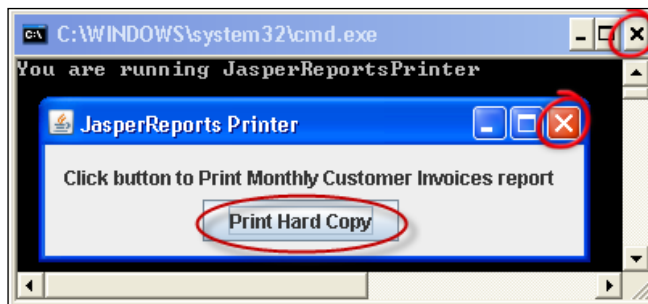
Let's learn how to insert printing capability into a Java Swing application for JasperReports.

1. Browse to the `C:\JasperReportsCookBookSamples` folder on your PC into which you copied files from the `Task3` folder of the source code for this chapter.

2. Double-click the `compile.bat` file. A command prompt will appear. It will compile `JasperReportsPrinter.java` and show the compilation results. After successful compilation, dismiss the command prompt by clicking the **Close** button on its top-right corner. Now you will see that a `JasperReportsPrinter.class` file will appear in the `C:\JasperReportsCookBookSamples\com` folder.



3. Now double-click the `run.bat` file to execute the compiled `JasperReportsPrinter.class` file. A command prompt will appear, which will later show a GUI window containing a **Print Hard Copy** button. When you click the **Print Hard Copy** button, you will see **Your report compile successfully** and **Your report filled with application data** messages on the command prompt. Dismiss the command prompt by clicking the **Close** button.



4. Open the `JasperReportsPrinter.java` file from the `C:\JasperReportsCookBookSamples` folder on your PC in a text editor. The `JasperReportsPrinter.java` file contains the code for a class named `JasperReportsPrinter`. You will find three methods named `initComponents()`, `jButtonActionPerformed()`, and `printHardCopy()` in the `JasperReportsPrinter` class. The `initComponents()` method initializes all the required components and the `jButtonActionPerformed()` method handles the action when the **Print Hard Copy** button is clicked. You are about to write code for the third method named `printHardCopy()`. This method will print a hard copy of your `JasperReport`.

5. Edit `JasperReportsPrinter.java` by typing the following code in the `printHardCopy()` method:

```
// Print Jasper report
if (jasperPrint != null)
{
    try {
        JasperPrintManager.printReport(jasperPrint, true);
    } catch (JRException ex) {
        ex.printStackTrace();
    }
}
else
    System.out.println ("Report printing failed");
```

After adding the code, the `printReport()` method will become:

```
public void printHardCopy()
{
    // Create wrapper instance and connect to DB
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    Connection connection = wrapper.connect2DB(
        wrapper.dbServerAdd, wrapper.dbServerPort, wrapper.dbName,
        wrapper.dbUser, wrapper.dbPass);
    // Compile and fill Jasper report with application data
    JasperPrint jasperPrint = wrapper.fillReport(
        wrapper.compileJRXMLFile (
            wrapper.path2JRXMLFile), null, connection);

    // Print Jasper report
    if (jasperPrint != null)
    {
        try {
            JasperPrintManager.printReport(jasperPrint, true);
        } catch (JRException ex) {
            ex.printStackTrace();
        }
    }
    else
        System.out.println ("Report printing failed");
}
```

Save the `JasperReportsPrinter` class.

6. Double-click the `compile.bat` file to compile the `JasperReportsPrinter` class. A command prompt will appear. It will compile `JasperReportsPrinter.java` and show the compilation results. Click the **Close** button to dismiss it.

7. Double-click the `run.bat` file. It will execute your compiled Java application to show the same GUI window that you saw earlier in step 3. Click the **Print Hard Copy** button, and a **Print** dialog will appear. Click the **OK** button. Your JasperReport will be sent to printer for printing. Click the **Close** button on the top-right corner of the GUI as well as the command prompt to dismiss both.

Creating an Excel report from a Java Swing application

This recipe teaches you how to use JasperReports functionality in your Java application to generate a report in Excel format.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb9` and copy sample data for this recipe into the database.

You will need Java Development Kit (JDK) version 1.6.X to compile and run the Java Swing application of this recipe. You will find all the necessary downloads and instructions on Sun's official website. After installing JDK, you will also set the `JAVA_HOME` environment variable to point to the main folder of your JDK installation (`.. \jdk1.6.X`).

You will be using a JXML file named `MonthlyCustomerReport.jrxml` and three Java files named `ConnectionManager.java`, `JasperReportsWrapper.java`, and `JasperReportsXLSExporter.java` in this recipe. You will find these three files in the `Task4` folder of the source code download for this chapter. In addition, you will also find two `.bat` files named `compile.bat` and `run.bat`, along with two folders named `com` and `lib` in the same `Task4` folder. Copy all five files and the `com` and `lib` folders (along with their contents) to the `C:\JasperReportsCookBookSamples\` folder on your PC.

In order to compile `JasperReportsXLSExporter.java`, you will need the following seven JAR files:

- ▶ `jasperreports-3.6.0.jar`
- ▶ `commons-beanutils-1.8.0.jar`
- ▶ `commons-collections-3.2.1.jar`
- ▶ `commons-digester-1.7.jar`
- ▶ `commons-logging-1.1.jar`
- ▶ `groovy-all-1.5.5.jar`
- ▶ `poi-3.2-FINAL-20081019.jar`

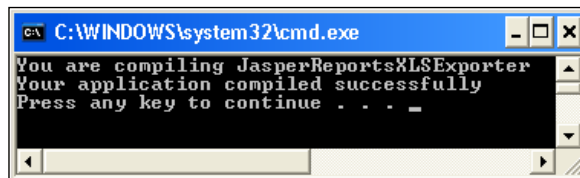
Although your `JasperReportsXLSExporter.java` will compile with these seven JAR files, your complete application will not run without the JDBC JAR file that contains the driver for PostgreSQL. The name of the JAR file is `postgresql-8.2-506.jdbc3.jar`. You can download PostgreSQL JDBC driver from <http://jdbc.postgresql.org/download.html>. You can also find this `postgresql-8.2-506.jdbc3.jar` file in the `...\Jaspersoft\iReport-nb-3.X\ide8\modules\ext` folder of your iReport installation.

Once you have all the JAR files ready, you will put them in the `C:\JasperReportsCookBookSamples\lib\` folder on your PC. The `compile.bat` and `run.bat` files assume that all JAR files are located in the `C:\JasperReportsCookBookSamples\lib\` folder.

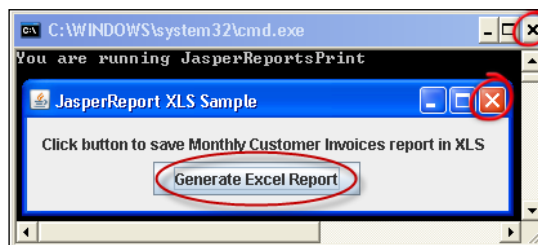
How to do it...

The following steps teach you how to generate an XLS spreadsheet from your report:

1. Browse to the `C:\JasperReportsCookBookSamples` folder on your PC into which you copied files from the `Task4` folder of the source code for this chapter.
2. Double-click the `compile.bat` file, and a command prompt will appear. It will compile `JasperReportsXLSExporter.java` and show the compilation results. After successful compilation, dismiss the command prompt by clicking the **Close** button on its top-right corner. Now you will see that a `JasperReportsXLSExporter.class` file will appear in the `C:\JasperReportsCookBookSamples\com` folder.



3. Now double-click the `run.bat` file to execute the compiled `JasperReportsXLSExporter.class` file. A command prompt will appear, which will later show a GUI window containing a **Generate Excel Report** button. If you click the **Generate Excel Report** button, nothing will happen. Click the **Exit** button on the top-right corner of the GUI to dismiss it.



4. Open the `JasperReportsXLSExporter.java` file from the `C:\JasperReportsCookBookSamples` folder on your PC in a text editor. The `JasperReportsXLSExporter.java` file contains the code for a class named `JasperReportsXLSExporter.class`. You will find three methods named `initComponents()`, `jButtonActionPerformed()`, and `saveXLSReport()` in the `JasperReportsXLSExporter` class. The `initComponents()` method initializes all the required components and the `jButtonActionPerformed()` method handles the action when the **Generate Excel Report** button is clicked. In this recipe, you will add code to the third method named `saveXLSReport()`. This method will save your JasperReport in XLS format on your PC.
5. Edit `JasperReportsXLSExporter.java` by typing the following code in the `saveXLSReport()` method:

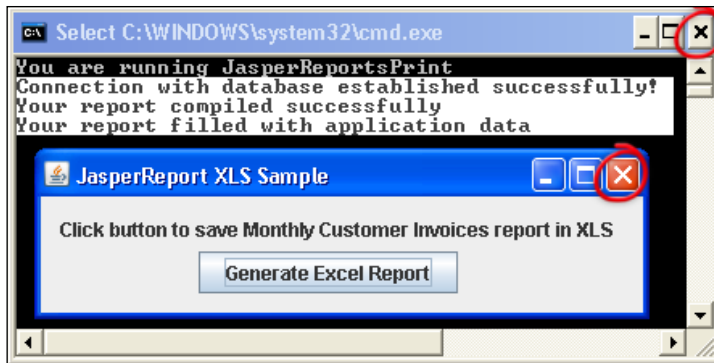
```
String xlsFileName = "MonthlyCustomerInvoices.xls";
// Create wrapper instance and connect to DB
JasperReportsWrapper wrapper = new JasperReportsWrapper();
Connection connection = wrapper.connect2DB(
    wrapper.dbServerAdd, wrapper.dbServerPort,
    wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
// Compile and fill Jasper report with application data
JasperPrint jasperPrint = wrapper.fillReport(
    wrapper.compileJRXMLEFile (wrapper.path2JRXMLEFile),
    null, connection);
```

After adding the code, the `saveXLSReport()` method will become:

```
public void saveXLSReport ()
{
    try
    {
        String xlsFileName = "MonthlyCustomerInvoices.xls";
        // Create wrapper instance and connect to DB
        JasperReportsWrapper wrapper = new JasperReportsWrapper();
        Connection connection = wrapper.connect2DB(
            wrapper.dbServerAdd, wrapper.dbServerPort,
            wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
        // Compile and fill Jasper report with application data
        JasperPrint jasperPrint = wrapper.fillReport(
            wrapper.compileJRXMLEFile (wrapper.path2JRXMLEFile),
            null, connection);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Save the `JasperReportsXLSExporter` class.

6. Double-click the `compile.bat` file to compile the `JasperReportsXLSExporter` class. A command prompt will appear. It will compile `JasperReportsViewer.java` and show the compilation results. Click the **Close** button to dismiss the command prompt.
7. Double-click the `run.bat` file to execute your Java application. A command prompt will appear, which will show a GUI window containing a **Generate Excel Report** button. Click the **Generate Excel Report** button, and you will see **Your report compiled successfully** and **Your report filled with application data** messages on the command prompt. Dismiss the GUI as well as the command prompt by clicking the **Close** button.



8. Next you will add code to save your JasperReport as XLS in your PC using the JasperReports XLS exporter class named `JRXlsExporter`. Type the following code at the end of the `saveXLSReport()` method:

```
JRXlsExporter xlsExporter = new JRXlsExporter();
xlsExporter.setParameter(
    JRExporterParameter.JASPER_PRINT, jasperPrint);
xlsExporter.setParameter(
    JRExporterParameter.OUTPUT_FILE_NAME, xlsFileName);
xlsExporter.exportReport();
System.out.println("Your report saved successfully in
    "+xlsFileName+" file");
```

After adding the code, the `saveXLSReport()` method will become:

```
public void saveXLSReport ()
{
    try
    {
        String xlsFileName = "MonthlyCustomerInvoices.xls";
        // Create wrapper instance and connect to DB
```

```

JasperReportsWrapper wrapper = new JasperReportsWrapper();
Connection connection = wrapper.connect2DB(
    wrapper.dbServerAdd, wrapper.dbServerPort,
    wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
// Compile and fill JasperReport with application data
JasperPrint jasperPrint = wrapper.fillReport(
    wrapper.compileJRXMLFile (wrapper.path2JRXMLFile),
    null, connection);
// Save Jasper report as Excel file
JRxlsExporter xlsExporter = new JRxlsExporter();
xlsExporter.setParameter(
    JRExporterParameter.JASPER_PRINT, jasperPrint);
xlsExporter.setParameter(
    JRExporterParameter.OUTPUT_FILE_NAME, xlsFileName);
xlsExporter.exportReport();
System.out.println ("Your report saved successfully in
"+xlsFileName+" file");
} catch (JRException ex) {
    ex.printStackTrace();
}
}

```

Save your JasperReportsXLSExporter class.

9. Double-click the `compile.bat` file to compile the `JasperReportsXLSExporter` class. A command prompt will appear. It will compile `JasperReportsXLSExporter.java` and show the compilation results. Click the **Close** button to dismiss the command prompt.
10. Double-click the `run.bat`. It will execute your compiled Java application to show the same GUI window that you saw earlier in step 7. Click the **Generate Excel Report** button. You will see a **Your report successfully saved in XLS** message on command prompt. Click the **Exit** button on top-right corner of the GUI to dismiss it. Now browse to the `C:\JasperReportsCookBookSamples\` folder on your PC, where you will find a `MonthlyCustomerInvoices.xls` file. This is the JasperReport, which your `JasperReportsXLSExporter` Java application has generated and saved for you.

Creating a JasperReport on the fly in a Java web application

This recipe teaches you how to embed JasperReports functionality in a JSP-based web application.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed.

The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb9` and copy sample data for this recipe into the database.

You will need Java Development Kit (JDK) version 1.6.X and Apache Tomcat version 6.0.X to compile and run the Java web application of this recipe. You will find all the necessary downloads and instructions for JDK on Sun's official web site <http://java.sun.com>. After installing JDK, you will set the `JAVA_HOME` environment variable to point to the main folder of your JDK installation.

Download Apache Tomcat from Apache's web site (<http://tomcat.apache.org/>) and install it. You will also set the `CATALINA_HOME` environment variable to point to the main folder of your Tomcat installation (`..\apache-tomcat-6.0.X`).

You will be using `CustomerInvoicesReport.jsp` and `MonthlyCustomerInvoices.jrxml` files in this recipe. You will find these files in the `Task5\JasperWeb` folder of the source code download for this chapter.

You will find a subfolder named `classes` inside the `Task5\JasperWeb\WEB-INF\` folder of the source code download for this chapter. This `classes` folder contains source code as well as the compiled form of two Java classes named `ConnectionManager` and `JasperReportsWrapper`. Copy the complete `JasperWeb` folder along with all its subfolders from the `Task5` folder into the `..\apache-tomcat-6.0.X\webapps` folder of your Tomcat installation.

In order to compile `CustomerInvoicesReport.jsp`, you will need following seven JAR files:

- ▶ `jasperreports-3.6.0.jar`
- ▶ `commons-beanutils-1.8.0.jar`
- ▶ `commons-collections-3.2.1.jar`
- ▶ `commons-digester-1.7.jar`

- ▶ commons-logging-1.1.jar
- ▶ groovy-all-1.5.5.jar
- ▶ poi-3.2-FINAL-20081019.jar

Although your `CustomerInvoicesReport.jsp` will compile with these seven JAR files, your complete application will not run without the JDBC JAR file that contains the driver for PostgreSQL. The name of the JAR file is `postgresql-8.2-506.jdbc3.jar`. You can download PostgreSQL JDBC driver from <http://jdbc.postgresql.org/download.html>. You can also find this `postgresql-8.2-506.jdbc3.jar` file in the `...\Jaspersoft\iReport-nb-3.X\ide8\modules\ext` folder of your iReport installation.

Once you have all the JAR files ready, you will put all eight JAR files into the `...\apache-tomcat-6.0.X\webapps\JasperWeb\WEB-INF\lib` folder, which is a subfolder of the JasperWeb folder that you earlier copied into your Tomcat installation.

You will also need a web browser to try this web application. I used Mozilla Firefox version 3.6 to try this recipe. You can download Mozilla from its official website <http://www.mozilla.com/firefox/>.

How to do it...

The following steps demonstrate how to display your report in a web application:

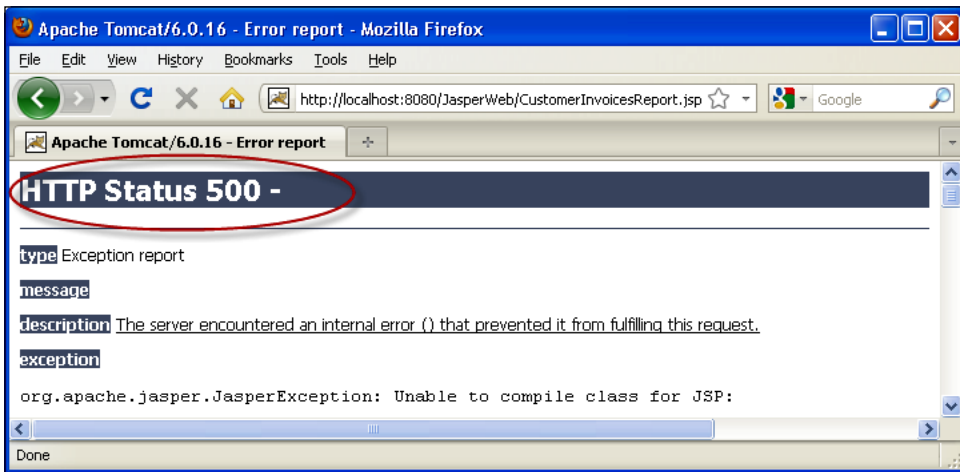
1. Open the `CustomerInvoicesReport.jsp` file from the `webapps\JasperWeb` folder in a text editor. You will find some existing JSP code in the `CustomerInvoicesReport.jsp` file. Locate the `<body>` tag in the existing code and type the following highlighted code inside the `try` block of the `<body>` tag

```
<body>
<%
try
{
    String path2JRXMLFile =
        getServletContext().getRealPath(
            "MonthlyCustomerInvoices.jrxml");
    //Connect to DB and compile JRXML file
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    wrapper.connect2DB(wrapper.dbServerAdd, wrapper.dbServerPort,
        wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
    JasperReport jasperReport =
        wrapper.compileJRXMLFile(path2JRXMLFile);
    JasperPrint jasperPrint =
        wrapper.fillReport(jasperReport,
            null, wrapper.getConnection());
} catch (Exception e) {
```

```
e.printStackTrace();  
}  
%>  
</body>
```

Note that you are using a wrapper Java class named `JasperReportsWrapper` in the code you just typed. Save the `CustomerInvoicesReport.jsp` file.

2. Double-click `startup.bat` from `..\apache-tomcat-6.0.X\bin` of your Tomcat installation. A command prompt will appear, which will start the Tomcat web server.
3. Open a new browser window and type `http://localhost:8080/JasperWeb/CustomerInvoicesReport.jsp` in the address bar of your browser. You will see an error page in your browser window. Exit the browser window.

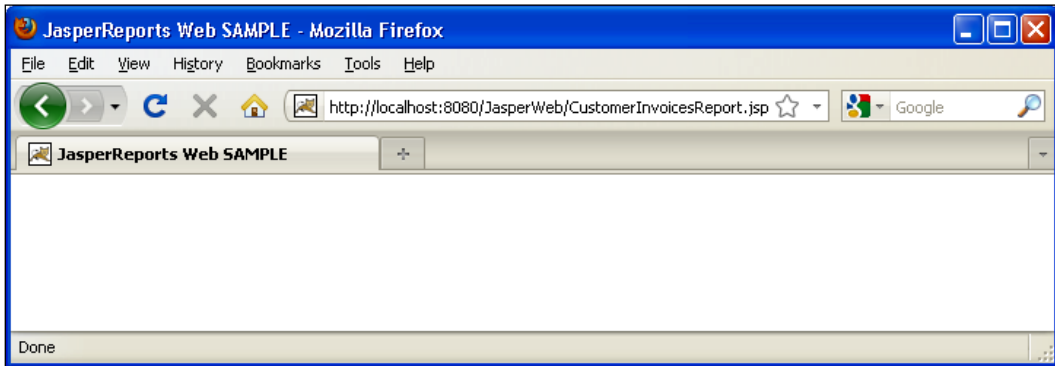


4. Shut down the Tomcat web server by double-clicking `shutdown.bat` from `..\apache-tomcat-6.0.X\bin` of your Tomcat installation.
5. Now open the `CustomerInvoicesReport.jsp` file in a text editor. Type the following code at the start of the JSP file to import Java libraries:

```
<%@ page language="java" session="false" %>  
<%@ page import="net.sf.jasperreports.engine.*" %>  
<%@ page import="net.sf.jasperreports.engine.export.*" %>  
<%@ page import="com.JasperReportsWrapper" %>
```

Save your `CustomerInvoicesReport.jsp` file.

6. Double-click `startup.bat` from `..\apache-tomcat-6.0.X\bin` of your Tomcat installation. A command prompt will appear, which will start the Tomcat web server. Open a new browser window and type `http://localhost:8080/JasperWeb/CustomerInvoicesReport.jsp` in the address bar. This time you will see a blank page in your browser window without any error. Exit the browser window.



7. Shut down the Tomcat web server by double-clicking `shutdown.bat` from `..\apache-tomcat-6.0.X\bin` of your Tomcat installation.
8. Now open the `CustomerInvoicesReport.jsp` file in a text editor. Type the following code in a try block of the `<body>` tag just below the code that you typed earlier in step 1, to generate and view the report inside the browser window.

```
<!--existing JSP Code-->
JRPdfExporter exporter = new JRPdfExporter();
exporter.setParameter(
    JRExporterParameter.JASPER_PRINT, jasperPrint);
exporter.setParameter(
    JRExporterParameter.OUTPUT_STREAM,
    response.getOutputStream());
exporter.exportReport();
```

After typing the code, your complete `<body>` tag will look as follows:

```
<body>
<%
try
{
    String path2JRXMLFile =
        getServletContext().getRealPath(
            "MonthlyCustomerInvoices.jrxml");
    //Connect to DB and compile JRXML file
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    wrapper.connect2DB(wrapper.dbServerAdd, wrapper.dbServerPort,
```



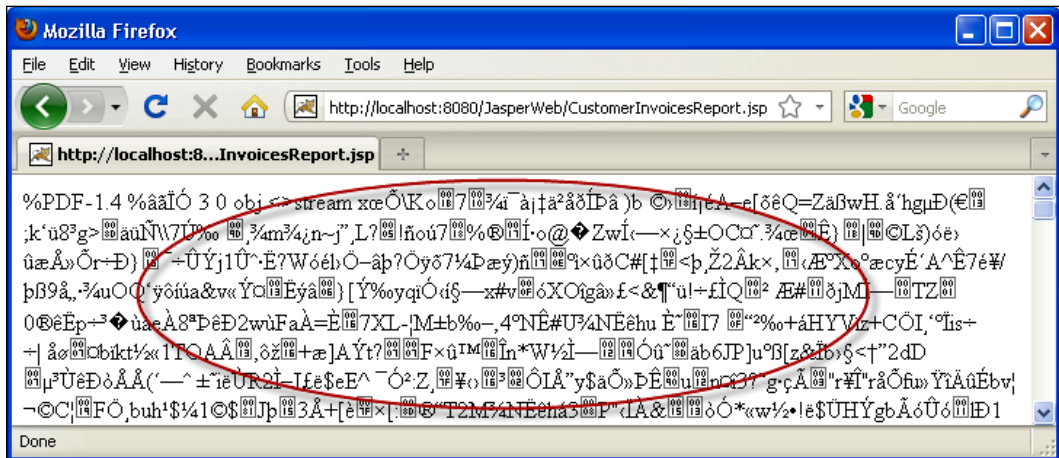
```

        wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
    // Compile JRXML file
    JasperReport jasperReport =
        wrapper.compileJRXMLFile(wrapper.path2JRXMLFile);
    JasperPrint jasperPrint =
        wrapper.fillReport(jasperReport,
            null, wrapper.getConnection());
    // Create and export PDF to browser window
    JRPdfExporter exporter = new JRPdfExporter();
    exporter.setParameter(
        JRExporterParameter.JASPER_PRINT, jasperPrint);
    exporter.setParameter(
        JRExporterParameter.OUTPUT_STREAM,
        response.getOutputStream());
    exporter.exportReport();
} catch (Exception e) {
    e.printStackTrace();
}
%>
</body>

```

Save your `CustomerInvoicesReport.jsp` file.

9. Double-click `startup.bat` from your Tomcat installation. A command prompt will appear, which will start the Tomcat web server. Now type `http://localhost:8080/JasperWeb/CustomerInvoicesReport.jsp` in the address bar of your browser. You will see a lot of junk characters in the browser window.



10. Shut down the Tomcat web server by double-clicking shutdown.bat from ..\apache-tomcat-6.0.X\bin of your Tomcat installation.
11. Open the CustomerInvoicesReport.jsp file in a text editor. Locate the line of code containing JRPdfExporter inside the try block of the <body> tag and type the following code just before the JRPdfExporter line of code:

```
response.setContentType("application/pdf");
```

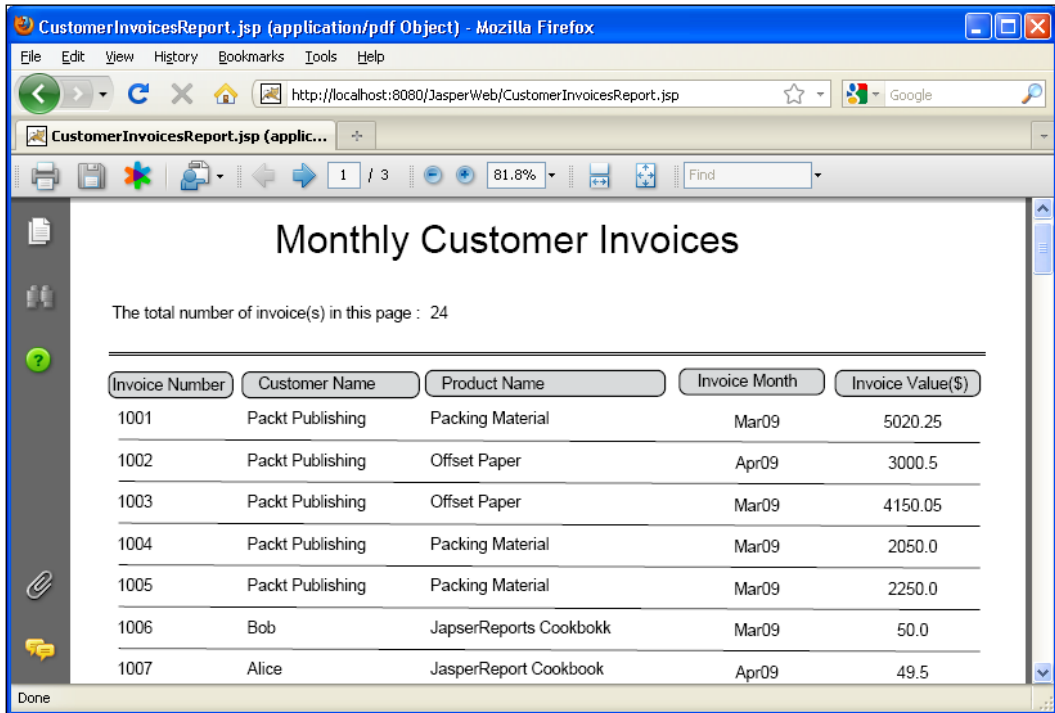
After typing the code, your complete <body> tag will look as follows:

```
<body>
<%
try
{
    String path2JRXMLFile =
        getServletContext().getRealPath(
            "MonthlyCustomerInvoices.jrxml");
    JasperReportsWrapper wrapper = new JasperReportsWrapper();
    //Connect to DB and compile JRXML file
    wrapper.connect2DB(wrapper.dbServerAdd, wrapper.dbServerPort,
        wrapper.dbName, wrapper.dbUser, wrapper.dbPass);
    JasperReport jasperReport =
        wrapper.compileJRXMLFile(wrapper.path2JRXMLFile);
    JasperPrint jasperPrint =
        wrapper.fillReport(jasperReport,
            null, wrapper.getConnection());

    // Set response content type
    response.setContentType("application/pdf");
    // Create and export PDF to browser window
    JRPdfExporter exporter = new JRPdfExporter();
    exporter.setParameter(
        JRExporterParameter.JASPER_PRINT, jasperPrint);
    exporter.setParameter(
        JRExporterParameter.OUTPUT_STREAM,
        response.getOutputStream());
    exporter.exportReport();
} catch (Exception e) {
    e.printStackTrace();
}
%>
</body>
```

Save your CustomerInvoicesReport.jsp file.

12. Double-click `startup.bat` from your Tomcat installation. A command prompt will appear, which will start the Tomcat web server. Now type `http://localhost:8080/JasperWeb/CustomerInvoicesReport.jsp` in the address bar of your browser. This time you will see a PDF report containing customer invoices in your browser window, as shown in the following screenshot:



How it works...

You have displayed your **Monthly Customer Invoices** report in four steps in this recipe. First, in step 1, you wrote the code to directly use a Java wrapper class named `JasperReportsWrapper` in your JSP code. But it didn't work, as you could not compile the JSP page in step 3. That's because Tomcat could not find the JasperReport libraries.

So you imported JasperReports libraries in step 5 and then again tried the application. This time it compiled but you got a blank page in your browser (step 6). The page was blank because your `CustomerInvoicesReport.jsp` was not sending the report in its response stream to the browser. So you added code to include the report in the response stream in step 8 and tried again. This time you saw a lot of junk characters being displayed in the browser window. You saw junk characters because the browser could not recognize the type of content coming in the response stream. So, in step 11, you set the content type of the response data to `application/pdf`. This time, the browser recognized the content as a PDF file and displayed it correctly.

Refer to steps 1, 8, and 11 of the recipe, where you typed Java code directly into the JSP file. These steps are only to demonstrate the working of JasperReports in a JSP page. In actual practice, you will normally wrap all this Java code inside some Java classes and use the classes from your JSP code. This is recommended much more than directly embedding Java code in JSP.

There's more...

In this recipe, you learned how to use JasperReports in a JSP application. You can use the concepts you learned in this recipe to integrate JasperReports into popular frameworks like Spring or Struts. Although discussing an actual framework is beyond the scope of this cookbook, you can find useful information and tutorials on the Web that discuss this topic.

While you are considering integrating JasperReports into a framework, it will always help to keep in mind the flow diagram given in the *How it works* section of the *Creating a Java wrapper for your report* recipe of this chapter.

9

Using Mathematical and Logical Expressions

In this chapter, you will learn:

- ▶ Calculating the sum and average of a column
- ▶ Calculating the number of records shown on a page
- ▶ Counting the number of records with a particular field value
- ▶ Finding the maximum of a particular field
- ▶ Changing the background color of alternate records
- ▶ Applying styles on your data based on a logical or mathematical condition
- ▶ Changing background color of a particular record based on a mathematical or logical expression

Introduction

This chapter demonstrates the use of mathematical and logical expressions. Expressions are a valuable and powerful feature of JasperReports and the recipes of this chapter present several important ways to use expressions.

Expressions make use of variables and functions. Therefore, you will learn about declaring and using variables and also come across several built-in functions of JasperReports in this chapter.

Here you will also learn an important property named **Print When Expression**. This property is useful whenever you want to display something conditionally, that is based on a mathematical or logical condition.

Calculating the sum and average of a column

Almost every report requires some sort of mathematical operations like summing record values or finding their average. JasperReports has built-in features to provide these simple mathematical operations. This recipe teaches how you will use the built-in functions of JasperReports to find the sum and average of a column in your report.

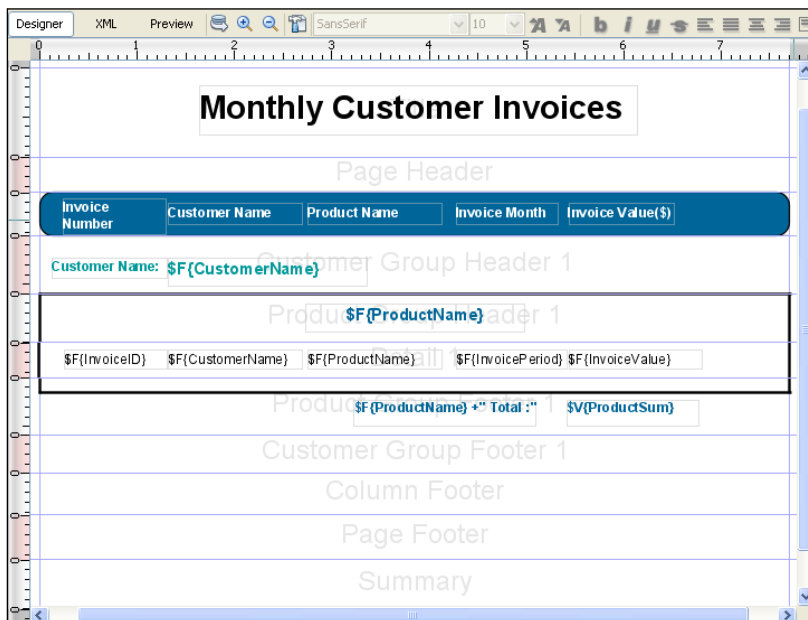
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code of this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

The following steps guide you in how to use built-in mathematical functions of JasperReports:

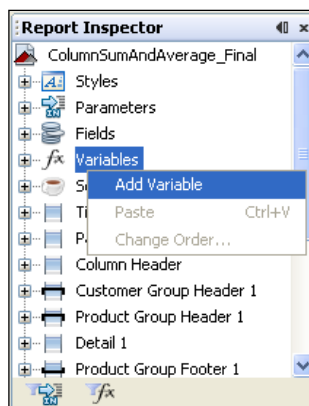
1. Open the `ColumnSumAndAverage.jrxml` file from the `Task1` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in **Title**, **Column Header**, **Customer Group Header 1**, **Product Group Header 1**, **Detail 1**, and **Product Group Footer 1** sections, as shown in the following screenshot:



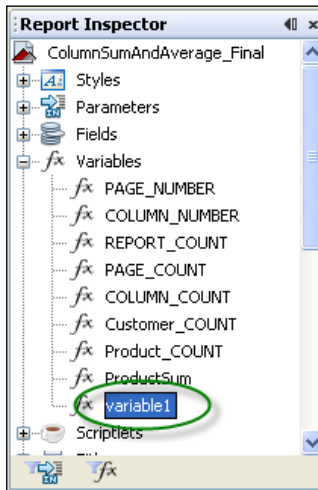
- Switch to the **Preview** tab and you will see invoices for each customer grouped by product names, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
Offset Paper				
1012	ABC Publishing	Offset Paper	Jun09	8875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
Offset Paper Total :				11427.0
Packing Material				
1011	ABC Publishing	Packing Material	Jun09	8746.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				18597.0
Printing Ink				
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Printing Ink Total :				5890.0

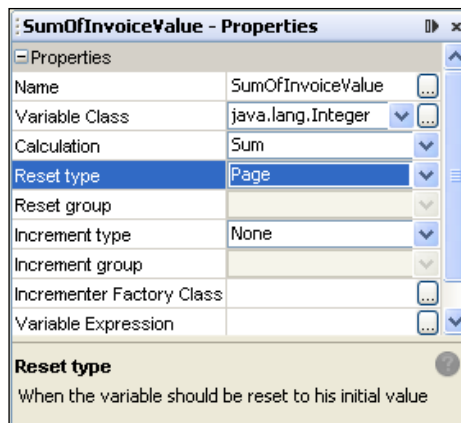
- Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window on the left side of your report. A pop-up menu will appear. Select the **Add Variable** option.



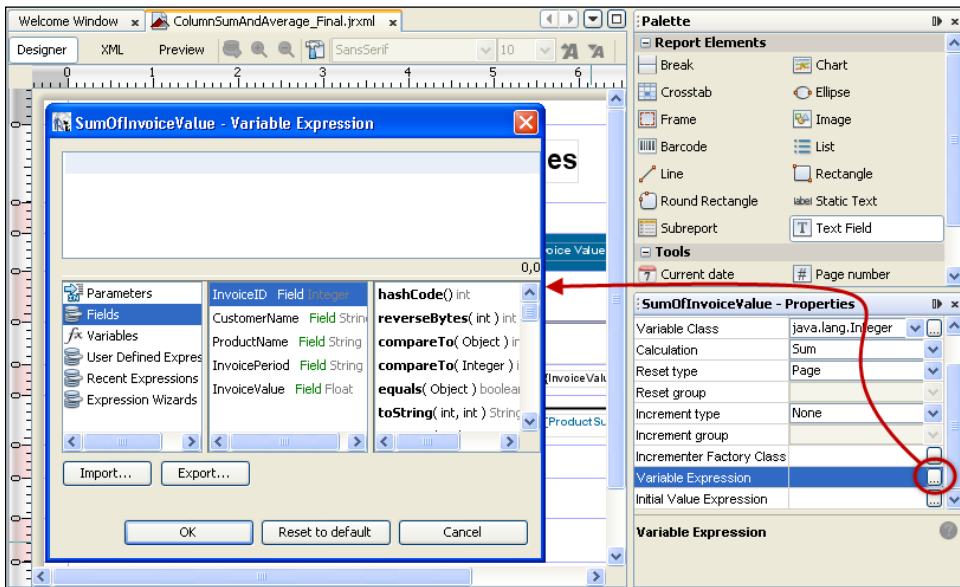
4. A new variable named **variable1** will be added at the end of the variables list, as shown follows:



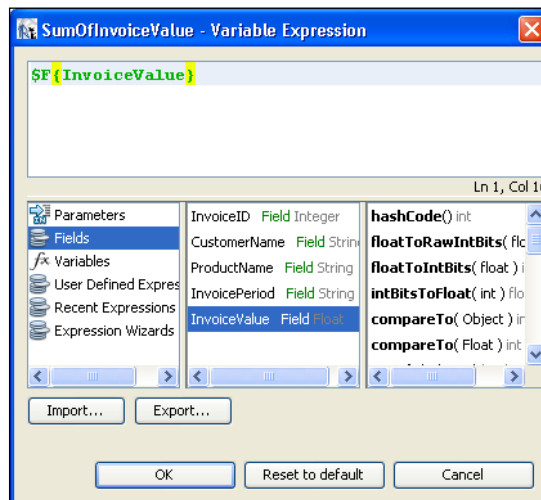
5. While **variable1** is selected, find the **Name** property in the **Properties** window below the palette of components and change its value to **SumOfInvoiceValue**. Now the name of the variable1 variable will change to **SumOfInvoiceValue**.
6. Select the **Variable Class** property and change its value to **java.lang.Integer**.
7. Select the **Calculation** property and change its value to **Sum**.
8. Select the **Reset Type** property and change its value to **Page**, as shown in the following screenshot:



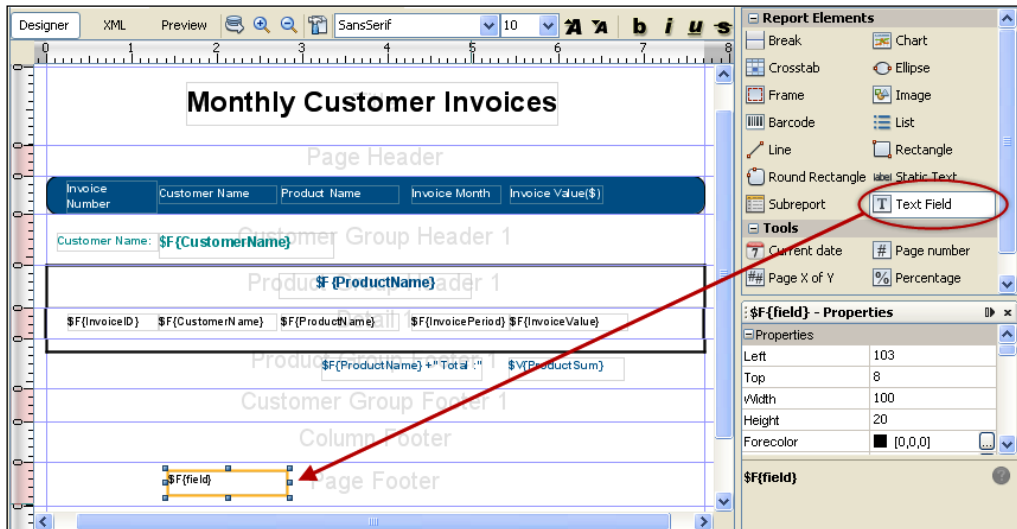
9. Select **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open, as shown in the following screenshot:



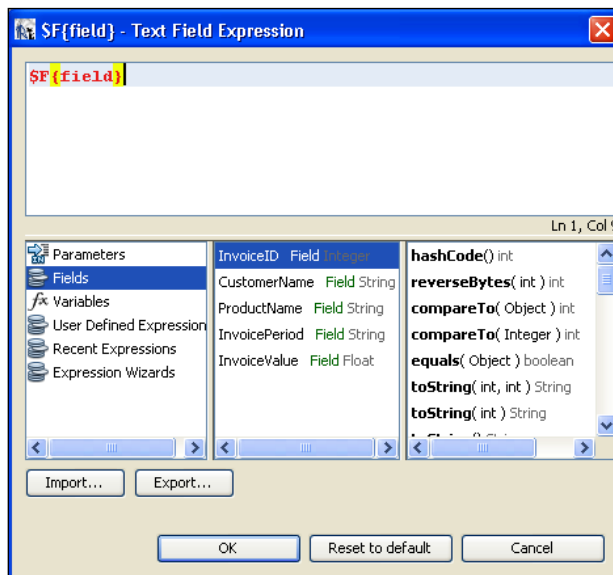
10. Look for **InvoiceValue** field in the second column of the lower-half of the **Variable Expression** window and double-click on it. The `$F{InvoiceValue}` expression will be added to the expression window. Click **OK**, as shown in the following screenshot:



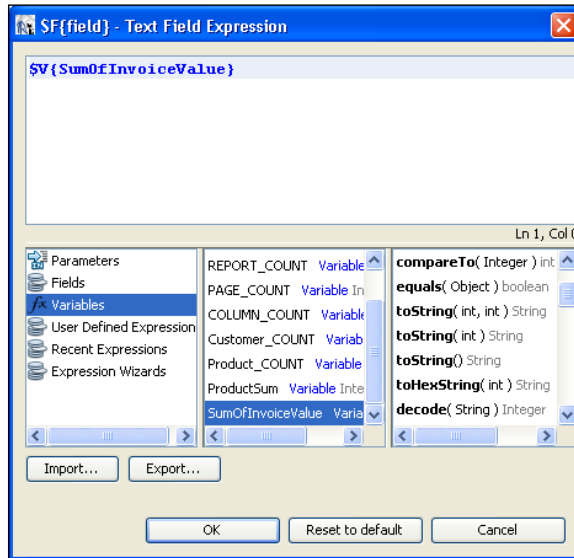
11. Now, drag-and-drop a **Text Field** component into the **Page Footer** section, as shown in the following screenshot:



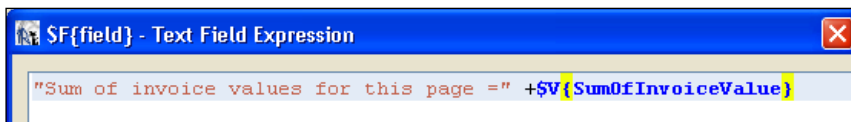
12. Select the **Text Field** component; its properties will appear in the **Properties** window below the palette. Find the **Text Field Expression** property and click on the button beside it. A **Text Field Expression** window will open, as shown in the following screenshot:



13. Delete the default expression (`$F{field}`) from the **Text Field Expression** window. Select **Variables** in the first column of the lower-half of the **Text Field Expression** window. Then double-click on the **SumOfInvoiceValue** variable in the second column. A new expression **`$V{SumOfInvoiceValue}`** will appear in the **Text Field Expression** window, as shown in the following screenshot:



14. Type Sum of invoice values for this page = + in the expression editor window just before the **`$V{SumOfInvoiceValue}`** expression. The complete expression in the editor will become **"Sum of invoice values for this page =" +`$V{SumOfInvoiceValue}`**, as shown in the following screenshot. Click **OK**.

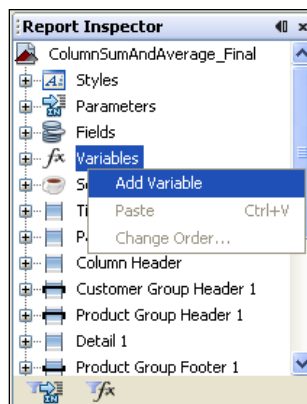


15. While the `$V{SumOfInvoiceValue}` field is selected, find the **Width** property in the **Properties** window and set **210** as its value. Now the text field will show almost the complete expression.

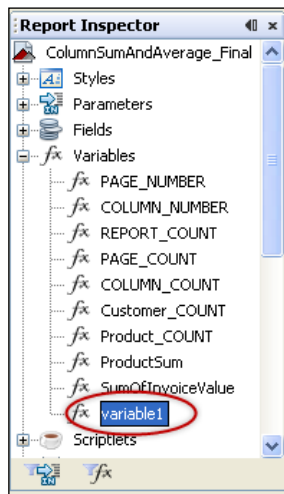
16. Switch to the **Preview** tab and you will see the sum on invoice values of the page at the bottom of each page in the report, as shown:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
Offset Paper				
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1012	ABC Publishing	Offset Paper	Jun09	6875.0
Offset Paper Total :				11427.0
Packing Material				
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				18597.0
Printing Ink				
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
Printing Ink Total :				5890.0
Sum of invoice values for this page =35914				

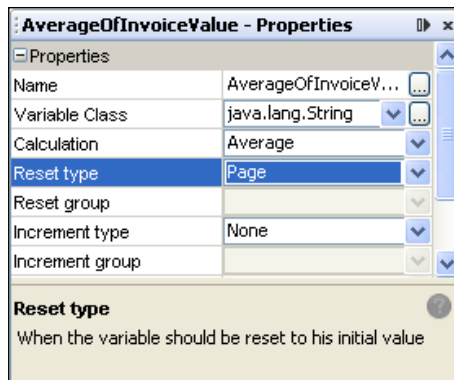
17. Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Variable** option.



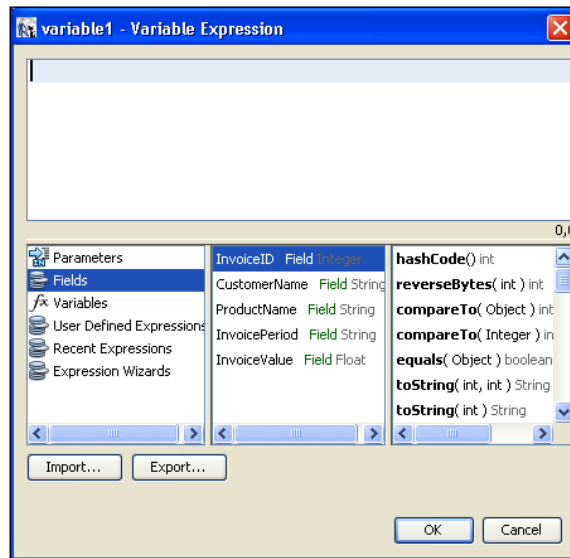
18. A new variable **variable1** will be added to the variables list, as shown:



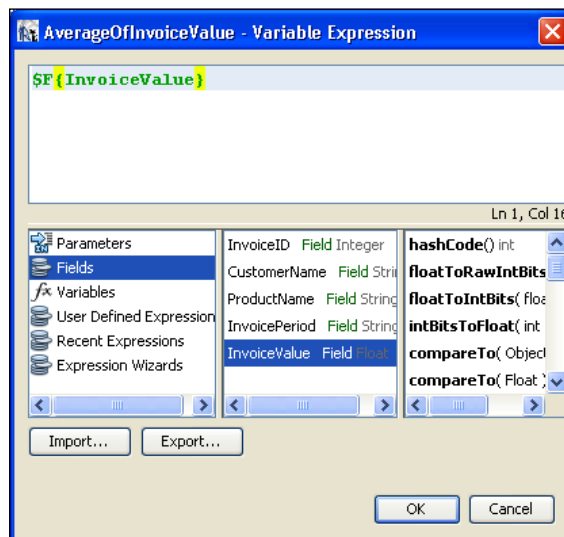
19. While **variable1** is selected, find the **Name** property in the **Properties** window below the palette of components and change its value to **AverageOfInvoiceValue**. Now the name of the **variable1** variable will change to **AverageOfInvoiceValue**.
20. Select the **Variable Class** property and change its value to **java.lang.Integer**.
21. Select the **Calculation** property and change its value to **Average**.
22. Select the **Reset Type** property and change its value to **Page**, as shown in the following screenshot:



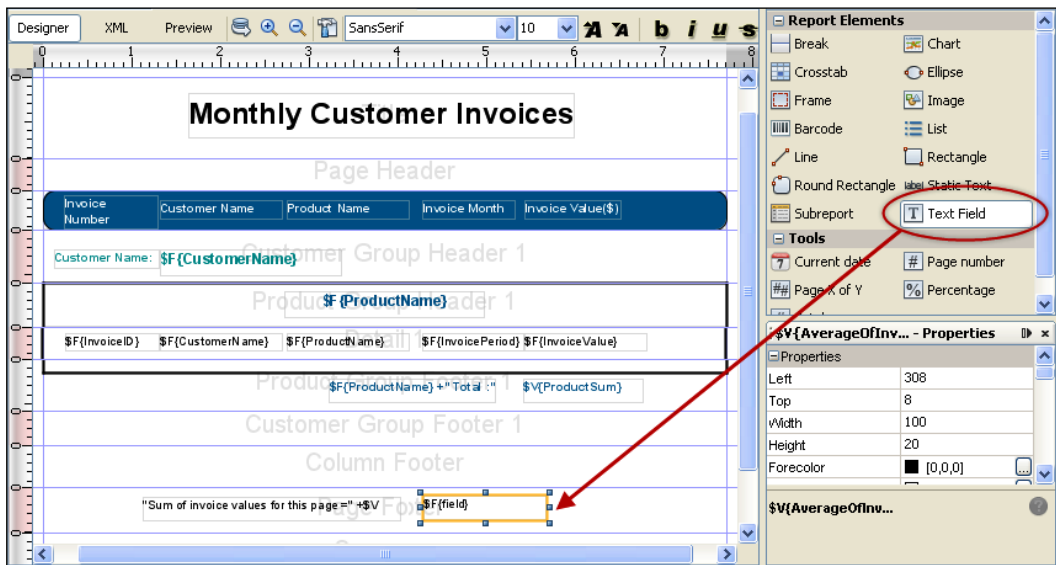
23. Select the **Variable Expression** property and click on the button beside it. A **Variable Expression** window with no default expression will open, as shown in the following screenshot:



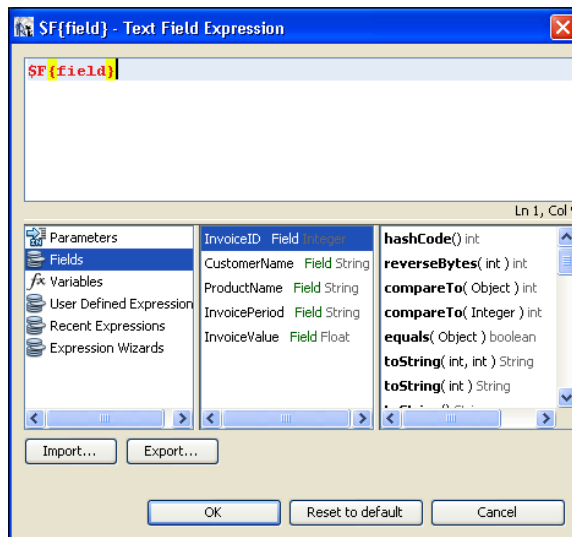
24. Find the **InvoiceValue** field in the second column of the lower-half of the **Variable Expression** window and double-click on it. The `$F{InvoiceValue}` expression will be added to the expression window. Click **OK**, as shown in the following screenshot:



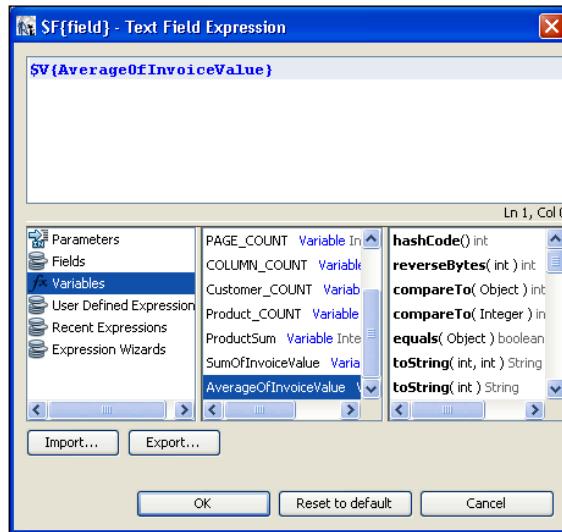
25. Now, drag-and-drop another **Text Field** component into the **Page Footer** section and place it to the right of the **Text Field** component you dropped earlier in step 11, as shown in the following screenshot:



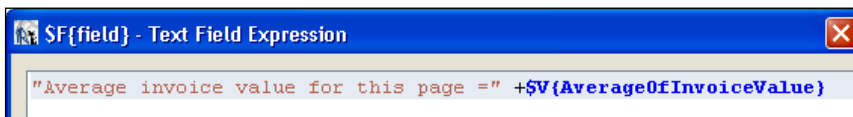
26. Select the **Text Field** component; its properties will appear in the **Properties** window below the palette. Find the **Text Field Expression** property and click on the button beside it. A **Text Field Expression** window will open, as shown in the following screenshot:



27. Delete the default expression (`$F{field}`) from the **Text Field Expression** window. Select **Variables** in the first column of the lower-half of the **Text Field Expression** window. Double-click on the **AverageOfInvoiceValue** variable in the second column. A new expression **`$V{AverageOfInvoiceValue}`** will appear in the **Text Field Expression** window, as shown in the following screenshot:



28. Type `Average of invoice value for this page =` + text as suffix to the existing variable expression text (`$V{InvoicesAverage}`) in the **Text Field Expression** window. The complete expression in the editor will become **`"Average of invoice value for this page =" + $V{SumOfInvoiceValue}`**, as shown in the following screenshot. Click **OK**.



29. While the **`$V{SumOfInvoiceValue}`** field is selected, find the **Width** property in the **Properties** window and set **210** as its value. Now the text field will show almost the complete expression.

30. Switch to the **Preview** tab and you will see the average of all invoices on the page at the bottom of each page of the report, as shown encircled in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
Offset Paper				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4652.2
Offset Paper Total :				11427.0
Packing Material				
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Packing Material Total :				18597.0
Printing Ink				
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Printing Ink Total :				5890.0
Sum of invoice values for this page = 35914				Average invoice value for this page = 5985

How it works...

You have displayed two things in this recipe. First, in steps 3 to 15, you displayed the sum of the invoice values and then, in steps 17 to 29, you displayed the average of the invoice values.

While displaying the sum, you did four things:

1. First, in step 3, you added a new variable to your report.
2. Then, in step 7, you associated the new variable with a built-in function named **Sum**. Then, in step 8, you set the **Reset Type** property of the new variable to **Page**, which means, value of the new variable will be reset at the start of every page of your report.
3. Then, in step 10, you associated the new variable with the **InvoiceValue** field. This means you want this variable to calculate the sum of all invoices in the **InvoiceValue** field. The combined effect of steps 7, 8, and 10 is that the new variable keeps on calculating the mathematical sum of invoice values and when a new page starts it gets reset, so that it is always fresh at the first invoice of every page.

4. Finally, in step 13, you displayed the sum of all the invoice values by associating the variable with a **Text Field** component.

You did the same four things while displaying the average. The only difference was that in step 21, you used the `Average` built-in function (instead of the `Sum` built-in function).

There's more...

In steps 7 and 21, you used the **Calculation** property of your variables. This **Calculation** property actually maps to built-in mathematical and statistical functions of the JasperReports. In this recipe, you have used two of the functions, `Sum` and `Average`. Later in *Finding the minimum and maximum of a particular field value among a range of records* recipe, you will use `Lowest` and `Highest` functions of the **Calculation** property. You can try the rest of the functions by playing with them in a similar manner.

You may also refer to *Inserting a heading for a group of records* recipe of Chapter 2, *Working with the Body and Footer of your Report*, which teaches you how to group records shown in the body of your report. You can find `Sum` and `Average` of all invoices in a group by simply selecting **Group** as the value of the **Reset Type** property in step 8 and 22 of this recipe, respectively.

Calculating the number of records shown on a page

This recipe teaches you how to display the count of records shown on a single page of your report. Counting the number of records is required in scenarios such as when you want each page of your customer invoices report to tell how many invoices it shows.

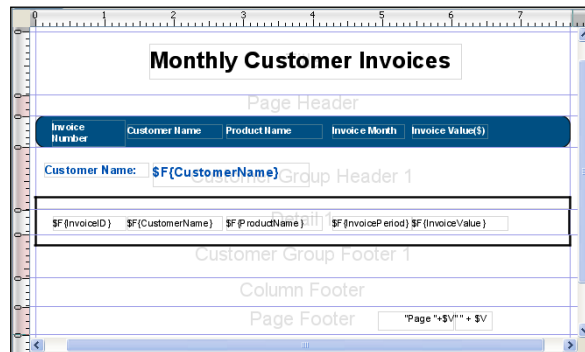
Getting ready

Refer to `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

The following nine steps demonstrate how you can include a per-page record counter in a report:

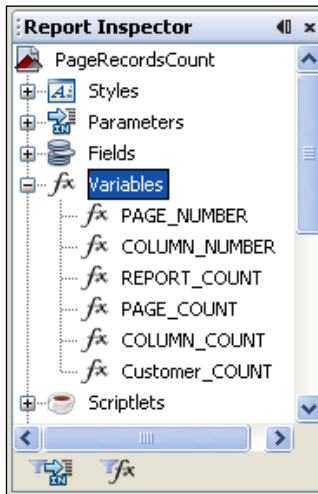
1. Open the `PageRecordsCount.jrxml` file from the `Task2` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header1**, and **Detail 1** sections, as shown in the following screenshot:



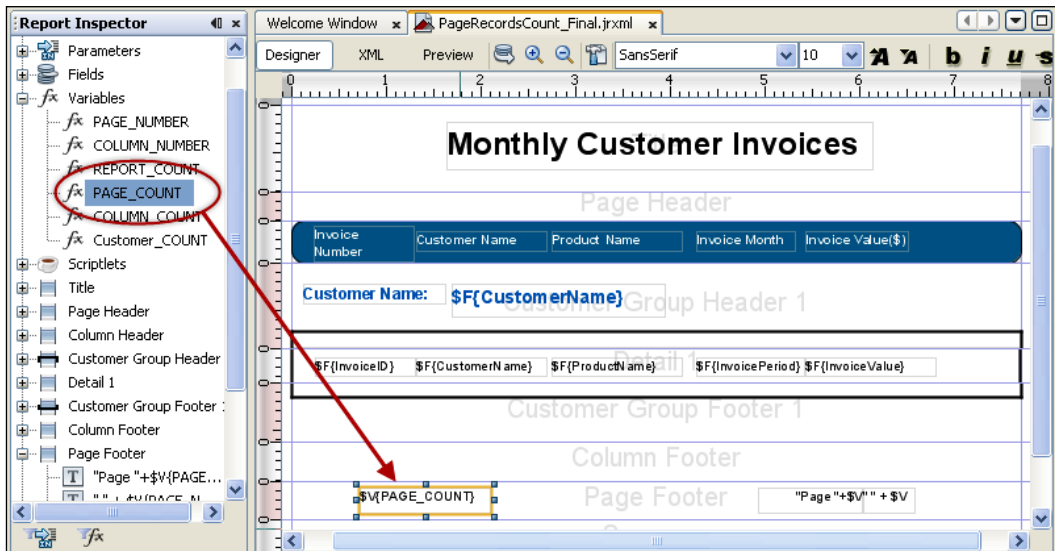
2. Switch to the **Preview** tab and you will see invoices grouped by customer names, as shown in the following screenshot:

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing link	Mar09	3450.0
1016	ABC Publishing	Printing link	Jun09	2440.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
Customer Name: Bob				
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0
1018	Bob	Printing link	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0

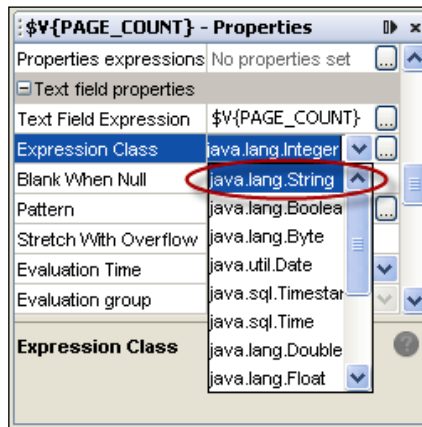
- Switch back to the **Designer** tab. Double-click on the **Variables** node in the **Report Inspector** window on the left side of your report. The **Variables** node will expand showing all its children (that is, all variables present in the report), as shown in the following screenshot:



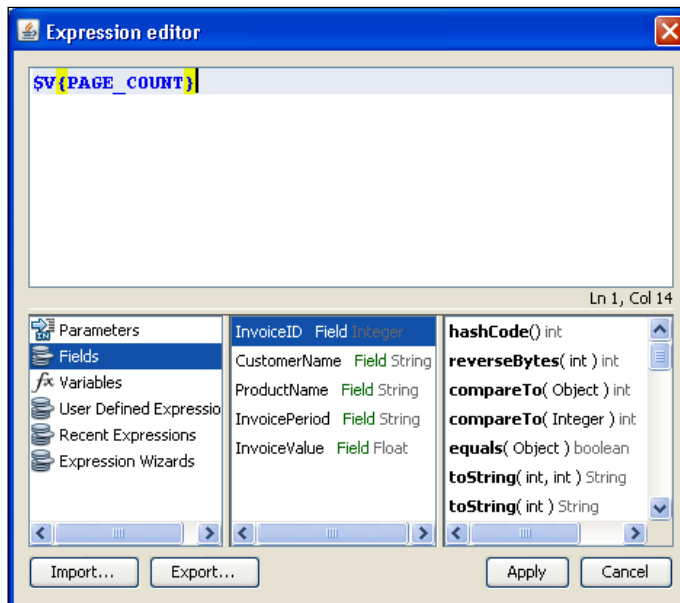
- Drag-and-drop the **PAGE_COUNT** variable from the **Variables** node into the **Page Footer** section of your report, as shown in the following screenshot. This **PAGE_COUNT** variable tells how many records are shown on a page of your report.



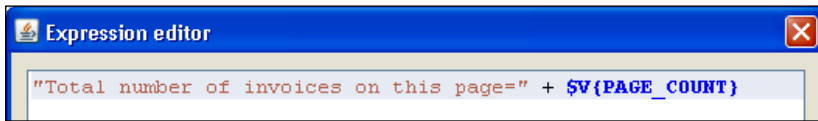
5. Select the **`$V{PAGE_COUNT}`** field dropped into the **Page Footer** section; its properties will appear in the **Properties** window below the palette of components. Find the **Expression Class** property under the **Text field properties** section in the **Properties** window and select `java.lang.String` as its value.



6. Right-click on the **`$V{PAGE_COUNT}`** field and select the **Edit expression** option from the pop-up menu. An **Expression editor** window will open.



7. Type "Total number of invoices on this page=" + in the **Expression editor** window just before the `$V{PAGE_COUNT}` expression. The complete expression in the editor will become `Total number of invoices on this page=" + $V{PAGE_COUNT}`, as shown in the following screenshot. Click **Apply** button.



8. While the `$V{PAGE_COUNT}` field is selected, find the **Width** property in the **Properties** window and set **200** as its value. Now the text field will show almost the complete expression.
9. Switch to the **Preview** tab and you will see the total number on invoices of the page shown at the bottom of each page, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6675.0
1010	ABC Publishing	Offset Paper	Feb09	4662.2
1011	ABC Publishing	Packing Material	Jun09	8746.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
Customer Name: Bob				
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0
Total number of invoices on this page=13				Page 1 of 3

How it works...

In this recipe, you have used a built-in variable of JasperReports named **PAGE_COUNT**. This variable contains the number of records shown on a page. Therefore, whenever you need to know the number of records shown on a page, you will drop this variable into your report.

There's more...

When you expanded the variables node in step 3 of this recipe, it showed a number of variables (**PAGE_NUMBER**, **COLUMN_NUMBER**, **REPORT_COUNT**, **PAGE_COUNT**, **COLUMN_COUNT**, **Customer_COUNT**). Each of these variables is very useful and you can play with them to understand their functions:

- ▶ The **PAGE_NUMBER** variable contains the current page number. Refer to the *Resetting page numbering with the start of a particular record* recipe of *Chapter 6, Multi-page Reports*, to learn the use of this variable.
- ▶ The **COLUMN_NUMBER** variable specifies the current column of your report. You will play with this variable in the recipes of *Chapter 7, Multi-column Reports*.
- ▶ The **REPORT_COUNT** variable contains the total number of records shown in the complete report.
- ▶ The **COLUMN_COUNT** variable contains the total number of records shown in each column of a report page.
- ▶ The **Customer_COUNT** is a group-specific variable. You will learn this group-specific variable in the *Counting the number of records with a particular field value* recipe.

Counting the number of records with a particular field value

JasperReports has powerful features that allow you to count specific types of records. A common counting requirement in report designing is to count records which contain a specific value for a particular field. For example, you may need to count all invoices for a particular customer or a particular product or during a particular time period. This recipe shows you how to do this.

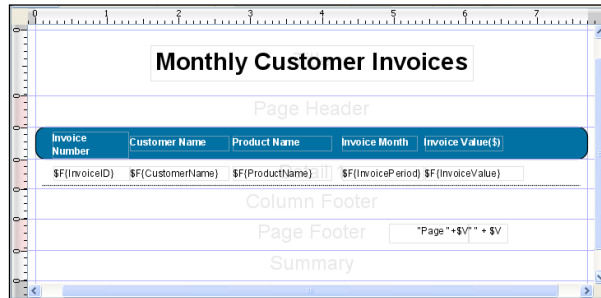
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

Execute the following simple steps to learn how to categorize your records depending upon a particular field value and then count the number of records in each category:

1. Open the `CountingGroupRecords.jrxml` file from the `Task3` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in **Title**, **Column Header**, **Detail 1**, and **Page Footer** sections, as shown in the following screenshot:

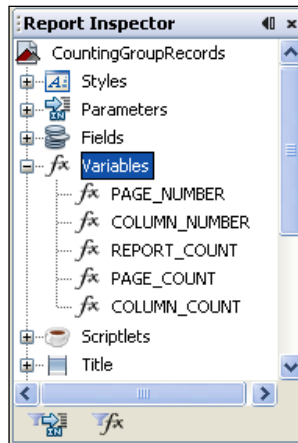


2. Switch to the **Preview** tab; you will see a report containing data about customer invoices, as shown in the following screenshot:

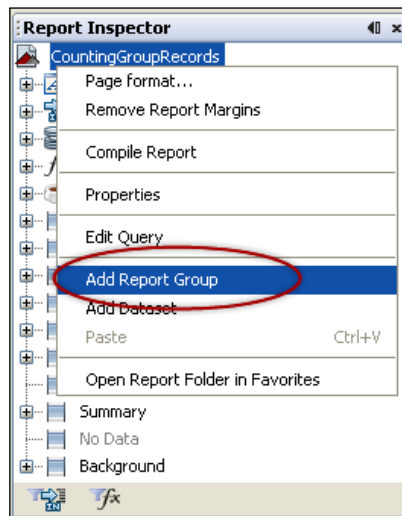
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0
1026	Packt Publishing	Binding Material	Sep09	4550.0
1054	Packt Publishing	Binding Material	Jun09	1558.85
1024	Packt Publishing	Binding Material	Jan09	8770.0
1021	Packt Publishing	Binding Material	Jan09	2580.0
1055	Packt Publishing	Binding Material	Jan09	7895.22
1043	Packt Publishing	Binding Material	Mar09	6522.0
1022	Packt Publishing	Binding Material	Feb09	null

Page 1 of 3

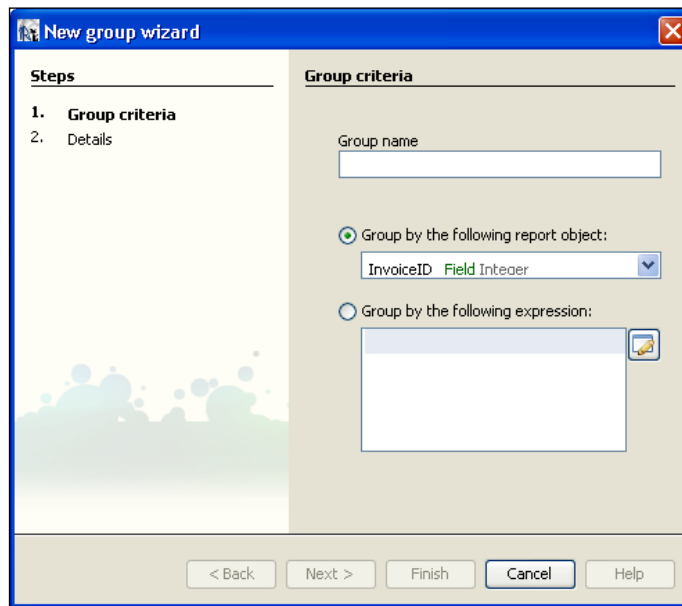
3. Switch back to **Designer** tab. Double-click on the **Variables** node in the **Report Inspector** window (on the left side of the **Designer** tab). The **Variables** node will expand showing all variables present in the report, as shown in the following screenshot:



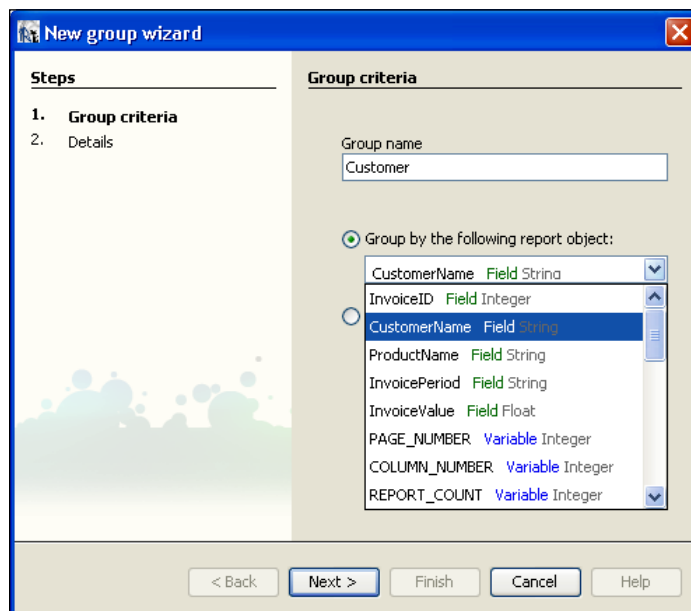
4. Right-click on the top-most element **CountingGroupRecords** in the **Report Inspector** window. A pop-up menu will appear. Select the **Add Report Group** option, as shown in the following screenshot:



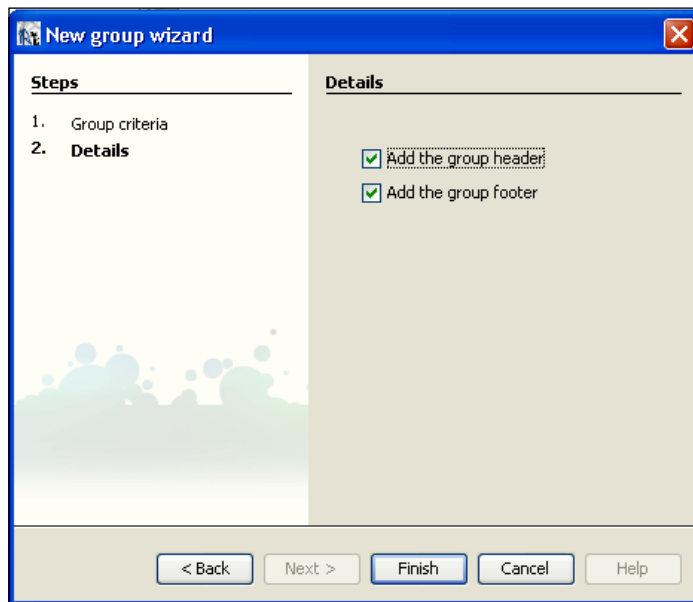
5. A **New group wizard** dialog will appear, as can seen in the following screenshot:



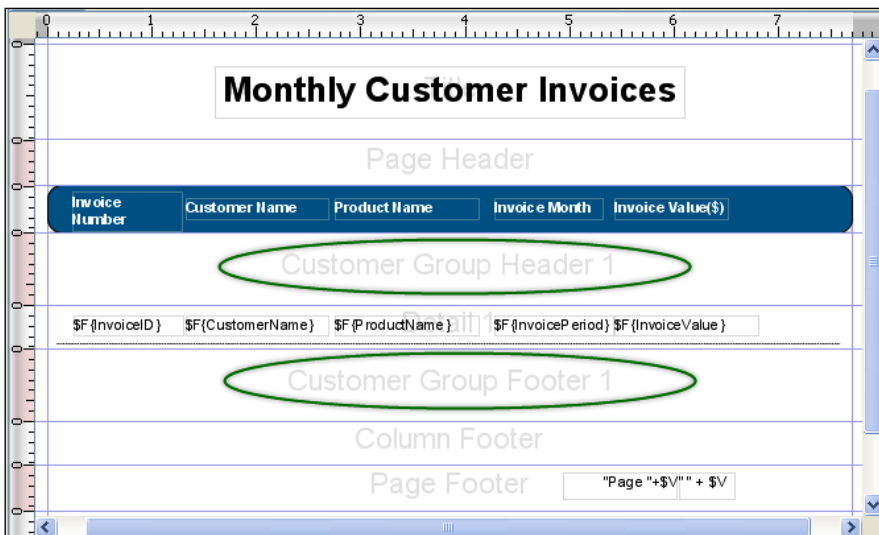
6. Type `Customer` in the **Group name** field and select **CustomerName** in **Group by the following report object** drop-down list, as shown in the following screenshot:



7. Click **Next** button, and the dialog will change as shown in the following screenshot:



8. Click the **Finish** button to dismiss the dialog. You will notice that **Customer Group Header 1** and **Customer Group Footer 1** sections have been added into your report, as shown in the following screenshot:

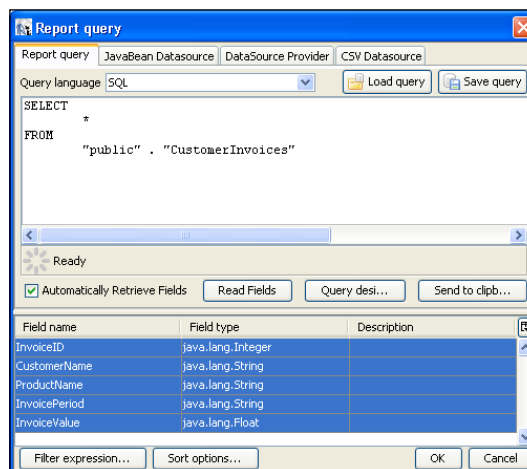


9. Switch to the **Preview** tab and this time you will see a report with invoices grouped based upon customer names. You will notice that there are a number of invoice groups for each customer spread randomly throughout the report, as shown encircled in the following screenshot:

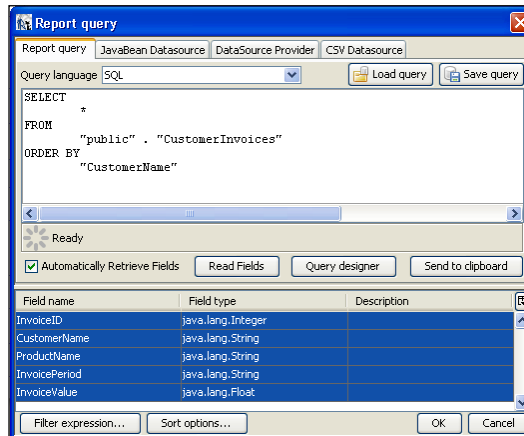
Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002		Offset Paper	Apr09	3000.5
1003		Offset Paper	Mar09	4150.05
1004		Packing Material	Mar09	2050.0
1006	Bdb	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009		Offset Paper	Feb09	5085.0

Page 1 of 6

10. Switch back to the **Designer** tab. Click the **Report query** button at the top of the report window (to right side of the **Preview** tab), a **Report query** dialog will appear, as shown in the following screenshot:



11. Type `ORDER BY "CustomerName"` at the end of the existing SQL query in **Query editor**. Click the OK button, as shown in the following screenshot:

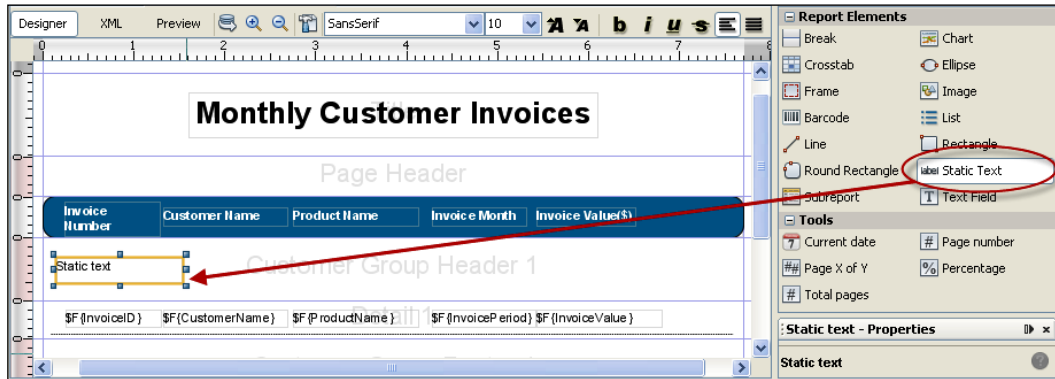


12. Switch to the **Preview** tab. Navigate through all pages of your report, focusing on the grouping of customer names. This time you will notice that there is only one group of records for each customer name (all the records for a single customer are displayed in a single group), as shown in the following screenshot:

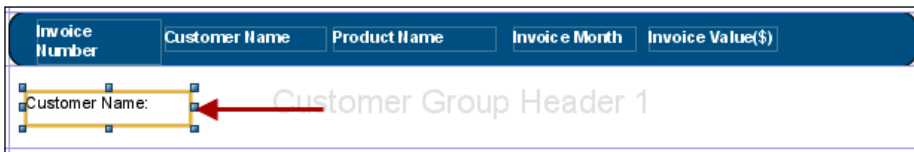
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1007	Alice	JasperReport	Apr09	49.5
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0
1006	Bob	JasperReports	Mar09	50.0

Page 1 of 4

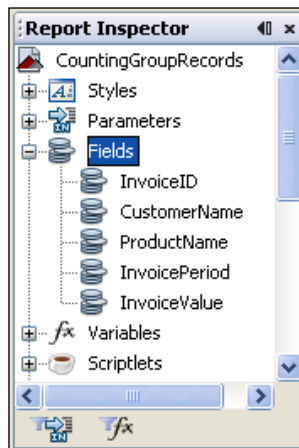
13. Switch back to the **Designer** tab. Drag-and-drop a **Static Text** component from the **Palette** onto the extreme-left position of the **Customer Group Header 1** section, as shown in the following screenshot:



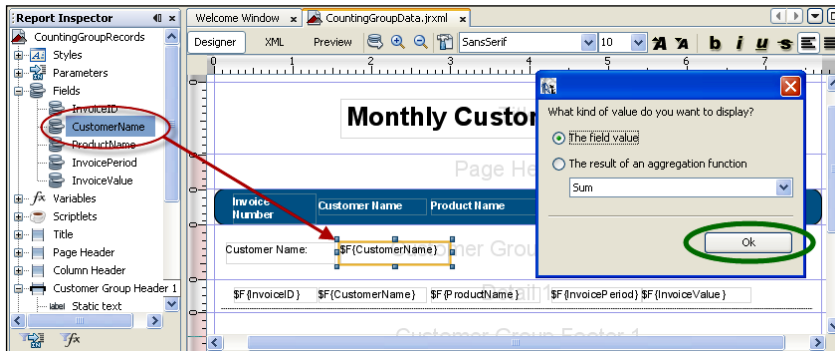
14. Double-click the **Static Text** component and enter **Customer Name:** as its label, as shown in the following screenshot:



15. Double-click on the **Fields** node in the **Report Inspector** window. You will see that it contains **InvoiceID**, **CustomerName**, **ProductName**, **InvoicePeriod**, and **InvoiceValue** fields, as shown in the following screenshot:



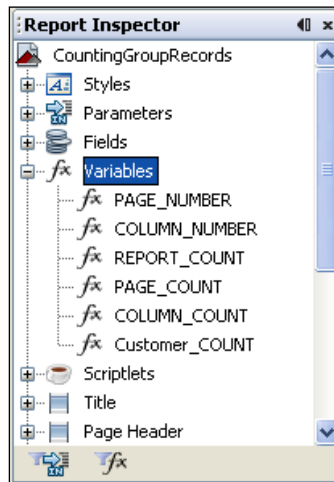
16. Drag-and-drop the **CustomerName** field from the **Fields** node in **Report Inspector** window into the **Customer Group Header 1** adjacent to the existing **Customer Name: Static Text** component. A dialog window will appear, as shown next. Don't change anything; just click the **Ok** button to dismiss it.



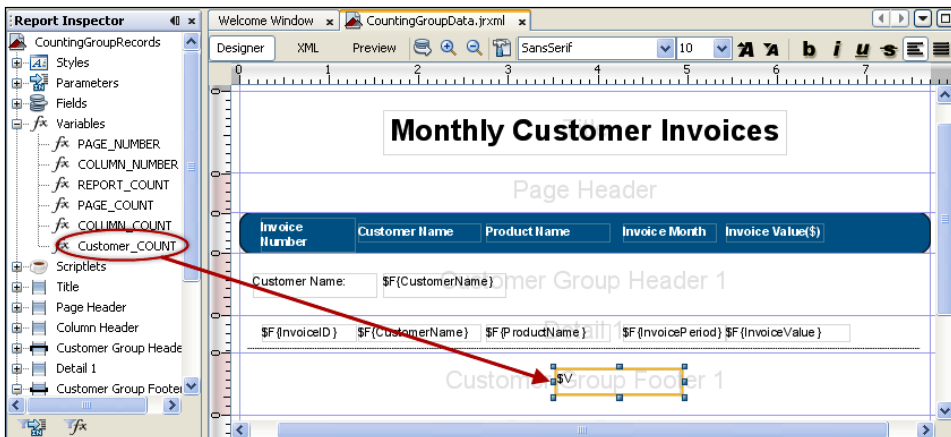
17. Switch to the **Preview** tab, you will see that a customer name is displayed as the heading of each group of records in your report, as shown encircled in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	8875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	40.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1007	Alice	JasperReport	Apr09	40.5

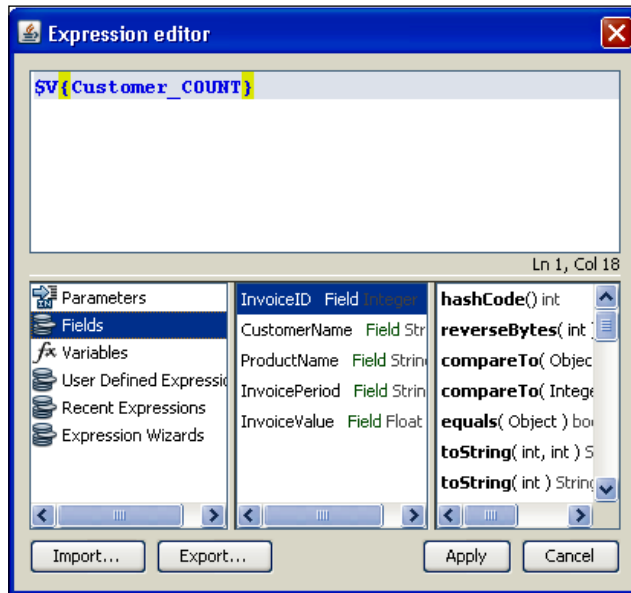
18. Switch back to the **Designer** tab. Double-click on the **Variables** node in the **Report Inspector** window. It will open showing a list of variables, as shown next. Compare the variables you saw last time in the screenshot of step 3 with the variables you now see. This time you will see that a variable named **Customer_COUNT** has been added. This is a group-specific variable, which holds the total number of records in the customer group you added to your reports in steps 4 to 8.



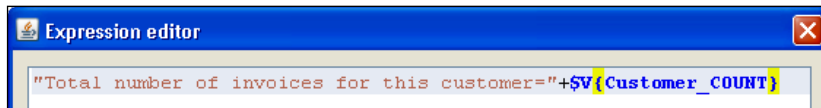
19. Drag-and-drop the `Customer_COUNT` variable from the variables list into the **Customer Group Footer 1** section, as shown in the following screenshot:



20. Right-click on the **`$V{Customer_COUNT}`** field and select the **Field Expression** option from the pop-up menu. An **Expression editor** window will open.



21. Type "Total number of invoices for this customer=" in the **Expression editor** window just before the **`$V{Customer_COUNT}`** expression. The complete expression in the editor will become **"Total number of invoices for this customer=" + `$V{Customer_COUNT}`**, as shown in the following screenshot. Click the **Apply** button.



22. While the **`$V{Customer_COUNT}`** field is selected, find the **Width** property in the **Properties** window and set 200 as its value. Now the text field will show almost the complete expression.
23. Find the **Expression Class** property under the **Text field properties** section in the **Properties** window and select `java.lang.String` as its value.

24. Switch to the **Preview** tab and you will see that the total number of records for each customer appears just after the invoices for each customer, as shown encircled in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4652.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Total number of invoices for this customer=6				
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1007	Alice	JasperReport	Apr09	49.5
Total number of invoices for this customer=3				

How it works...

In this recipe, you have displayed the total number of invoices for a particular customer. In order to do this, you created a group based on customer names in steps 4 to 8.

Refer to the *Inserting a heading for a group of records* recipe of *Chapter 2*, which discusses how to create groups.

When you created the group, iReport automatically generated a group-specific variable named **Customer_COUNT** to hold the number of records for that particular customer. Finally, in step 19, you dropped the **Customer_COUNT** variable into your report to display how many invoices exist for each customer.

Refer to the following JRXML code, which shows a CDATA section highlighted. This CDATA section contains the `Customer_COUNT` variable, which you dragged-and-dropped in step 19 of the recipe. You can guess that **iReport** authored this CDATA section in response to your action in step 19.

```
<groupFooter>
  <band height="50">
    <textField>
      <reportElement x="255" y="14" width="200" height="20"/>
      <textElement/>
      <textFieldExpression class="java.lang.String">
        <![CDATA["Total number of invoices for this
          customer="+$V{Customer_COUNT}]]>
      </textFieldExpression>
    </textField>
  </band>
</groupFooter>
```

Finding the maximum of a particular field

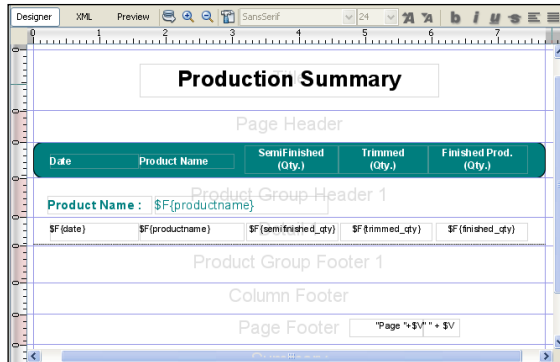
This recipe takes sample production data and evaluates the maximum production figure. It also finds the date when production was at a maximum level. While doing all this, it uses a combination of expressions, which you can use in a large variety of your business reports.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS_CH4_Rcp4.txt`, which helps you to create a database named `jasperdb4` with a table named `ProductionSummary_Ch4` with six columns (`Seq_ID`, `ProductName`, `Date`, `SemiFinished_Qty`, `Trimmed_Qty`, and `Finished_Qty`) and copy sample data of this recipe into the table.

How to do it...

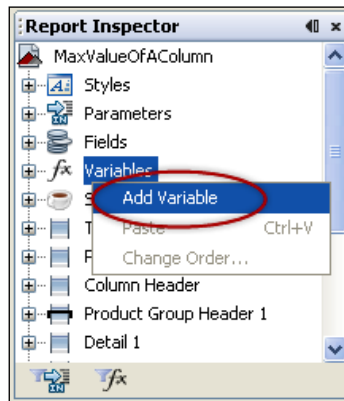
1. Open the `MaxValueOfAColumn.jrxml` file from the `Task4` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header 1**, **Detail 1**, and **Page Footer** sections, as shown in the following screenshot:



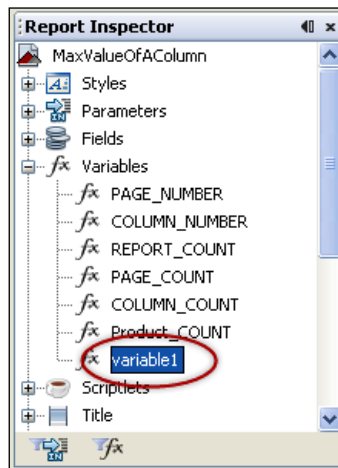
2. Switch to the **Preview** tab and you will see a production summary report, as shown in the following screenshot:

Production Summary				
Date	Product Name	Semi Finished (Qty.)	Trimmed (Qty.)	Finished Prod. (Qty.)
Product Name : Coca Cola Can				
18/07/2009	Coca Cola Can	1400	0	0
19/07/2009	Coca Cola Can	0	0	200
22/07/2009	Coca Cola Can	0	4050	0
20/07/2009	Coca Cola Can	0	3995	0
19/07/2009	Coca Cola Can	4000	0	0
18/07/2009	Coca Cola Can	0	2555	0
19/07/2009	Coca Cola Can	0	2000	0
22/07/2009	Coca Cola Can	0	0	4050
19/07/2009	Coca Cola Can	0	0	2800
18/07/2009	Coca Cola Can	2500	0	0
Product Name : Inner Assembly				
10/07/2009	Inner Assembly	400	0	0
14/07/2009	Inner Assembly	1000	0	0
14/07/2009	Inner Assembly	0	0	1246
Page 1 of 4				

3. Switch back to the **Designer** tab. Right-click on the **Variables** node in the **Report Inspector** window on the left side of your report. A pop-up menu will appear. Select the **Add Variable** option.

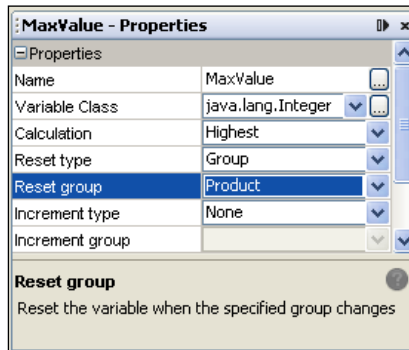


4. A new variable named **variable1** will be added at the end of the variables list, as shown in the following screenshot:

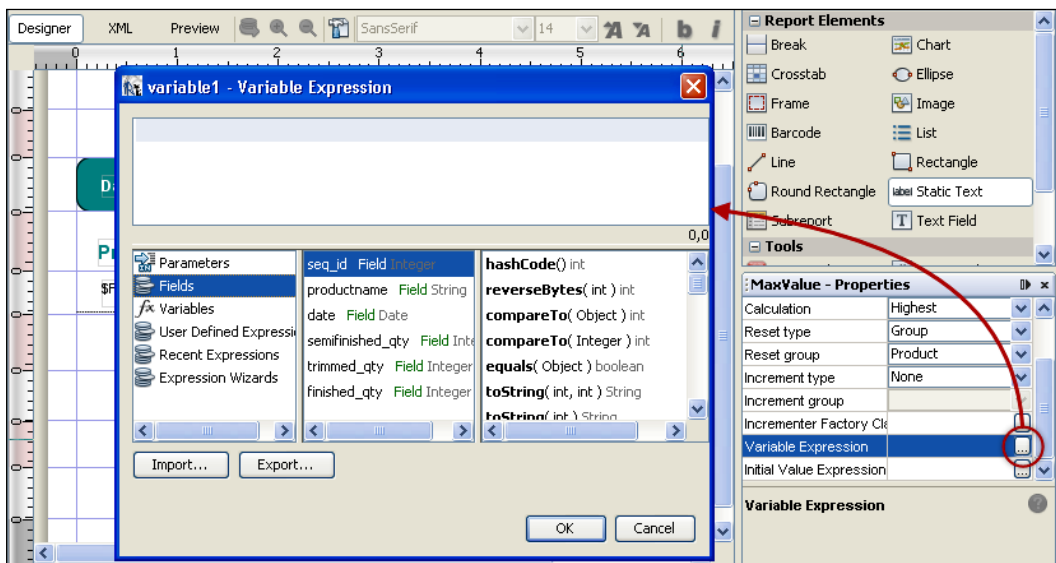


5. While **variable1** is selected, find the **Name** property in the **Properties** window below the palette of components and change its value to **MaxValue**. Now the name of the **variable1** variable will change to **MaxValue**.
6. Select the **Variable Class** property and change its value to **java.lang.Integer**.
7. Select the **Calculation** property and change its value to **Highest**.
8. Select the **Reset Type** property and change its value to **Group**.

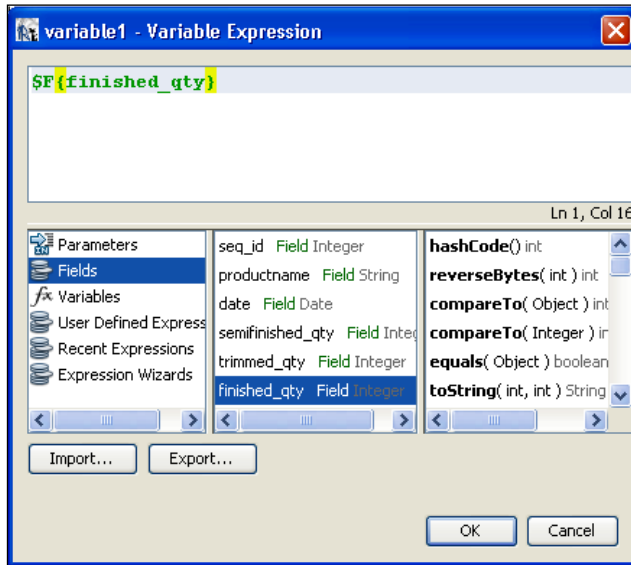
9. Select the **Reset Group** property and change its value to `Product`, as shown in the following screenshot:



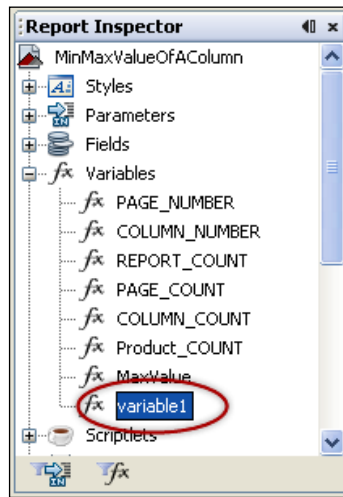
10. Select the **Variable Expression** property and click on the button beside it. A **Variable Expression** window with no default expression will open, as shown in the following screenshot:



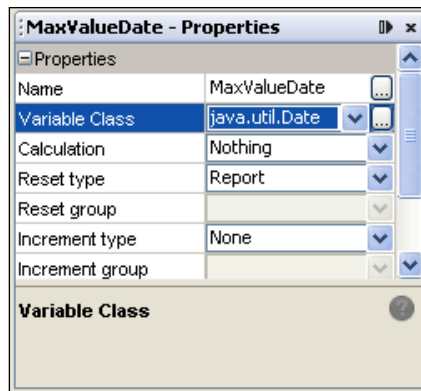
11. Look for the **finished_qty** field in the second column of the lower half of the **Variable Expression** window and double-click on it. The `$F{finished_qty}` expression will be added to the expression window. Click **OK**, as shown in the following screenshot:



12. Right-click on the **Variables** node in the **Report Inspector** window on the left-side of your report. A pop-up menu will appear. Select the **Add Variable** option.
13. A new variable named **variable1** will be added at the end of the variables list, as shown in the following screenshot:



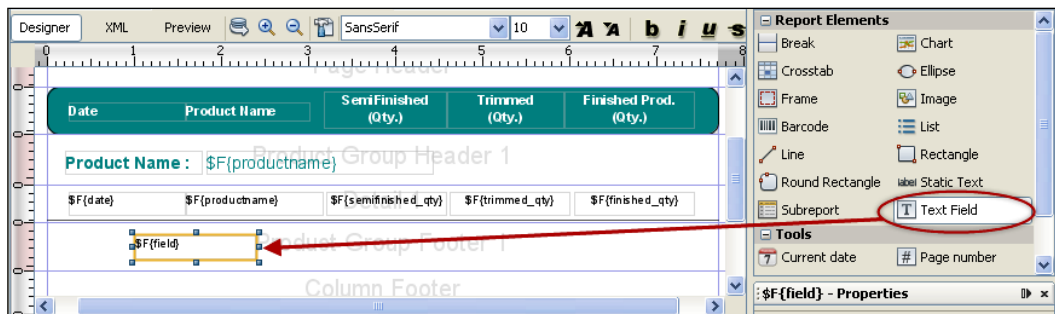
14. While **variable1** is selected, find the **Name** property in the **Properties** window below the palette of components and change its value to **MaxValueDate**. Now the name of the **variable1** variable will change to **MaxValueDate**.
15. Select the **Variable Class** property and change its value to `java.util.Date`, as shown in the following screenshot:



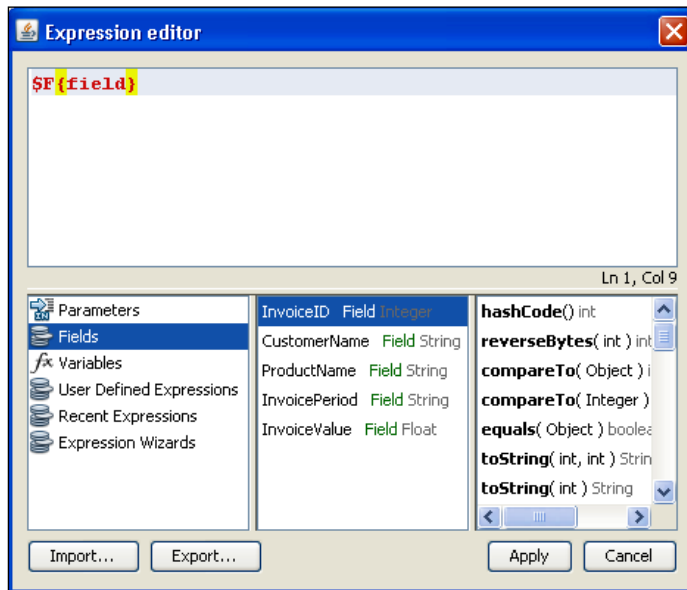
16. Select the **Variable Expression** property and click the button beside it. A **Variable Expression** window with no default expression will open.
17. Type `($V{MaxValue}.intValue() == $F{finished_qty}.intValue() ? $F{date} : $V{MaxValueDate})` in the expression editor window, as shown next. Click **OK**.



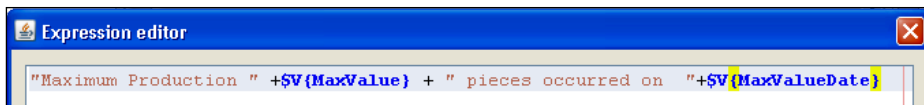
18. Drag-and-drop a **Text Field** component into the **Product Group Footer 1** section, as shown in the following screenshot:



19. Right-click on the newly dropped **Text Field** component in the **Product Group Footer 1** section, and a pop-up menu will appear. Select the **Edit expression** option. An **Expression editor** window will open, as shown in the following screenshot:



20. Delete the default expression and type "Maximum Production " + \$V{MaxValue} + " pieces occurred on " + \$V{MaxValueDate} in the expression editor window, as shown in the following screenshot: Click the **Apply** button.



21. While the **Maximum Production** text field is selected, find the **Width** property in the **Properties** window and set 340 as its value. This will open a text field view area.

22. Switch to the **Preview** tab and you will see maximum production is shown at the end of each product in your report, as shown encircled in the following screenshot:

Production Summary				
Date	Product Name	Semi Finished (Qty.)	Trimmed (Qty.)	Finished Prod. (Qty.)
Product Name : Coca Cola Can				
18/07/2009	Coca Cola Can	1400	0	0
19/07/2009	Coca Cola Can	0	0	200
22/07/2009	Coca Cola Can	0	4050	0
20/07/2009	Coca Cola Can	0	3995	0
19/07/2009	Coca Cola Can	4000	0	0
18/07/2009	Coca Cola Can	0	2555	0
19/07/2009	Coca Cola Can	0	2000	0
22/07/2009	Coca Cola Can	0	0	4050
19/07/2009	Coca Cola Can	0	0	2800
18/07/2009	Coca Cola Can	2500	0	0
Maximum Production 4050 pieces occurred on 2009-07-22				
Product Name : Inner Assembly				
10/07/2009	Inner Assembly	400	0	0
14/07/2009	Inner Assembly	1000	0	0
14/07/2009	Inner Assembly	0	0	1245
Maximum Production 1245 pieces occurred on 2009-07-14				

Page 1 of 4

How it works...

In this recipe you have achieved two things. First, you have evaluated the maximum of the `finished_qty` column. Second, you have found the date when maximum production occurred.

In order to evaluate the maximum, you simply used the `Highest` built-in function of JasperReports in step 7.

Finding the date when maximum production occurred is a bit tricky and requires writing an expression shown in step 17 (`($V{MaxValue}.intValue() == $F{finished_qty}.intValue() ? $F{date} : $V{MaxValueDate})`). This is an IF-THEN-ELSE statement expression, which evaluates the `MaxValueDate` variable.

The IF part (`($V{MaxValue}.intValue() == $F{finished_qty}.intValue())`) checks the value of the `finished_qty` column of each record and compares it with the `MaxValue` variable, which stores the maximum production value. If this condition is true for a record, then it means the record is the maximum production record, whose date you want to pick up and store in `MaxValueDate`. Therefore, the THEN part (`$F{date}`) of the expression simply assigns the date value of this record to the `MaxValueDate` variable.

On the other hand, if this condition is not true, then the ELSE part (`$V{MaxValueDate}`) of the expression simply maintains the previous value of the `MaxValueDate` variable.

Changing background color of alternate records

Use of appropriate background colors improves the readability of your reports. This recipe teaches you how to generate white and colored backgrounds for alternate records shown in a report.

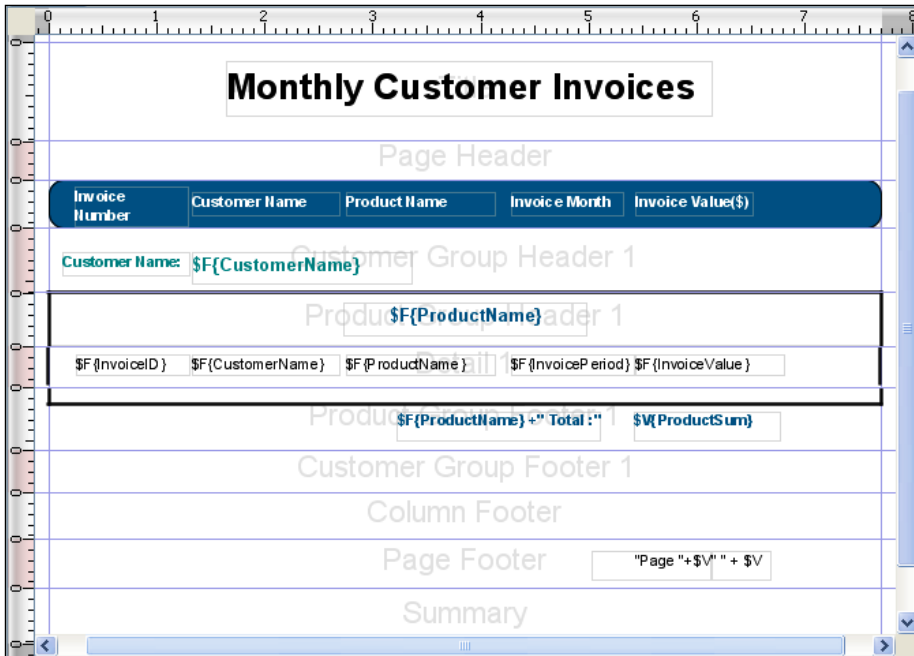
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

Let me show you how you can set different background colors for alternate records in a report:

1. Open the `ChangeBackgroundColor.jrxml` file from the `Task5` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header 1**, **Detail 1**, and **Page Footer** sections, as shown in the following screenshot:

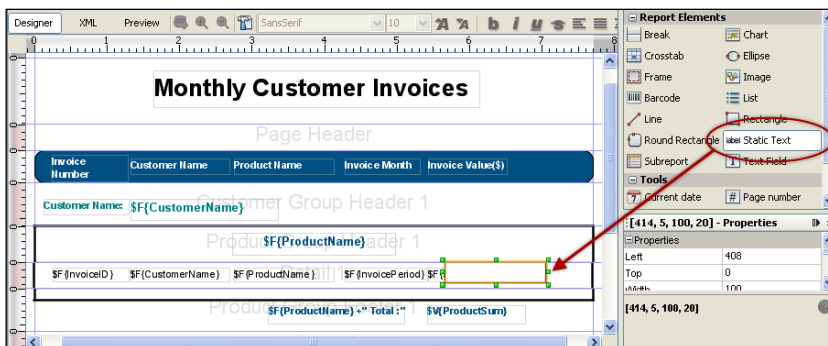


- Switch to the **Preview** tab; you will see that the report contains lists of records grouped by customer names, as shown in the following screenshot:

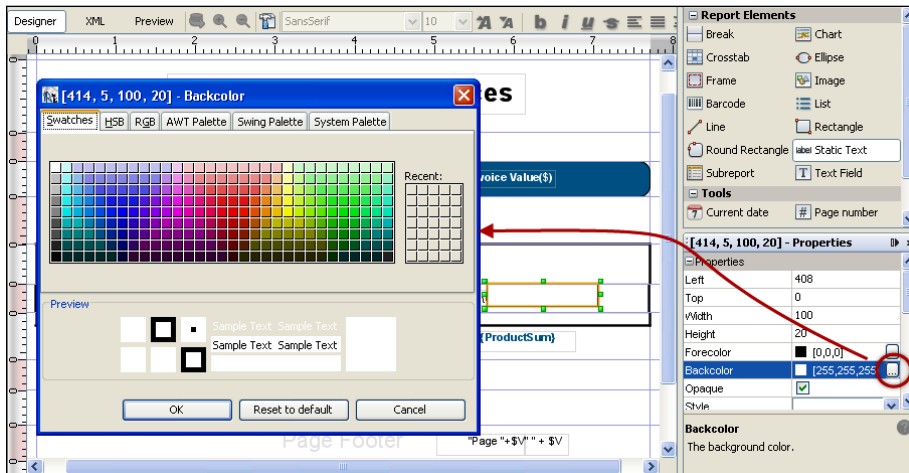
Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
Customer Name: Bob				
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0

Page 1 of 3

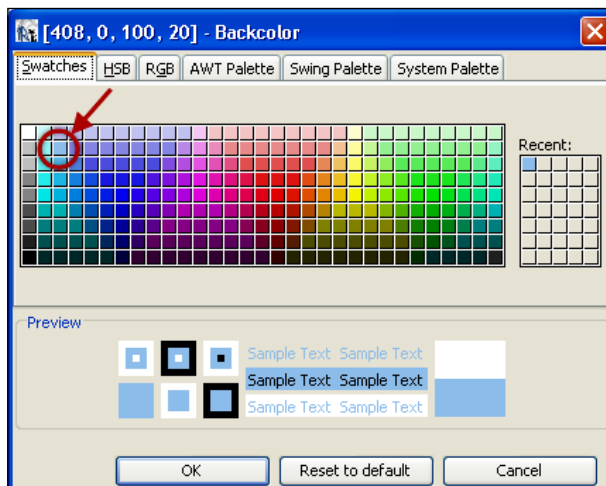
- Switch back to the **Design** tab and drag-and-drop a **Rectangle** component from the **Palette** into the **Detail 1** section. While dropping the rectangle, make sure that the upper side of the rectangle touches the blue line that separates the **Detail 1** and **Customer Group Header 1** sections, as shown in the following screenshot:



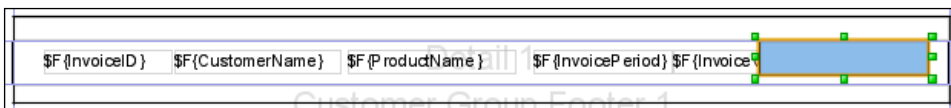
- Click on the **Rectangle** component inside the **Detail 1** section, and its properties will appear in the **Properties** window below the palette of components. Find the **BackColor** property and click on the button beside it. A color selector window will open as shown in the following screenshot:



- Set light blue as the background color and click **OK**, as shown in the following screenshot:



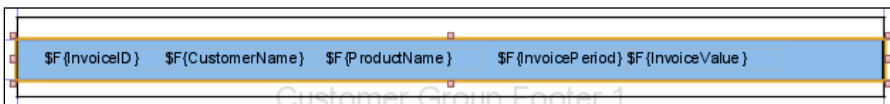
- Now the **Rectangle** will look like the following:



7. Select the **Rectangle** component inside the **Detail 1** section. Right-click on it, and a pop-up menu will appear. Select the **Size** option, and a sub-menu will open. Choose the **Adapt to parent** option from the sub-menu. The size of the rectangle will become equal to its parent **Detail 1** section and it will completely hide all components of the **Detail 1** section, as shown in the following screenshot:



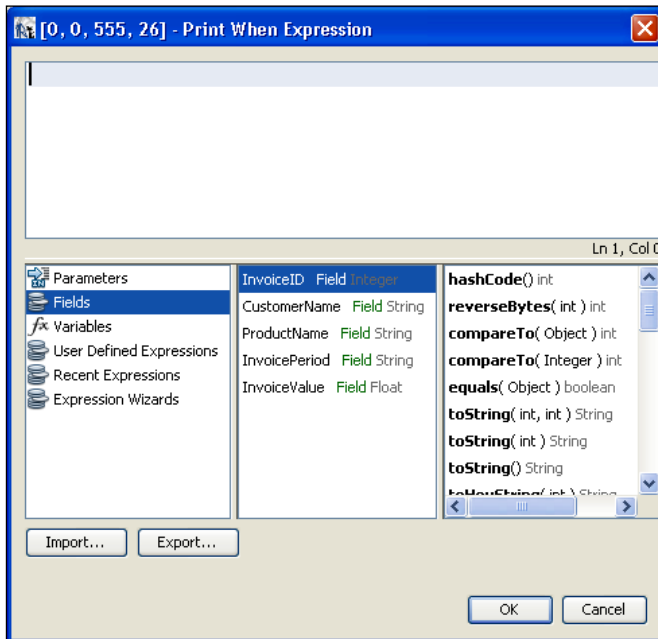
8. Right-click on the selected **Rectangle** component in the **Detail 1** section, and a pop-up menu will appear. Select the **Send to Back** option. You will see that the rectangle will become the background so that other components placed in the **Detail 1** section are now visible, as shown in the following screenshot:



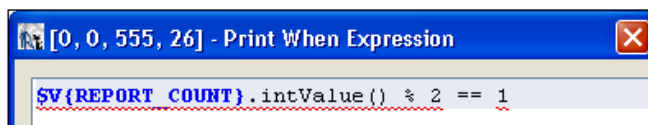
9. Switch to the **Preview** tab and you will see that a colored background is printed with each invoice record, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Customer Name: Alice				
1038	Alice	Jasper Report	Mar09	49.5
1007	Alice	Jasper Report	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
Customer Name: Bob				
1006	Bob	Jasper Reports	Mar09	50.0
1037	Bob	Jasper Reports	Mar09	50.0
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0

10. Switch back to the **Designer** tab, and select the **Rectangle** component in the **Detail 1** section. Its properties will appear in the **Properties** window. Find the **Print When Expression** property and click on the button beside it. A **Print When Expression** window will open, as shown in the following screenshot:



11. Type the expression `$V{REPORT_COUNT}.intValue() % 2 == 1` in the expression editor window, as shown in the following screenshot. Click the **OK** button.



12. Switch to the **Preview** tab and you will see a colored background printed with alternate invoice records, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Padding Material	Jun09	8745.5
1013	ABC Publishing	Padding Material	Aug09	9852.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1007	Alice	JasperReport	Apr09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
Customer Name: Bob				
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0

Page 1 of 3

How it works...

You have created changing background colors for alternate records in your report. For this purpose, you dropped a rectangle in step 3 and authored a **Print When Expression** property for the rectangle in step 11 of this recipe. The JRXML code for the rectangle looks like the following:

```
<rectangle>
  <reportElement x="0" y="0" width="555" height="26"
    bgcolor="#99CCFF">
    <printWhenExpression>
```

```

        <![CDATA[$V{REPORT_COUNT}.intValue() % 2 == 1
    </printWhenExpression>
</reportElement>
</rectangle>

```

The expression `$V{REPORT_COUNT}.intValue() % 2 == 1` for the **Print When Expression** property returns `TRUE` for alternate records and, therefore, the rectangle appears for alternate records. This creates the impression that alternate records have different background colors.

Applying styles to your data based on a logical or mathematical condition

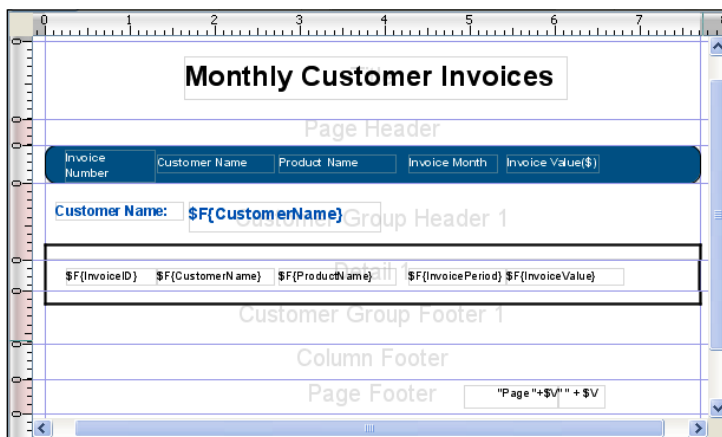
This recipe teaches you how to create and apply formatting styles to data that satisfies logical or mathematical conditions.

Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download of this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you to create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

1. Open the `ApplyStyleOnAValue.jrxml` file from the `Task6` folder of the source code for this chapter. The **Designer** tab of iReport shows a report containing data in the **Title**, **Column Header**, **Customer Group Header 1**, **Detail 1**, and **Product Group Footer 1** sections, as shown in the following screenshot:

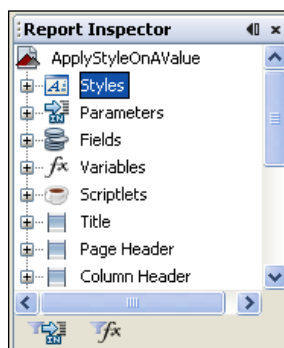


- Switch to the **Preview** tab, you will see that the report contains lists of records grouped by customer names, as shown in the following screenshot:

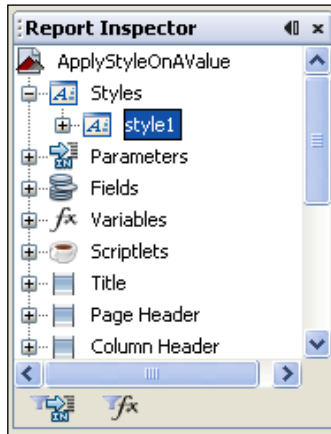
Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1007	Alice	JasperReport	Apr09	49.5
Customer Name: Bob				
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0

Page 1 of 3

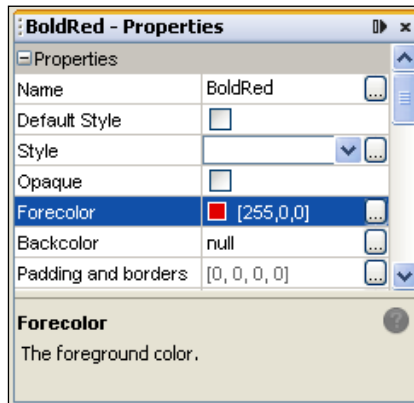
- Switch to the **Design** tab. You will see a **Report Inspector** window on the left-most part of the main iReport screen, which shows a tree with many nodes. The first node is named **Styles**, as shown in the following screenshot:



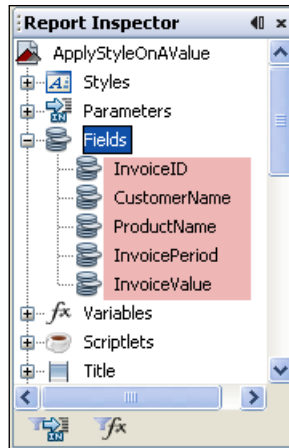
- Right-click on the **Styles** node, select **Add** from the pop-up menu, and a sub-menu will open. Choose **Style** from the sub-menu. A new style named **style1** will be added to the **Styles** list, as shown in the following screenshot:



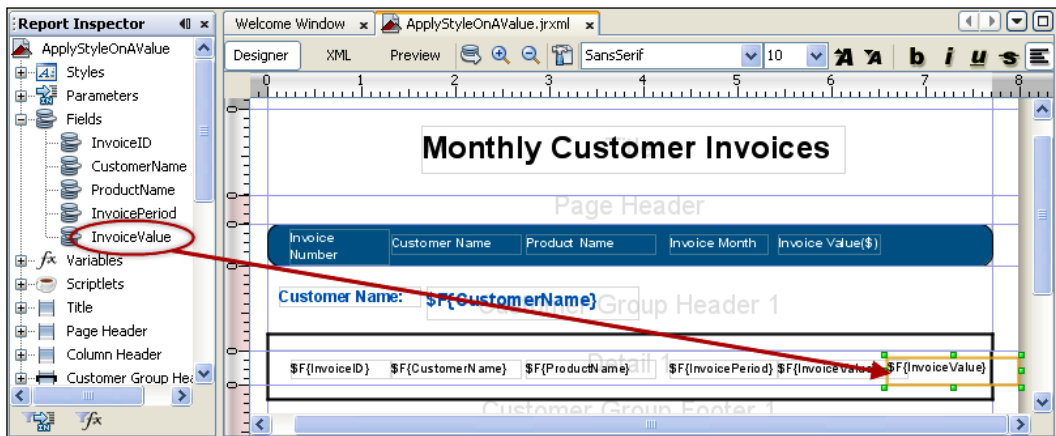
- While **style1** is selected, find the **Name** property in the **Properties** window below the palette of components on the left side of the main iReport window. Change the value of the **Name** property to **BoldRed**. Now the name of the **style1** style will change to **BoldRed**.
- Select the **Forecolor** property of the **BoldRed** style and choose the color red, as shown in the following screenshot:



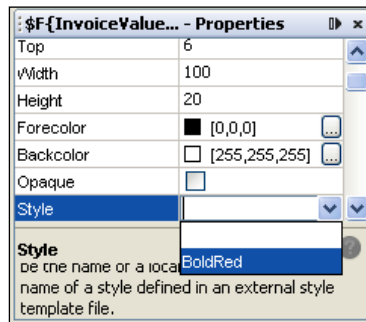
7. Find the **Bold** property of the **BoldRed** style and check the checkbox beside it. Leave the other properties at their default values.
8. Double-click on the **Fields** node in the **Report Inspector** window. The **Fields** node will open showing all available fields from the data source, as shown highlighted in the following screenshot:



9. Drag-and-drop the **InvoiceValue** field from the **Fields** node of the tree into the **Detail 1** section. While dropping the field, make sure that the newly dropped **InvoiceValue** field instance sits adjacent to the right of the existing **InvoiceValue** field in the **Detail 1** section, as shown in the following screenshot:



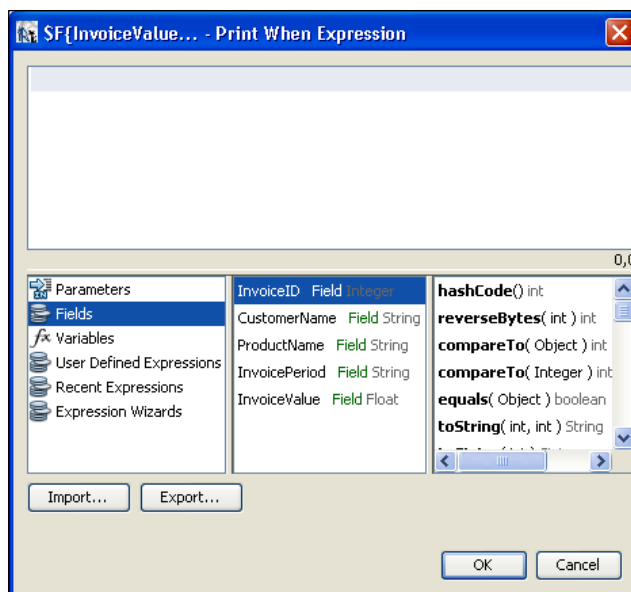
10. Select the newly dropped **InvoiceValue** field into the **Detail 1** section; its properties will appear in the **Properties** window below the **Palette**. Find the **Style** property and select **BoldRed** as its value, as shown in the following screenshot:



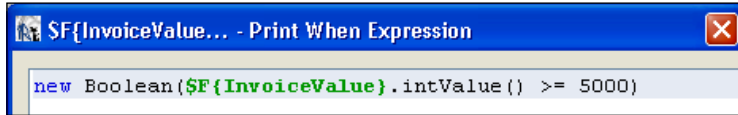
11. You will notice that the appearance of the newly dropped **InvoiceValue** text field is changed, as following:



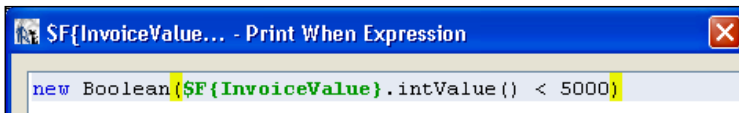
12. Find the **Print When Expression** property of the newly-dropped **InvoiceValue** text field and click on the button beside it. A **Print When Expression** window with no default expression will open, as shown in the following screenshot:



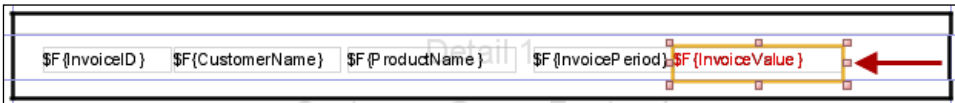
13. Type the new `Boolean($F{InvoiceValue}.intValue() >= 5000)` expression in the expression window, as shown in the following screenshot. Click **OK**.



14. Select the existing **InvoiceValue** text field in the **Detail 1** section, its properties will appear in the **Properties** window below the **Palette**. Find the **Print When Expression** property and click the button beside it. A **Print When Expression** window with no default expression will open.
15. Type a new `Boolean($F{InvoiceValue}.intValue() < 5000)` expression in the expression window, as shown in the following screenshot. Click **OK**.



16. Select the new **InvoiceValue** text field that you dropped earlier in step 9. Using the arrow key, move the text field towards the left until it sits exactly over the existing **InvoiceValue** text field, as shown in the following screenshot:



17. Switch to the **Preview** tab; you will notice that the color of invoices with values greater than or equal to 5000 is changed to red, as shown encircled in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
Customer Name: ABC Publishing				
1012	ABC Publishing	Offset Paper	Jun09	6875.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1020	ABC Publishing	Printing Ink	Mar09	3450.0
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
Customer Name: Alice				
1038	Alice	JasperReport	Mar09	49.5
1014	Alice	WordPress Cookbook	Jan09	45.0
1007	Alice	JasperReport	Apr09	49.5
Customer Name: Bob				
1018	Bob	Printing Ink	Sep09	150.0
1015	Bob	WordPress Cookbook	May09	45.0
1006	Bob	JasperReports	Mar09	50.0
1037	Bob	JasperReports	Mar09	50.0

Page 1 of 3

How it works...

You have played with two instances of the **InvoiceValue** text field in this recipe. You associated the **BoldRed** style with the newly dropped **InvoiceValue** text field in step 10. You did not associate any particular style with the existing **InvoiceValue** text field.

Then, in step 13, you authored an expression for the **Print When Expression** property of the newly dropped **InvoiceValue** text field. The expression was **new Boolean(\$F{InvoiceValue}.intValue() >= 5000)**, which means this new **InvoiceValue** text field will be printed only when the invoice value is greater than or equal to 5000. Similarly, you authored the expression **(new Boolean(\$F{InvoiceValue}.intValue() < 5000))** for the existing **InvoiceValue** text field in step 15, which prints the existing field when the invoice value is less than 5000.

A combined effect of this style + **Print When Expression** strategy is that all invoice values greater than or equal to 5000 are displayed in **BoldRed** style.

Changing the background color of a particular record based on a mathematical or logical expression

You will often need to highlight some specific types of records in your report. For example, you may want to highlight all invoices for a particular product. This recipe teaches you how to do that.

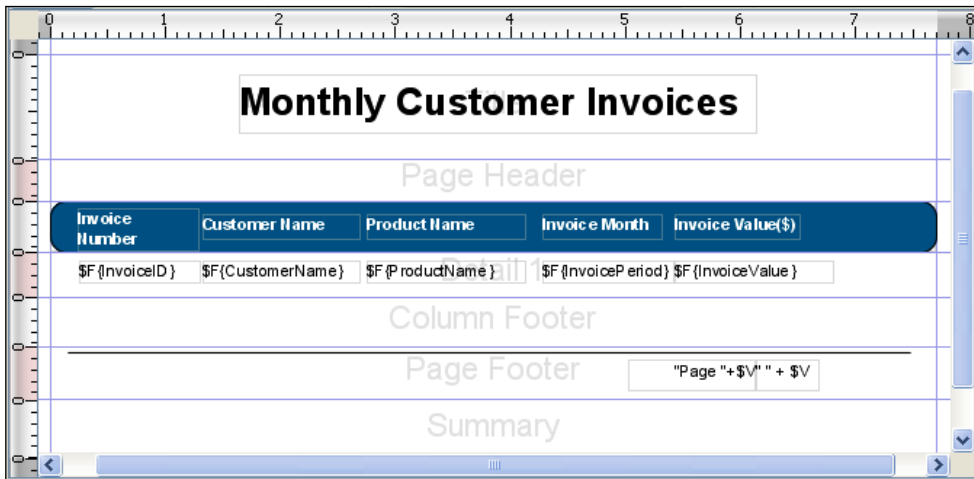
Getting ready

Refer to the `installPostgreSQL.txt` file included in the source code download for this chapter to install and run PostgreSQL, which should be up and running before you proceed. The source code for this chapter also includes a file named `copySampleDataIntoPGS.txt`, which helps you create a database named `jasperdb4` and copy sample data for this recipe into the database.

How to do it...

The following twelve steps demonstrate how to change the background color of a particular type of record during report processing:

1. Open the `ChangeBackgroundConditionally.jrxml` file from the `Task7` folder of the source code for this chapter. The **Designer** tab of iReport shows a simple report with data in the **Title**, **Column Header**, **Detail 1**, and **Page Footer** sections, as shown in the following screenshot:

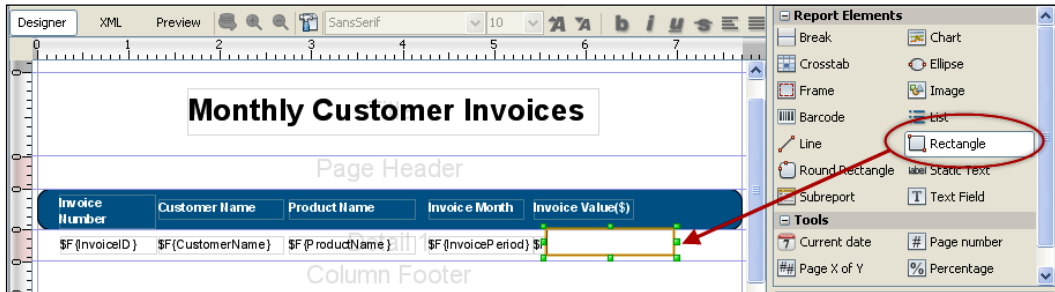


2. Switch to the **Preview** tab, and you will see a list of invoices in tabular form, as shown in the following screenshot:

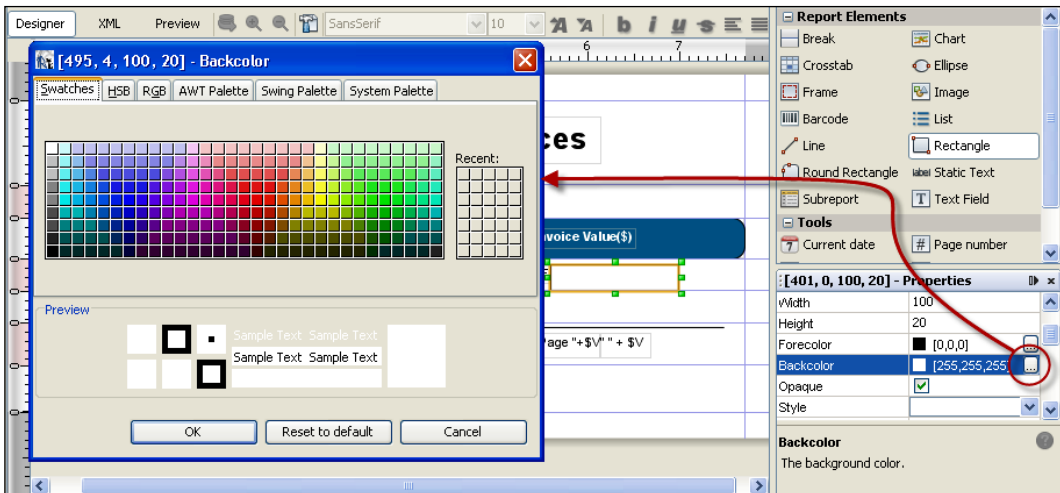
Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1005	Bob	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1037	Bob	JasperReports	Mar09	50.0
1038	Alice	JasperReport	Mar09	49.5
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1010	ABCPublishing	Offset Paper	Feb09	4552.2
1011	ABCPublishing	Packing Material	Jun09	8745.5
1013	ABCPublishing	Packing Material	Aug09	9852.0
1014	Alice	WordPress Cookbook	Jan09	45.0
1015	Bob	WordPress Cookbook	May09	45.0
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1042	Packt Publishing	Printing link	Mar09	8500.0
1016	ABCPublishing	Printing link	Jun09	2440.0
1017	Packt Publishing	Printing link	Aug09	6540.0
1018	Bob	Printing link	Sep09	150.0

Page 1 of 3

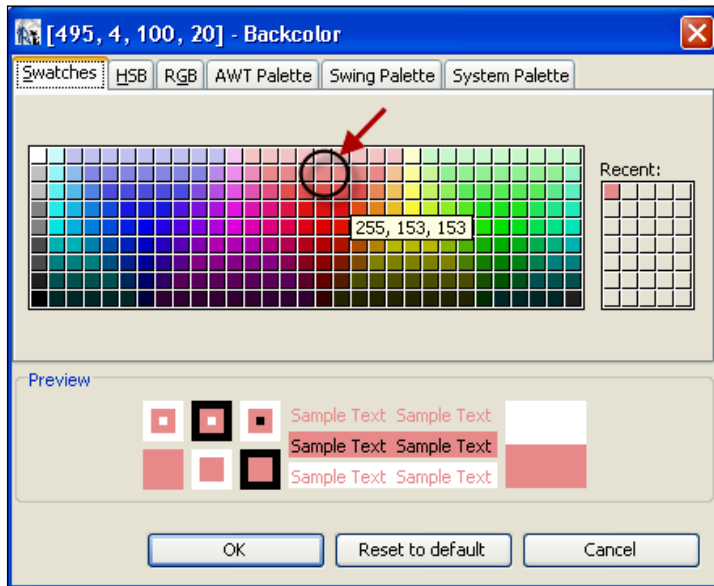
- Switch back to the **Design** tab, and drag-drop a **Rectangle** component from the **Palette** of components on the right of the main iReport window into the **Detail 1** section. While dropping the rectangle, make sure that the upper side of the rectangle touches the blue line that separates the **Detail 1** and **Column Header** sections, as shown in the following screenshot:



- Click the **Rectangle** component inside the **Detail 1** section; its properties will appear in the **Properties** window below the palette of components. Find the **Backcolor** property and click the button beside it. A color selector window will open, as shown in the following screenshot:



5. Set a light pink background color and click **OK**, as shown in the following screenshot:



6. Find the **Forecolor** property and click the button beside it. A color selector window will open as per step 4 of this recipe. Choose white as the **Forecolor** value by selecting the color rectangle at the top left of the color selector window shown in step 5. Click **OK**.
7. Right-click on the **Rectangle** component inside the **Detail 1** section, and a pop-up menu will appear. Select the **Size** option, and a sub-menu will open. Choose the **Adapt to parent** option from the sub-menu. The size of the rectangle will become equal to its parent **Detail 1** section and the rectangle will completely hide all components in the **Detail 1** section, as shown in the following screenshot:



8. Again, right-click on the selected **Rectangle** component in the **Detail 1** section; a pop-up menu will appear. Select the **Send to Back** option. You will notice that the rectangle will become the background so that other components placed in the **Detail 1** section are now visible, as shown in the following screenshot:



Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
<code>{InvoiceID}</code>	<code>{CustomerName}</code>	<code>{ProductName}</code>	<code>{InvoicePeriod}</code>	<code>{InvoiceValue}</code>

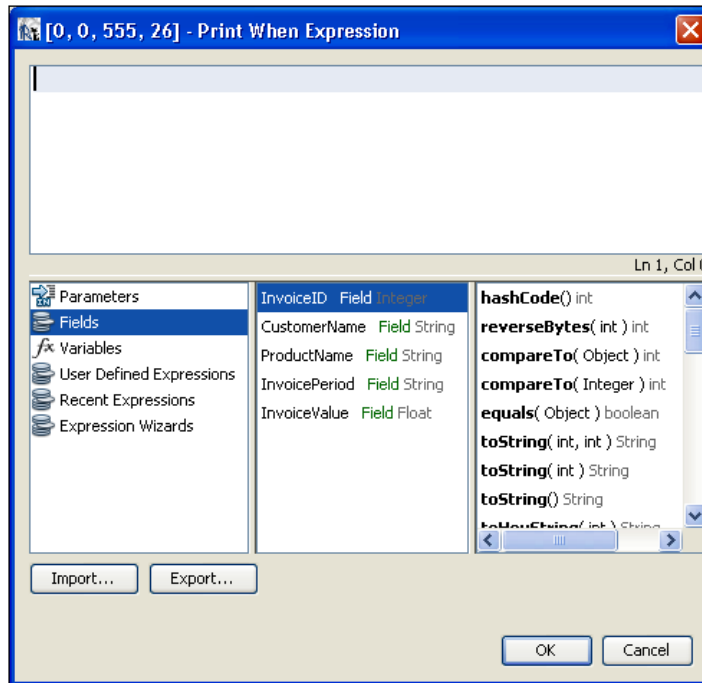
9. Switch to the **Preview** tab and you will see background is printed with each invoice record, as shown in the following screenshot:

Monthly Customer Invoices

Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	Packing Material	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	Packing Material	Mar09	2050.0
1006	Bdb	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1037	Bdb	JasperReports	Mar09	50.0
1038	Alice	JasperReport	Mar09	49.5
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	Packing Material	Jun09	8745.5
1013	ABC Publishing	Packing Material	Aug09	9852.0
1014	Alice	WordPress Cookbook	Jan09	45.0
1015	Bdb	WordPress Cookbook	May09	45.0
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1042	Packt Publishing	Printing Ink	Mar09	8500.0
1016	ABC Publishing	Printing Ink	Jun09	2440.0
1017	Packt Publishing	Printing Ink	Aug09	6540.0
1018	Bdb	Printing Ink	Sep09	150.0
1019	Packt Publishing	Printing Ink	Jan09	5240.0
1020	ABC Publishing	Printing Ink	Mar09	3450.0

Page 1 of 3

10. Switch back to the **Designer** tab; select the **Rectangle** component in the **Detail 1** section, and its properties will appear in the **Properties** window. Find the **Print When Expression** property and click the button beside it. A **Print When Expression** window will open, as shown in the following screenshot:



11. Type new Boolean (\$F{ProductName} == 'Offset Paper') in the expression editor window, as shown in the following screenshot. Click **OK**.



12. Switch to the **Preview** tab and you will see that all invoices for offset paper are printed with a light pink background, as shown in the following screenshot:

Monthly Customer Invoices				
Invoice Number	Customer Name	Product Name	Invoice Month	Invoice Value(\$)
1001	Packt Publishing	PackingMaterial	Mar09	5020.25
1002	Packt Publishing	Offset Paper	Apr09	3000.5
1003	Packt Publishing	Offset Paper	Mar09	4150.05
1004	Packt Publishing	PackingMaterial	Mar09	2050.0
1006	Bdb	JasperReports	Mar09	50.0
1007	Alice	JasperReport	Apr09	49.5
1008	Packt Publishing	Offset Paper	Jan09	8058.5
1009	Packt Publishing	Offset Paper	Feb09	5085.0
1037	Bdb	JasperReports	Mar09	50.0
1038	Alice	JasperReport	Mar09	49.5
1039	Packt Publishing	Offset Paper	Mar09	8058.5
1010	ABC Publishing	Offset Paper	Feb09	4552.2
1011	ABC Publishing	PackingMaterial	Jun09	8745.5
1013	ABC Publishing	PackingMaterial	Aug09	9852.0
1014	Alice	WordPress Cookbook	Jan09	45.0
1015	Bdb	WordPress Cookbook	May09	45.0
1040	Packt Publishing	Offset Paper	Mar09	5085.0
1041	Packt Publishing	Offset Paper	Mar09	5085.0
1042	Packt Publishing	Printing link	Mar09	8500.0
1016	ABC Publishing	Printing link	Jun09	2440.0
1017	Packt Publishing	Printing link	Aug09	6540.0
1018	Bdb	Printing link	Sep09	150.0
1019	Packt Publishing	Printing link	Jan09	5240.0
1020	ABC Publishing	Printing link	Mar09	3450.0

Page 1 of 3

Index

Symbols

`{CustomerName}` 118
`{FeederID}` expression 326
`{InvoiceID}-Properties` window 145
`{InvoiceID}` expression 125
`{InvoicePeriod}` expression 131
`{InvoiceValue}.intValue() >=`
 `{MinInvoiceValue}` 71
`{InvoiceValue}` expression 51, 132
`{InvoiceValue}` text field 49
`{ProductName}` expression 107, 131
`{JASPER_REPORT}.getName()` 83
`{PAGE_NUMBER}` text field 200
`{TOC_PageIndex}` text field 215, 218
`{TOC}` text field 208
`` tag 135
`<band>` child 50
`<band>` tag 39, 122
`<columnFooter>` tag 58
`<columnHeader>` tag 58
`<detail>` tag 50, 58, 122
`` tag 19
`<group>` tag 122, 226
`<groupExpression>` tag 227
`<image>` tag 28
`<imageExpression>` tag 28
`<jasperReport>` tag 39
`` tag 134
`<pageHeader>` tag 14, 33
`<parameter>` tag 25
`<printWhenExpression>` tag 296
`<queryString>` tag 25, 33, 50
`<rectangle>` tag 122
`<reportElement>` child 122
`<reportElement>` tag 18, 39

`<staticText>` tag 18, 25
`<subreport>` tag 296
`<subreportParameter>` tag 296
`<text>` tag 19
`<textField>` tag 33, 51, 90, 140, 227
`<textFieldExpression>` tag 25, 51
`<title>` tag 14, 18
2DPageCount.jrxml file 228

A

Adapt to Parent option 112, 120
Adapt to parent option 323
Adapt to Parent width option 109
Add Another Detail Band option 245
Add node as field option 166
Add Parameter option 172, 307
Add Variable option 203, 288
Align option 109
Align To Bottom Margin option 109
Align To Right Margin 198

B

backcolor attribute 122
background color
 setting, for data 114-122
 setting, for report 114-122
background images
 using, in report 146-148
Band height property 175, 197
bar graph
 embedding, inside tabular view 331-341
BigBoldRed (reference) style 105, 106
BillingMonth parameter 310
Blank When Null property 49

Break component 200, 209

Bring to Front option 118

C

Calculation property 204, 288

Chart component 333

class attribute 28

Column Header section 55, 298

com.CustomerInvoice class 192

com.CustomerInvoicesFactory class 187

compile.bat file 356

compileJRXMLFile() file 346

compileJRXMLFile() method 346

connect2DB() method 346

Connection object 347

copySampleDataIntoPGS.txt file 240

cover page

building, for multi-page report 196-201

crosstab

creating 315-321

Current Date component 198

CustHistorySubreport.jrxml subreport 254, 259

Customer Group Footer 1 section 224

Customer Group Header 1 section 62, 118, 224

CustomerName parameter 21, 243, 247, 286

D

data trends

displaying, as graph 321-330

Default Value Expression property 173, 307

Designer tab 136

design preview, iReport

column footer 42

column header 42

detail 1 42

group footer 1 42, 43

group header 1 42, 43

Detail 1 section 55

dynamic title, report

creating, steps 21-24

using 20

working 24, 25

E

Edit expression 128

evaluationTime attribute 227

Evaluation Time property 208

Excel report

creating, Java Swing application used 361-365

Expression Class property 127

Expression editor 126

Expression editor option 134

Expression editor window 46

Expression field 243

F

factory class 187

FeederSummaryReport.jrxml report 314

field

expanding vertically, to accomodate large data 135-140

Fields node 299

FillManager class 347

fillReport() method 346, 347

FirstRecordOfANewGroup variable 205, 209

fontName attribute 19

Font property 17

footer

adding, to report 81-90

Forecolor property 102

Format option 263

G

getBeanCollection() static method 193

getBeanCollection static method 188

graph

data trends, displaying as 321-330

Group by the following report object drop-down list 67

Group name field 223

GroupPageCount variable 239

groups

nested hierarchy of groups, implementing 73-81

H

header

creating, for report 28-33

heading

inserting, for group of records 59-67

height attribute 14

Hello World report

Blank A4 report template, selecting 11

creating, steps 10-13

Designer tab 11

iReport, running 10

Preview tab 11, 13

simple header, adding 28-33

Static Text component 12

title, creating 15-19

working 13, 15

XML tab 11, 12

HTML tag 123, 133

I

initComponents() method 353, 359, 363

Initial Value Expression property 206, 207, 216

installPostgreSQL.txt file 201, 210, 344

InvoiceID field 55

InvoiceID parameter 175, 180

InvoicePeriod1 Group Header 1 section 300-303

InvoicePeriod Field option 317

InvoiceValue field 73, 299

iReport

<reportElement> tag 18

<staticText> child tag 19

<staticText> tag 18, 19

<text> child tag 19

<title> tag 18, 19

about 8

background color, setting for report 114-122

background images, using 146, 147, 148

bar graph, embedding inside tabular view 331-341

bullet lists, using 123-135

complex multi-dimensional page numbering, implementing 228-239

crosstab, creating 315-321

data displaying as name-value pairs, in multiple column report 270-275

data trends, displaying as graph 321-330

Designer tab 16

design preview 42

downloading 8

downloading, for Linux 9

downloading, for Mac OS X 9

downloading, for Microsoft Windows 8, 9

downloading, for Solaris 9

Empty datasource 15

field, displaying with label 43-51

field expanding vertically, to accomodate large data 135-140

footer, adding 81-89

formatting pattern, applying on data field value 140-144

general information, displaying at end 90-98

heading, inserting for group of records 59-67

height attribute 18

Hello World report, creating 10

HTML tags, using 123-135

installing, for Linux 9

installing, for Mac OS X 9

installing, for Microsoft Windows 8, 9

installing, for Solaris 9

loose coupling 151

multi-column report designing, subreports used 283-296

multi-level summary report, designing 303-314

multi-page report, cover page building for 196-201

multiple components, aligning 19, 20

multiple groups of data, displaying in separate columns 266-270

multiple types of data, displaying in same report 239-249

nested hierarchy of groups, implementing 73-81

null values, handling 43-51

older versions, downloading 10

page numbering resetting, with start of particular record 218-227

pagination of multiple types of data, managing 250-259

- Preview tab 17
- records filtering during report processing, parameters used 68-72
- report, creating from model beans of Java applications 184-193
- report, creating from relational data 152-158
- report, margin setting 34-39
- report body, dividing in multiple columns 261-266
- report creating from XML data, XPath used 162-171
- report filling horizontally, in multiple columns 276-283
- report generating, multiple relational database used 171-184
- report title, creating 15-18
- simple header, adding to report 28-32
- simple one-page TOC, creating for report 201-209
- simple summary report, designing 297-303
- simple TOC, style applying to 210-218
- styles, deploying 100
- styles, reusing 100-113
- summary, displaying at end 90-98
- table of records, creating with labels for each column 51-58
- watermarks, using 146-148
- website, URL 8
- width attribute 18
- XML datasource, connecting to 158-162
- XML tab 17

iReport, views

- designer view 14
- preview view 14
- XML view 14

isStretchWithOverflow="true" attribute 140

J

JAR files

- commons-beanutils-1.8.0.jar 344, 352, 366
- commons-collections-3.2.1.jar 344, 352, 366
- commons-digester-1.7.jar 344, 352, 366
- commons-logging-1.1.jar 344, 352, 367
- groovy-all-1.5.5.jar 344, 352, 367
- iText-2.1.0.jar 344, 352
- jasperreports-3.6.0.jar 344, 352, 366

- location 344
- poi-3.2-FINAL-20081019.jar 367

JasperReport

- background color, setting for report 114-122
- background images, using 146-148
- bar graph, embedding inside tabular view 331-341
- bullet lists, using 123-135
- company logo, inserting in report title 25-28
- complex multi-dimensional page numbering, implementing 228-239
- creating, in Java web application 366-373
- crosstab, creating 315-321
- data displaying as name-value pairs, in multiple column report 270-275
- data trends, displaying as graph 321-330
- embedding, in JSP-based Web application 366-373
- Excel report creating, Java Swing application used 361-365
- field, displaying with label 43-51
- field expanding vertically, to accomodate large data 135-140
- footer, adding 81-90
- formatting pattern, applying on data field value 140-144
- general information, displaying at end 90-98
- hard copy printing, Java Swing application used 357-361
- heading, inserting for group of records 59- 67
- HTML tags, using 123-135
- multi-column report designing, subreports used 283-296
- multi-level summary report, designing 303-314
- multi-page report, cover page building for 196-201
- multiple groups of data, displaying in separate columns 266-270
- multiple types of data, displaying in same report 239-249
- nested hierarchy of groups, implementing 73-81
- null values, handling 43-51
- page numbering resetting, with start of particular record 218-227

- pagination of multiple types of data, managing 250-259
- records filtering during report processing, parameters used 68-72
- report, compiling in Java Swing application 351-357
- report, creating from model beans of Java applications 184-193
- report, creating from relational data 152-158
- report, viewing in Java Swing application 351-357
- report body, dividing in multiple columns 261-266
- report creating from XML data, XPath used 162-171
- report filling horizontally, in multiple columns 276-283
- report generating, multiple relational database used 171-184
- simple one-page TOC, creating for report 201-209
- simple summary report, designing 297-303
- simple TOC, style applying to 210-218
- styles, deploying 100
- styles, reusing 100-113
- summary, displaying at end 90-98
- table of records, creating with labels for each column 51-58
- watermarks, using 146-148
- XML datasource, connecting to 158-162
- JasperReportsPrinter.class file 359**
- JasperReportsPrinter class**
 - initComponents() method 359
 - jButtonActionPerformed() method 359
 - printHardCopy() method 359
- JasperReportsViewer class 353**
- JasperReportsWrapper class 347, 357**
 - connect2DB() method 346
 - fillReport() method 346
 - main() method 346
 - saveReportInPDF() method 346
- JasperReportsXLSExporter.class file 362**
- JasperReports XML. *See* JRXML**
- java.sql.DriverManager class 184**
- java.util.Date() text field 199**
- Java applications**
 - model beans, report creating from 184-193
- Java Development Kit (JDK) 344**
- Java files**
 - ConnectionManager.java 344, 352, 361
 - JasperReportsWrapper.java 344, 352, 361
 - JasperReportsXLSExporter.java 361
- JavaScript**
 - Java wrapper, creating 344, 345
- Java Swing application**
 - report, compiling 352-357
 - report, viewing 352-357
 - used, for creating Excel report 361-365
 - used, for printing hard copy of report 357-361
- Java web application**
 - JasperReport, creating 366-373
- Java wrapper**
 - compile.bat file 345
 - compileJRXMLFile() file 346
 - compileJRXMLFile() method 346
 - Connection object 347
 - creating 344, 345
 - FillManager class 347
 - fillReport() method 347
 - fillReport() method, call adding to 348
 - JasperReportsWrapper.class file 345
 - JasperReportsWrapper.java file 345
 - JasperReportsWrapper.java file, opening 346
 - JasperReportsWrapper class 345-347
 - main() method 347
 - MonthlyCustomerInvoices.jrxml file 346
 - run.bat file 349
 - saveReportInPDF() method 349
 - saveReportInPDF() method, call adding to 349
 - working 350, 351
- jButtonActionPerformed() method 353, 359**
- JRXML**
 - about 12, 13
 - writing, manually 13
- JRXML files**
 - CustHistorySubreport.jrxml 250
 - FeederSummaryReport.jrxml 304, 361
 - MonthlyCustomerReport.jrxml 344, 352
 - MonthlyTransformerBill.jrxml 304

MultipleData.jrxml 250
ProductInvoices.jrxml 250

JRXML tag 14

L

leftMargin attribute 39

Left property 19

Line component 198

Linux

iReport, downloading for 9
iReport, installing for 9
iReport older versions, downloading 10

loose coupling 151

M

Mac OS X

iReport, downloading for 9
iReport, installing for 9
iReport older versions, downloading 10

main() method 346, 347

Markup property 133, 208

Maximize Background option 147

Measure combobox 318

Microsoft Windows

iReport, downloading for 8, 9
iReport, installing for 8, 9
iReport older versions, downloading 10

MinInvoiceValue parameter 73

model beans, Java applications

report, creating from 184-193

MonthlySummaryReport.jrxml subreport 314

multi-column report

designing, subreports used 283-296
multiple groups of data, displaying in separate columns 266-269

multi-level summary report

designing 303-314

multi-page report

cover page, building 196-201

multiple columns

data, displaying as name-value pairs 270-275
report, filling horizontally 276-283
report body, dividing 261-266

multiple groups of data

displaying, in separate columns 266-270

multiple relational databases

used, for generating report 171-184

multiple types of data

displaying, in same report 239-249

N

Name and location window 101

Name property 21, 102, 173

nested hierarchy of groups

implementing 73-81

New group wizard dialog 223

new java.util.Date() expression 85, 92

O

Opaque property 106, 275

overflow attribute 25

Overflow property 139

P

Padding and Borders option 108, 110

Page break component 252, 253

Page break option 200, 209

Page Footer section 43, 220

Page format... window 279

Page format... option 263

Page number component 257

page numbering

complex multi-dimensional page numbering,
implementing 228-239
resetting, with start of particular record 218-
227

Page X of Y component 220

pagination of multiple types of data

managing 250-259

Parameter Class property 70, 173

Parameter prompt dialog 300, 302, 308

parameters

using, to filter records 68-72

Parameters node 70, 173, 309

Pattern editor window 198, 199

Pattern property 143, 145

Pen property 103

**Postgre SQL (org.postgresql.Driver) option
156**

Preview tab 45, 137
preview view 14
printHardCopy() method 359
Print order property 281
printReport() method 360
ProductInvoices.jrxml subreport 257
ProductName text field 78
Properties window 120

Q

query editor 64

R

Read Attributes button 192

records

filtering, parameters used 68-72
group of records, heading inserting 59-67

Rectangle component 116

relational data

report, creating from 152-158

Remove Line When Blank property 290, 294

Remove Report Margins option 306

report

background color, setting 114-122
background images, using 146-148
bar graph, embedding inside tabular view 331-341
bullet lists, using 123-135
company logo, inserting in report title 25
company logo, inserting in title 27, 28
compiling, in Java Swing application 351-357
complex multi-dimensional page numbering, implementing 228-239
creating, from model beans of Java applications 184-193
creating, from relational data 152-158
creating, JasperReports used 8, 13
creating from XML data, XPath used 162-171
crosstab, creating 315-321
data displaying as name-value pairs, in multiple column report 270-275
data trends, displaying as graph 321-330
dynamic title, using 20
Excel report creating, Java Swing application used 361-365

field expanding vertically, to accomodate large data 135-140
filling horizontally, in multiple columns 276-283
footer, adding 81-90
formatting pattern, applying on data field value 140-144
general information, displaying at end 90-98
generating, multiple relational database used 171-184
generating, steps 351
hard copy printing, Java Swing application used 357-361
heading, inserting for group of records 59-67
HTML tags, using 123-135
JasperReport, creating in Java web application 366-373
Java wrapper, creating 344, 345
margins, setting 34-39
multi-column report designing, subreports used 283-296
multi-level summary report, designing 303-314
multi-page report, cover page building for 196-201
multiple groups of data, displaying in separate columns 266-270
multiple types of data, displaying in same report 239-249
nested hierarchy of groups, implementing 73-81
page numbering resetting, with start of particular record 218-227
pagination of multiple types of data, managing 250
records filtering during report processing, parameters used 68-72
report header relative, aligning to report margins 34-39
simple header, adding 28-33
simple one-page TOC, creating 201-209
simple summary report, designing 297-303
simple TOC, style applying to 210-218
static title 20
styles, deploying 100
styles, reusing 100-113

- summary, displaying at end 90-98
- title, creating 15-18
- viewing, in Java Swing application 351-357
- watermarks, using 146-148
- XML datasource, connecting to 158-162
- report body**
 - dividing, in multiple columns 261-266
 - field, displaying with label 43-51
 - null values, handling 43-51
 - table of records, creating with labels for each column 51-58
- report generating, steps**
 - compile manager 351
 - export manager 351
 - fill manager 351
 - report design 351
- Report Inspector window 207**
- Report name field 101**
- Report Query Button 53**
- Report query window 45**
- report title**
 - company logo, inserting 25-28
 - creating 15-18
 - dynamic title 20
 - static title 20
- Reset group property 204**
- Reset page number property 225**
- Reset type property 204, 205, 209, 239, 288**
- result of an aggregation function radio button 299**
- Round Rectangle component 106**
- Round Rectangle transparent 106**
- run.bat file 353**

S

- saveReportInPDF() method 346, 349**
- saveXLSReport() method 363, 364**
- Select an Report Template (JRTX) file dialog box 105**
- Send to Back option 120**
- Show Tick Labels property 338**
- simple one-page TOC**
 - creating, for report 201-209

- simple summary report**
 - designing 297-303
- simple TOC**
 - style, applying 210-218
- SimpleTOCReport.jrxml file 202**
- size parameter 138**
- Size property 16**
- Solaris**
 - iReport, downloading for 9
 - iReport, installing for 9
 - iReport older versions, downloading 10
- Start on a new column property 269**
- Static Text component 107, 198, 273**
- Store the directory name in a parameter option 181, 243, 287, 293**
- Style property 105, 106**
- Style Reference option 105**
- styles**
 - reusing 100-113
- StylishTOCReport.jrxml file 211**
- SUBREPORT_DIR parameter 314**
- Subreport component 290, 310**
- Subreport element 181, 244, 248**
- subreports**
 - used, for designing multi-column report 283-296
- Subreport wizard dialog 246**
- Sum option 299**

T

- Table of contents (TOC) 195**
- table of records**
 - creating, with labels for each column 51-58
- tabular view**
 - bar graph, embedding 331-341
- Template Inspector window 102**
- Text Field component 128, 225, 236**
- Text Field components 220**
- The result of an aggregation function radio button 301**
- TOC_Index variable 216**
- TOC_PageIndex variable 215**
- TOC variable 206**
- Top property 19**
- Total pages component 199**

U

Use an existing report option 178, 285, 292, 310

Use another connection option 180, 242

Use the report XPath expression when filling the report option 162

Use the same connection used to fill the master report option 286

V

Variable Class property 204, 288

Variable Expression property 204

Variable Expression window 204

Variables node 212

viewReport() method 353, 355, 357

W

watermarks

using, in report 146-148

width attribute 18

Width property 181, 198

Windows. *See* Microsoft Windows

X

x attribute 19

XML datasource

connecting to 158-162

XML tab 57

XML view 14

XPath

used, for creating report 162-171

Y

y attribute 19



Thank you for buying JasperReports 3.6 Development Cookbook

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.

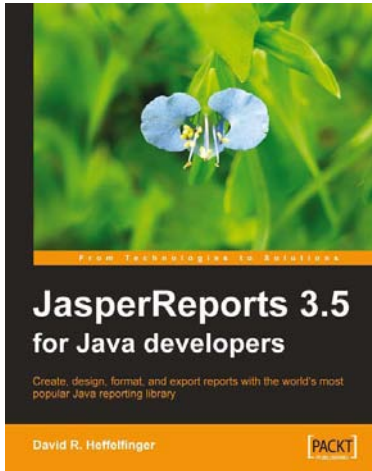
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

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.



JasperReports 3.5 for Java Developers

ISBN: 978-1-847198-08-2

Paperback: 368 pages

Create, Design, Format, and Export Reports with the world's most popular Java reporting library

1. Create better, smarter, and more professional reports using comprehensive and proven methods
2. Group scattered data into meaningful reports, and make the reports appealing by adding charts and graphics
3. Discover techniques to integrate with Hibernate, Spring, JSF, and Struts, and to export to different file formats



JasperReports for Java Developers

ISBN: 978-1-904811-90-9

Paperback: 344 pages

Create, Design, Format and Export Reports with the world's most popular Java reporting library

1. Get started with JasperReports, and develop the skills to get the most from it
2. Create, design, format, and export reports
3. Generate report data from a wide range of datasources
4. Integrate Jasper Reports with Spring, Hibernate, Java Server Faces, or Struts

Please check www.PacktPub.com for information on our titles