Pentaho Analytics for MongoDB Cookbook

MongoDB is an open source, schema-less NoSQL database system. Pentaho, as a famous open source analysis tool, provides high performance and easy scalability for large sets of data.

Whether you are brand new to online learning or a seasoned expert, this book will provide you with the skills you need to create turnkey analytic solutions that deliver insight and drive value for your organization. The book will begin by taking you through Pentaho Data Integration and how it works with MongoDB. This will be followed by an exploration of a MongoDB collection using Pentaho Instant view and creating reports with MongoDB as a datasource using Pentaho Report Designer. The book concludes by highlighting the contributions of the Pentaho Community.

What this book will do for you...

- Extract, load, and transform data from MongoDB collections to other datasources
- Design Pentaho Reports using different types of connections for MongoDB
- Create an OLAP Mondrian schema for MongoDB
- Explore your MongoDB data using Pentaho Analyzer
- Utilize the drag and drop web interface to create dashboards
- Use Kettle Thin JDBC with MongoDB for analysis
- Integrate advanced dashboards with MongoDB using different types of connections
- Publish and run a report on the Pentaho BI server using a web interface

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions to solve problems efficiently
- Clear explanations of what you did
- Solutions that can be applied to solve real-world problems

Pentaho Analytics for MongoDB Cookbook

Over 50 recipes to learn how to use Pentaho Analytics and MongoDB to create powerful analysis and reporting solutions


ISBN 178-32-8739-353277

$ 44.99 US
£ 28.99 UK

Prices do not include local sales tax or VAT where applicable

Joel Latino        Harris Ward
In this package, you will find:

- The authors biography
- A preview chapter from the book, Chapter 1 'PDI and MongoDB'
- A synopsis of the book’s content
- More information on Pentaho Analytics for MongoDB Cookbook
About the Authors

**Joel Latino** was born in Ponte de Lima, Portugal, in 1989. He has been working in the IT industry since 2010, mostly as a software developer and BI developer.

He started his career at a Portuguese company and specialized in strategic planning, consulting, implementation, and maintenance of enterprise software that is fully adapted to its customers' needs.

He earned his graduate degree in informatics engineering from the School of Technology and Management of Viana do Castelo Polytechnic Institute.

In 2014, he moved to Edinburgh, Scotland, to work for Ivy Information Systems, a highly specialized open source BI company in the United Kingdom.

Joel mainly focuses on open source web technology, databases, and business intelligence, and is fascinated by mobile technologies. He is responsible for developing some plugins for Pentaho, such as Android and Apple push notification steps, and lot of other plugins under Ivy Information Systems.

**Harris Ward** has been working in the IT sector since 2004, initially developing websites using LAMP and moving on to business intelligence in 2006. His first role was based in Germany on a product called InfoZoom, where he was introduced to the world of business intelligence. He later discovered open source business intelligence tools and dedicated the last 9 years to not only working on developing solutions, but also working to expand the Pentaho community with the help of other committed members.

Harris has worked as a Pentaho consultant over the past 7 years under Ambient BI. Later, he decided to form Ivy Information Systems Scotland, a company focused on delivering more advanced Pentaho solutions as well as developing a wide range of Pentaho plugins that you can find in the marketplace today.
Preface

With an increasing interest in big data technologies, Pentaho, as a famous open source analysis tool, and MongoDB, the most famous NoSQL database, have gained special focus. The variety of features in Pentaho for MongoDB are end-to-end. This means from data storage in MongoDB clusters to visualization in a dashboard, in a report by e-mail, it's definitely a good change for the processes in enterprises. It's a powerful combination of scalable data storage, data transformation, and analysis.

*Pentaho Analytics for MongoDB Cookbook* explains the features of Pentaho for MongoDB in detail through clear and practical recipes that you can quickly apply to your solutions. Each chapter guides you through the different components of Pentaho: data integration, OLAP, reporting, dashboards, and analysis. This book is a guide to getting started with Pentaho and provides all of the practical information about the connectivity of Pentaho for MongoDB.

**Pentaho Installation**

Pentaho is a commercial open source product, which that means there are two versions available: Pentaho Community Edition (CE) and Pentaho Enterprise Edition (EE). To be able to cover all the recipes of this book, please choose Pentaho EE. You can download the trial version, available at [http://www.pentaho.com](http://www.pentaho.com). In this book, it is mentioned if a specific feature is available in Pentaho CE. You can get that version from [http://community.pentaho.com](http://community.pentaho.com).
Now, we will explain the installation for Pentaho EE:


2. Run the `pentaho-business-analytics-<version>.exe` file for a Windows environment or `pentaho-business-analytics-<version>.bin` for a Linux environment. You will get a **Welcome** window, like what is shown in the following screenshot:

![Welcome Window](image)

3. Click on **Next** and you will get the license agreement, as shown in this screenshot:

![License Agreement](image)
4. After carefully reading the license agreement and accepting, you will be able to choose the setup type in the next screen, as shown in the following screenshot:

![Setup Type Screen]

5. In this case, we'll choose a **Default** installation and click on **Next**. You'll be taken to a screen to choose the folder where Pentaho will be installed, as shown in this screenshot:

![Installation Folder Screen]
6. Feel free to choose your folder path and click on **Next**. You'll get a screen for setting an administrator password, like this:

![Password Setup](image)

7. After typing your password, click on **Next** and you'll be taken to a **Ready To Install** screen, as shown in the following screenshot. Click on **Next** to start the installation and wait a few minutes.

![Ready To Install](image)

8. After some minutes, you will see a screen saying that the installation is complete, and you can test it by accessing `http://localhost:8080/` from your web browser.
What this book covers

Chapter 1, *PDI and MongoDB*, introduces Pentaho Data Integration (PDI), which is an ETL tool for extracting, loading, and transforming data from different data sources.

Chapter 2, *The Thin Kettle JDBC Driver*, teaches you about the JDBC driver for querying Pentaho transformations that connect to various data sources.

Chapter 3, *Pentaho Instaview*, shows you how to create a quick analysis over MongoDB.

Chapter 4, *A MongoDB OLAP Schema*, explains how to create and publish Pentaho OLAP schemas from MongoDB.

Chapter 5, *Pentaho Reporting*, focuses on the creation of printable reports using the Pentaho Report Designer tool. This report can be exported in several formats.


Chapter 7, *Pentaho Dashboards*, focuses on the creation of complex dashboards using the open source suite CTools.

Chapter 8, *Pentaho Community Contributions*, explains the functionality of some contributions from the Pentaho community for MongoDB in Pentaho Data Integration.
In this chapter, we will cover these recipes:

- Learning basic operations with Pentaho Data Integration
- Migrating data from the RDBMS to MongoDB
- Loading data from MongoDB to MySQL
- Migrating data from files to MongoDB
- Exporting MongoDB data using the aggregation framework
- MongoDB Map/Reduce using the User Defined Java Class step and MongoDB Java Driver
- Working with jobs and filtering MongoDB data using parameters and variables

Introduction

Migrating data from an RDBMS to a NoSQL database, such as MongoDB, isn't an easy task, especially when your RBDMS has a lot of tables. It can be a time consuming issue, and in most cases, using a manual process is like developing a bespoke solution.

Pentaho Data Integration (or PDI, also known as Kettle) is an Extract, Transform, and Load (ETL) tool that can be used as a solution for this problem. PDI provides a graphical drag-and-drop development environment called Spoon. Primarily, PDI is used to create data warehouses. However, it can also be used for other scenarios, such as migrating data between two databases, exporting data to files with different formats (flat, CSV, JSON, XML, and so on), loading data into databases from many different types of source data, data cleaning, integrating applications, and so on.

The following recipes will focus on the main operations that you need to know to work with PDI and MongoDB.
Learning basic operations with Pentaho Data Integration

The following recipe is aimed at showing you the basic building blocks that you can use for the rest of the recipes in this chapter. We recommend that you work through this simple recipe before you tackle any of the others. If you want, PDI also contains a large selection of sample transformations for you to open, edit, and test. These can be found in the sample directory of PDI.

Getting ready

Before you can begin this recipe, you will need to make sure that the JAVA_HOME environment variable is set properly. By default, PDI tries to guess the value of the JAVA_HOME environment variable. Note that for this book, we are using Java 1.7. As soon as this is done, you’re ready to launch Spoon, the graphical development environment for PDI. To start Spoon, you can use the appropriate scripts located at the PDI home folder. To start Spoon in Windows, you will have to execute the spoon.bat script in the home folder of PDI. For Linux or Mac, you will have to execute the spoon.sh bash script instead.

How to do it...

First, we need configure Spoon to be able to create transformations and/or jobs. To acclimatize to the tool, perform the following steps:

1. Create a new empty transformation:
   1. Click on the New file button from the toolbar menu and select the Transformation item entry. You can also navigate to File | New | Transformation from the main menu. Ctrl + N also creates a new transformation.

2. Set a name for the transformation:
   1. Open the Transformation settings dialog by pressing Ctrl + T. Alternatively, you can right-click on the right-hand-side working area and select Transformation settings. Or on the menu bar, select the Settings... item entry from the Edit menu.
   2. Select the Transformation tab.
   3. Set Transformation Name to First Test Transformation.
   4. Click on the OK button.
3. **Save the transformation:**
   1. Click on the **Save current file** button from the toolbar. Alternatively, from the menu bar, go to **File | Save.** Or finally, use the quick option by pressing **Ctrl + S.**
   2. Choose the location of your transformation and give it the name `chapter1-first-transformation`.
   3. Click on the **OK** button.

4. **Run a transformation using Spoon.**
   1. You can run the transformation by either of these ways: click on the green play icon on the transformation toolbar and navigate to **Action | Run** on the main menu or simply press **F9.**
   2. You will get an **Execute a transformation** dialog. Here, you can set **parameters, variables, or arguments** if they are required for running the transformation.
   3. Run the transformation by clicking on the **Launch** button.

5. **Run the transformation in preview mode using Spoon.**
   1. In the **Transformation debug** dialog, select the step you want to preview the output data.
   2. After selecting the desired output step, you can preview the transformation by either clicking on the magnify icon on the transformation toolbar, going to **Action | Preview** on the main menu, or simply pressing **F10.**
   3. You will get a **Transformation debug** dialog that you can use to define the number of rows you want to see, breakpoints, and the step that you want to analyze.
   4. You can click on the **Configure** button to define **parameters, variables, or arguments.** Click on the **Quick Launch** button to preview the transformation.

**How it works...**

In this recipe, we just introduced the Spoon tool, touching on the main basic points for you to manage ETL transformations. We started by creating a transformation. We gave a name to the transformation, **First Test Transformation** in this case. Then, we saved the transformation in the filesystem with the name `chapter1-first-transformation`.

Finally, we ran the transformation normally and in debug mode. Understanding how to run a transformation in debug mode is useful for future ETL developments as it helps you understand what is happening inside of the transformation.
There's more...

In the PDI home folder, you will find a large selection of sample transformations and jobs that you can open, edit, and run to better understand the functionality of the diverse steps available in PDI.

Migrating data from the RDBMS to MongoDB

In this recipe, you will transfer data from a sample RDBMS to a MongoDB database. The sample data is called SteelWheels and is available in the Pentaho BA server, running on the Hypersonic Database Server.

Getting ready

Start the Pentaho BA Server by executing the appropriate scripts located in the BA Server's home folder. It is start-pentaho.sh for Unix/Linux operating systems, and for the Windows operating system, it is start-pentaho.bat. Also in Windows, you can go to the Start menu and choose Pentaho Enterprise Edition, then Server Management, and finally Start BA Server.

Start Pentaho Data Integration by executing the right scripts in the PDI home folder. It is spoon.sh for Unix/Linux operating systems and spoon.bat for the Windows operating system. Besides this, in Windows, you can go to the Start menu and choose Pentaho Enterprise Edition, then Design Tools, and finally Data Integration.

Start MongoDB. If you don't have the server running as a service, you need execute the mongod -dbpath=<data folder> command in the bin folder of MongoDB.

To make sure you have the Pentaho BA Server started, you can access the default URL, which is http://localhost:8080/pentaho/. When you launch Spoon, you should see a welcome screen like the one pictured here:
How to do it...

After you have made that sure you are ready to start the recipe, perform the following steps:

1. Create a new empty transformation.
   1. As was explained in the first recipe of this chapter, set the name of this transformation to **Migrate data from RDBMS to MongoDB**.
   2. Save the transformation with the name **chapter1-rdbms-to-mongodb**.
2. Select a customer’s data from the **SteelWheels** database using **Table Input** step.
   1. Select the **Design** tab in the left-hand-side view.
   2. From the **Input** category folder, find the **Table Input** step and drag and drop it into the working area in the right-hand-side view.
   3. Double-click on the **Table Input** step to open the configuration dialog.
   4. Set the **Step Name** property to **Select Customers**.
   5. Before we can get any data from the **SteelWheels** Hypersonic database, we will have to create a JDBC connection to it.

To do this, click on the **New** button next to the **Database Connection** pulldown. This will open the **Database Connection** dialog.
Set **Connection Name** to **SteelWheels**. Next, select the **Connection Type** as **Hypersonic**. Set **Host Name** to **localhost**, **Database Name** to **SampleData**, **Port** to **9001**, **Username** to **pentaho_user**, and finally **Password** to **password**.

Your setup should look similar to the following screenshot:

6. You can test the connection by clicking on the **Test** button at the bottom of the dialog. You should get a message similar to **Connection Successful**. If not, then you must double-check your connection details.

7. Click on **OK** to return to the **Table Input** step.

8. Now that we have a valid connection set, we are able to get a list of customers from the **SteelWheels** database. Copy and paste the following SQL into the query text area:

   ```sql
   SELECT * FROM CUSTOMERS
   ```

9. Click on the **Preview** button and you will see a table of customer details.
10. Your **Table Input** step configuration should look similar to what is shown in the following screenshot:

![Table Input Configuration](image)

11. Click on **OK** to exit the **Table Input configuration** dialog.

3. Now, let's configure the output of the customer's data in the MongoDB database.
   1. Under the **Design** tab, from the **Big Data** category folder, find the **MongoDB Output** step and drag and drop it into the working area in the right-hand-side view.
   2. As we want data to flow from the **Table Input** step to the **MongoDB Output** step, we are going to create a **Hop** between the steps. To do this, simply hover over the **Table Input** step and a popup will appear, with some options below the step. Click on **Right Arrow** and then on the **MongoDB Output** step. This will create a **Hop** between the two steps.

![Hop between steps](image)

3. It's time to configure the MongoDB Output step. Double-click on it.
4. Set **Step Name** to **Customers Output**.
5. As we're running a default MongoDB instance, we only have to set some simple properties in this step. Set **Hostname** to **localhost** and **Port** to **27017**.
6. Select the **Output options** tab. In this tab, we can define how the data will be inserted into MongoDB.

7. Set the **Database** property to **SteelWheels**. Don't worry if this database doesn't exist in MongoDB, as it will be created automatically.

8. Set the **Collection** property to **Customers**. Again, don't worry if this collection doesn't exist in MongoDB, as it will be created automatically.

9. Leave the **Batch insert size** property at 100. For performance and/or production purposes, you can increase it if necessary. If you don't provide any value to this field, the default value will be 100.

10. We are going to truncate the collection each time before we load data. In this way, if we rerun the transformation many times, we won't get duplicate records. Your **Output options** page should look like what is shown in this screenshot:

![MongoDB Output](image)

11. Now, let's define the MongoDB documents structure. Select the **MongoDB document fields** tab.

12. Click on the **Get fields** button, and the fields list will be populated with the SteelWheels database fields in the ETL stream.
13. By default, the column names in the SteelWheels database are in uppercase. In MongoDB, these field names should be in camel case. You can manually edit the names of the MongoDB document paths in this section also. Make sure that the Use Field Name option is set to No for each field, like this:

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Mongo document path</th>
<th>Use field name</th>
<th>JSON</th>
<th>Match/field for update</th>
<th>Modifier operation</th>
<th>Modifier policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CUSTOMERNUMBER</td>
<td>customerNumber</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>2</td>
<td>CUSTOMERNAME</td>
<td>customerName</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3</td>
<td>CONTACT/lastname</td>
<td>contact.lastName</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4</td>
<td>CONTACT/firstname</td>
<td>contact.firstName</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5</td>
<td>PHONE</td>
<td>contact.phone</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>6</td>
<td>ADDRESS1/line1</td>
<td>address.address1/line1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>7</td>
<td>ADDRESS1/line2</td>
<td>address.address1/line2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>8</td>
<td>CITY</td>
<td>address.city</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>9</td>
<td>STATE</td>
<td>address.state</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>10</td>
<td>POSTALCODE</td>
<td>address.postalCode</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>11</td>
<td>COUNTRY</td>
<td>address.country</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>12</td>
<td>SALESMERPEO/employeeID</td>
<td>address.salemsmerpeo/employeeID</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>13</td>
<td>CREDITLIMIT</td>
<td>creditLimit</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

14. By clicking on Preview document structure, you will see an example of what the document will look like when it is inserted into the MongoDB Customers collection.

15. Click on the OK button to finish the MongoDB Output configuration.

4. The transformation design is complete. You can run it for testing purposes using the Run button, as illustrated here:
**How it works...**

As you can see, this is a basic transformation that loads data from the RDBMS database and inserts it into a MongoDB collection. This is a very simple example of loading data from one point to another. Not all transformations are like this. That is why PDI comes with various steps that allow you to manipulate data along the way.

In this case, we truncate the collection each time the transformation is run. However, it is also possible to use other combinations, such as Insert&Update or just Insert or Update individually.

**There's more...**

Now that we have designed a transformation, let's look at a simple way of reusing the MongoDB connection for future transformations.

**How to reuse the properties of a MongoDB connection**

If you have to create MongoDB connections manually for each transformation, you are likely to make mistakes and typos. A good way to avoid this is to store the MongoDB connection details in a separate `.properties` file on your filesystem. There is a file called `kettle.properties` that is located in a hidden directory called `.kettle` in your home directory. For example, in Linux, the location will be `/home/latino/.kettle`. In Windows, it will be `C:\Users\latino\.kettle`. Navigate to and open this `.properties` file in your favorite text editor. Then, copy and paste the following lines:

```plaintext
MONGODB_STEELWHEELS_HOSTNAME=localhost
MONGODB_STEELWHEELS_PORT=27017
MONGODB_STEELWHEELS_USERNAME=
MONGODB_STEELWHEELS_PASSWORD=
```

Save the `.properties` file and restart Spoon.

Now, where can we use these properties?

You will notice that when you are setting properties in certain PDI steps, you can see the following icon:
This icon denotes that we can use a variable or parameter in place of a static value. Variables are defined using the following structure: $\text{MY_VARIABLE}$. You will notice that the variables are encapsulated in $\{\}$. If you are not sure what the name of your variable is, you can also press Ctrl and the Spacebar; this will open a drop-down list of the available variables. You will see the MongoDB variables that you defined in the .properties file earlier in this list. With this in mind, we can now replace the connection details in our steps with variables as shown in this screenshot:

You can find out more about the MongoDB Output step on this documentation website: http://wiki.pentaho.com/display/EAI/MongoDB+Output

### Loading data from MongoDB to MySQL

In this recipe, we will guide you through extracting data from MongoDB and inserting it into a MySQL database. You will create a simple transformation as you did in the last recipe, but in reverse. You don't have to use MySQL as your database. If you want, you can use any other database. You just need to make sure that you can connect to Pentaho Data Integration via JDBC. However, in this book, we will use MySQL as an example.

### Getting ready

Make sure you have created a MySQL database server or some other database type server with a database called SteelWheels. Also make sure that your MongoDB instance is running and launch Spoon.

### How to do it...

After you have made sure that you have the databases set up, perform the following steps:

1. Create a new empty transformation.
2. Set the name for this transformation to **Loading data from MongoDB to MySQL**.
3. Save the transformation with the name `chapter1-mongodb-to-mysql`.
2. Select Customers from MongoDB using the MongoDB Input step.
   1. Select the Design tab in the left-hand-side view.
   2. From the Big Data category folder, find the MongoDB Input step and drag and drop it into the working area in the right-hand-side view.
   3. Double-click on the MongoDB Input step to open the configuration dialog.
   4. Set the Step Name property to Select Customers.
   5. Select the Input options tab. Click on Get DBs and select SteelWheels from the Database select box.
   6. After selecting the database, you can click on the Get Collections button and then select Customers Collection from the select box.
   7. As we're just running one MongoDB instance, we'll keep Read preference as primary and will not configure any Tag set specification.
   8. Click on the Query tab. In this section, we'll define the where filter data condition and the fields that we want to extract.
   9. As we just want the customers from USA, we'll write the following query in the Query expression (JSON) field: \{"address.country": "USA"\}.

   In this recipe, we are not going to cover the MongoDB aggregation framework, so you can ignore those options for now.

10. Click on the Fields tab. In this tab, we'll define the output fields that we want. By default, the Output single JSON field comes checked. This means that each document is extracted in the JSON format with the field name defined in the Name of JSON output field. As we want to define the fields, we remove the selection of the Output single JSON field.

11. Click on the Get fields button and you will get all the fields available from MongoDB. Remove the _id field because it isn't necessary. For deletion, you can select the row of the _id field and press the Delete key from your keyboard, or right-click on the row and select the Delete selected lines option.

12. Click on OK to finish the MongoDB input configuration.

3. Let's configure the output of the MongoDB Customers data in the MySQL database.
   1. On the Design tab, from the Output category folder, find the Table Output step and drag and drop it into the working area in the right-hand-side view.
   2. Connect the MongoDB Input step to the Table output step by creating a hop between them.
3. Double-click on the step to open the Table Output configuration dialog.

4. Set Step Name to Customers Output.

5. Click on the New button next to the Database Connection pulldown. This will open the Database Connection dialog.

Set Connection Name to SteelWheels. Select the Connection Type as MySQL. Set Host Name to localhost, Database Name to SteelWheels, and Port to 3306. Then, set Username and Password to whatever you had set them as. Your setup should look similar to the following screenshot:

6. Test this, and if all is well, click on OK to return to the Table Output step.

4. Insert this data into a MySQL table using the Table Output step:

1. Set the Target table field to Customers. This is the name of the MySQL table to insert data into.

2. As we haven't created a customer's table in the MySQL database, we can use a PDI function that will try to generate the required SQL to create the table and structure. Simply click on the SQL button and it will open the Execute SQL dialog. Here, you will see the SQL that PDI will execute to create the customers table. Click on Execute to send this SQL to MySQL and create the table. Then, click on OK.
3. Click on OK again to exit the Table Output configuration dialog. The transformation is complete. You can now run it to load data from MongoDB to MySQL.

How it works...

In this transformation, we are simply selecting a collection from the MongoDB Input step where the country field is USA. Next, we map this collection to the fields in the PDI stream. Lastly, we insert this data into a MySQL table using the Table Output step. In the Fields tab, we use JSONPath to select the correct data from the MongoDB collection (http://goessner.net/articles/JsonPath/). JSONPath is like XPath for JSON documents.

Migrating data from files to MongoDB

In this recipe, we will guide you through creating a transformation that loads data from different files in your filesystem, and then load them into a MongoDB Collection. We are going to load data from files called orders.csv, customers.xls, and products.xml. Each of these files contains a key that we can use to join data in PDI before we send it to the MongoDB Output step.

Getting ready

Start Spoon and take a look at the content of the orders.csv, customers.xls, and products.xml files. This will help you understand what the data looks like before you start loading it into MongoDB.

How to do it...

You will need the orders.csv, customers.xls, and products.xml files. These files will be available at the Packt Publishing website, just in case you don't have them. Make sure that MongoDB is up and running, and then you will be able to perform to the following steps:

1. Create a new empty transformation.
   1. Set the transformation name to Migrate data from files to MongoDB.
   2. Save the transformation with the name chapter1-files-to-mongodb.

2. Select data from the orders.csv file using the CSV file input step.
   1. Select the Design tab in the left-hand-side view.
   2. From the Input category folder, find the CSV file input step and drag and drop it into the working area in the right-hand-side view.
   3. Double-click on the step to open the CSV Input configuration dialog.
4. Set **Step Name** to **Select Orders**.

5. In the **Filename** field, click on the **Browse** button, navigate to the location of the `.csv` file, and select the `order.csv` file.

6. Set the **Delimiter** field to a semicolon (`;`).

7. Now, let's define our output fields by clicking on the **Get Fields** button. A **Sample size** dialog will appear; it is used to analyze the format data in the CSV file. Click on **OK**. Then, click on **Close** in **Scan results**.

8. Click on **OK** to finish the configuration of the **CSV file input**.

3. Select data from the **customers.xls** file using the **Microsoft Excel Input** step.

1. Select the **Design** tab in the left-hand-side view.

2. From the **Input** category folder, find the **Microsoft Excel Input** step and drag and drop it into the working area in the right-hand-side view.

3. Double-click on the step to open the **Microsoft Excel Input** dialog.

4. Set **Step Name** to **Select Customers**.

5. On the **Files** tab, in the **File or directory** field, click on the **Browse** button and choose the location of the **customers.xls** file in your filesystem. After that, click on the **Add** button to add the file to the list of files to be processed.

6. Select the **Sheets** tab. Then, click on the **Get sheetname(s)...** button. You'll be shown an **Enter list** dialog. Select **Sheet1** and click on the > button to add a sheet to the **Your selection** list. Finally, click on **OK**.

7. Select the **Fields** tab. Then, click on the **Get field from header row...** button. This will generate a list of existing fields in the spreadsheet. You will have to make a small change; change the **Type** field for **Customer Number** from **Number** to **Integer**. You can preview the file data by clicking on the **Preview rows** button.

8. Click on **OK** to finish the configuration of the **Select Customers step**.

4. Select data from the **products.xml** file using the **Get data from XML** step.

1. Select the **Design** tab in the left-hand-side view.

2. From the **Input** category folder, find the **Get data from XML** step and drag and drop it into the working area in the right-hand-side view.

3. Double-click on the step to open the **Get data from XML** dialog.

4. Set **Step Name** to **Select Products**.

5. On the **File** tab, in the **File or directory** field, click on the **Browse** button and choose the location of the **products.xml** file in your filesystem. After that, click on the **Add** button to add the file to the list of files to be processed.

6. Select the **Content** tab. Click on the **Get XPath nodes** button and select the `/products/product` option from the list of the **Available Paths** dialog.
7. Next, select the **Fields** tab. Click on the **Get fields** button and you will get a list of available fields in the XML file. Change the types of the last three fields (*stockquantity*, *buyprice*, and *MSRP*) from **Number** to **Integer**. Set the **Trim Type** to **Both** for all fields.

5. Now, let's join the data from the three different files.
   1. Select the **Design** tab in the left-hand-side view.
   2. From the **Lookup** category folder, find the **Stream lookup** step. Drag and drop it onto the working area in the right-hand-side view. Double-click on **Stream lookup** and change the **Step name** field to **Lookup Customers**.
   3. We are going to need two lookup steps for this transformation. Drag and drop another Stream Lookup step onto the design view, and set **Step Name** to **Lookup Products**.
   4. Create a hop between the **Select Orders** step and the **Lookup Customers** step.
   5. Then, create a hop from the **Select Customers** step to the **Lookup Customers** step.
   6. Next, create a hop from the **Lookup Customers** step to the **Lookup Products** step.
   7. Finally, create a hop from **Select Products** to the **Lookup Products** step.

6. Let's configure the **Lookup Customers** step. Double-click on the **Lookup Customers** step and set the **Lookup step** field to the **Select Customers** option.
   1. In the **Keys** section, set the **Field** and **Lookup Field** options to **Customer Number**.
   2. Click on the **Get lookup fields** button. This will populate the step with all the available fields from the lookup source. Remove **Customer Number** from the field from the list.
   3. Click on **OK** to finish.

7. Let's configure the **Lookup Products** step. The process is similar to that of the **Lookup Customers** step but with different values. Double-click on the **Lookup Products** step and set the **Lookup step** field to the **Select Products** option.
   1. In the **Keys** section, set **Field** to **Product Code** and the **LookupField** option to **Code**.
   2. Click on the **Get lookup fields** button. This will populate the step with all the available fields from the lookup source. Remove **Code** from the field in the list.
   3. Click on **OK** to finish.
8. Now that we have the data joined correctly, we can write the data stream to a MongoDB collection.

1. On the **Design** tab, from the **Big Data** category folder, find the **MongoDB Output** step and drag and drop it into the working area in the right-hand-side view.

2. Create a hop between the **Lookup Products** step and the **MongoDB Output** step.

3. Double-click on the **MongoDB Output** step and change the **Step name** field to **Orders Output**.

4. Select the **Output options** tab. Click on the **Get DBs** buttons and select the **SteelWheels** option for the **Database** field. Set the **Collection** field to **Orders**. Check the **Truncate collection** option.

5. Select the **Mongo document fields** tab. Click on the **Get fields** button and you will get a list of fields from the previous step.

6. Configure the Mongo document output as seen in the following screenshot:

```plaintext
<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Mongo document path</th>
<th>UseField name</th>
<th>JSON</th>
<th>MatchField for update</th>
<th>Modifier operation</th>
<th>Modifier policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Order Number</td>
<td>orderNumber</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>QuantityOrdered</td>
<td>quantityOrdered</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Price Each</td>
<td>priceEach</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Order Line Number</td>
<td>orderLineNumber</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Total Price</td>
<td>totalPrice</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Order Date</td>
<td>orderDate</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Required Date</td>
<td>requiredDate</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Shipped Date</td>
<td>shippedDate</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Status</td>
<td>status</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Customer Number</td>
<td>customer.customerNumber</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Line</td>
<td>customer.line</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Customer Name</td>
<td>customer.name</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Contact First Name</td>
<td>customer.contact.firstName</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Contact Last Name</td>
<td>customer.contact.lastName</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Phone</td>
<td>customer.contact.phone</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Address Line 1</td>
<td>customer.address.line1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Address Line 2</td>
<td>customer.address.line2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>City</td>
<td>customer.address.city</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>State</td>
<td>customer.address.state</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Postal Code</td>
<td>customer.address.postalCode</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Country</td>
<td>customer.address.country</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Salesforce Employee Number</td>
<td>customer.salesforce.employeeNumber</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Credit Limit</td>
<td>customer.creditLimit</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Product Code</td>
<td>product.code</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>name</td>
<td>product.name</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>site</td>
<td>product.site</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>scale</td>
<td>product.scale</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>vendor</td>
<td>product.vendor</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>description</td>
<td>product.description</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>stockQuantity</td>
<td>product.stockQuantity</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>buyPrice</td>
<td>product.buyPrice</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>retSale</td>
<td>product.retSale</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>
```

7. Click on **OK**.
9. You can run the transformation and check out MongoDB for the new data. Your transformation should look like the one in this screenshot:

![Transformation Diagram]

**How it works...**

In this transformation, we initially get data from the Orders CSV. This first step populates the primary data stream in PDI. Our other XLS and XML steps also collect data. We then connect these two streams of data to the first stream using the Lookup steps and the correct keys. When we finally have all of the data in the single stream, we can load it into the MongoDB collection.

You can learn more about the **Stream lookup** step online at:

http://wiki.pentaho.com/display/EAI/Stream+Lookup

**Exporting MongoDB data using the aggregation framework**

In this recipe, we will explore the use of the MongoDB aggregation framework in the MongoDB Input Step. We will create a simple example to get data from a collection and show you how you can take advantage of the MongoDB aggregation framework to prepare data for the PDI stream.

**Getting ready**

To get ready for this recipe, you will need to start your ETL development environment **Spoon**, and make sure that you have the MongoDB server running with the data from the previous recipe.
Chapter 1

How to do it...

The following steps introduce the use of the MongoDB aggregation framework:

1. Create a new empty transformation.
   1. Set the transformation to **PDI using MongoDB Aggregation Framework**.
   2. Set the name for this transformation to `chapter1-using-mongodb-aggregation-framework`.

2. Select data from the **Orders** collection using the **MongoDB Input** step.
   1. Select the **Design** tab in the left-hand-side view.
   2. From the **Big Data** category folder, find the **MongoDB Input** step and drag and drop it into the working area in the right-hand-side view.
   3. Double-click on the step to open the **MongoDB Input** dialog.
   4. Set the step name to **Select 'Baane Mini Imports' Orders**.
   5. Select the **Input options** tab. Click on the **Get DBs** button and select the `SteelWheels` option for the **Database** field. Next, click on **Get collections** and select the **Orders** option for the **Collection** field.
   6. Select the **Query** tab and then check the **Query is aggregation pipeline** option. In the text area, write the following aggregation query:
      
      ```json
      [  
        { $match: { "customer.name" : "Baane Mini Imports" } },  
        { $group: { "_id" : { "orderNumber": "$orderNumber",  
                        "orderDate" : "$orderDate"},  
                      "totalSpend": { $sum:  
                                      "$totalPrice"} } }  
      ]
      ```

   7. Uncheck the **Output single JSON field** option.
   8. Select the **Fields** tab. Click on the **Get Fields** button and you will get a list of fields returned by the query. You can preview your data by clicking on the **Preview** button.
   9. Click on the **OK** button to finish the configuration of this step.

3. We want to add a **Dummy** step to the stream. This step does nothing, but it will allow us to select a step to preview our data. Add the **Dummy** step from the **Flow** category to the workspace and name it **OUTPUT**.

4. Create a hop between the **Select 'Baane Mini Imports' Orders** step and the **OUTPUT** step.

5. Select the **OUTPUT** dummy step and preview the data.
How it works...

The MongoDB aggregation framework allows you to define a sequence of operations or stages that is executed in pipeline much like the Unix command-line pipeline. You can manipulate your collection data using operations such as filtering, grouping, and sorting before the data even enters the PDI stream.

In this case, we are using the MongoDB Input step to execute an aggregation framework query. Technically, this does the same as `db.collection.aggregate()`. The query that we execute is broken down into two parts. For the first part, we filter the data based on a customer name. In this case, it is Baane Mini Imports. For the second part, we group the data by order number and order date and sum the total price.

See also

In the next recipe, we will talk about other ways in which you can aggregate data using MongoDB Map/Reduce.

MongoDB Map/Reduce using the User Defined Java Class step and MongoDB Java Driver

In this recipe, we will use the MongoDB Map/Reduce on PDI. Unfortunately, PDI doesn't provide a step for this MongoDB feature. However, PDI does provide a step called User Defined Java Class (UDJC) that will allow you to write Java code to manipulate your data.

We are going to get the total price for all orders for a single client, which we will pass to the transformation as a parameter. We will also get a total for all other clients in the collection. In total, we should get two rows back.

Getting ready

To get ready for this recipe, you need to download the MongoDB driver. In this case, we are using the `mongo-java-driver-2.11.1` version. You can use the last version, but the code in this recipe may be a bit out of date. The driver should live in the `lib` folder of PDI. Then, you just need start your ETL development environment Spoon and make sure you have the MongoDB server started with the data from the last recipe inserted.
How to do it...

In this recipe, we'll program Java code and utilize the MongoDB Java driver to connect to the MongoDB database. So, make sure you have the driver in the lib folder of PDI and then perform the following steps:

1. Create a new empty transformation.
   1. Set the transformation name to **MongoDB Map/Reduce**.
   2. On the **Transformation properties and Parameters** tab, create a new parameter with the name as **CUSTOMER_NAME**.
   3. Save the transformation with the name **chapter1-mongodb-map-reduce**.

2. From the **Job** category folder, find the **Get Variables** step and drag and drop it into the working area in the right-side view.
   1. Double-click on the **Get Variables** step to open the configuration dialog.
   2. Set the **Step name** property to **Get Customer Name**.
   3. Add a row with the name as **customerName**, the variable as **${CUSTOMER_NAME}**, and **Type set** to **String**.

3. From the **Scripting** category folder, find the **User Defined Java Class** step and drag and drop it into the working area in the right-hand-side view.

4. Create a hop between the **Get Customer Name** step and the **User Defined Java Class** step.
   1. Double-click on the **User Defined Java Class** step to open the configuration dialog.
   2. In the **Step name** field, give a suggested name of **MapReduce**.
   3. In **Class code**, let's define our Java code that is sent to MongoDB by a command using the MapReduce functions and then we will get the result:

```java
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodbDBObject;
import com.mongodb.MapReduceCommand;
import com.mongodb.MapReduceOutput;
import com.mongodb.Mongo;

private FieldHelper customerNameIn = null;

public boolean processRow(StepMetaInterface smi,
StepDataInterface sdi) throws KettleException
{
  Object[] r = getRow();
```
if (r == null) {
    setOutputDone();
    return false;
}
if (first) {
    first = false;
    customerNameIn = get(Fields.In, "customerName");
}

try {
    final Mongo mongo = new Mongo("localhost", 27017);
    final DB db = mongo.getDB("SteelWheels");
    final DBCollection ordersCol = db.getCollection("Orders");
    final String map = "function() { " +
        "var category; " +
        "if ( this.customer.name == "+
            "\"" + customerNameIn.getString(r) + "\"\" ) " +
            "category = " + customerNameIn.
                getString(r) + "; " +
            "else " + "category = 'Others'; " +
            "emit(category, {totalPrice: this.
                totalPrice,
                count: 1});\"\"}" +
                 "var n = { count: 0, totalPrice: 0}; " +
            "for ( var i = 0; i < values.length; i++ ) {" +
            "n.count += values[i].count; " +
            "n.totalPrice += values[i].totalPrice; " +
            "} + "return n;\"}" +
                 final MapReduceCommand cmd = new MapReduceCommand(
                    ordersCol, map, reduce, null,
                    MapReduceCommand.OutputType.INLINE, null);
    final MapReduceOutput out = ordersCol.mapReduce(cmd);
    get(Fields.Out, "mapReduceJSON").setValue(r, out.toString());
} catch (Exception e) {
    e.printStackTrace();
    get(Fields.Out, "mapReduceJSON").setValue(r, "");
}
r = createOutputRow(r, data.outputRowMeta.size());
putRow(data.outputRowMeta, r);
return true;"
4. On the **Fields** tab, set **Fieldname** to **mapReduceJSON** and the **Type** property to **String**. This will be the field output from the **MapReduce** command.

5. Click on **OK** to finish the configuration.

5. From the **Input** category folder, find the **Json Input** step and drag and drop it into the working area in the right-hand-side view.

6. Create a hop between the **MapReduce** step and the **Json Input** step.
   1. Double-click on the **JSON Input** step to open the configuration dialog.
   2. Set the **Step Name** property to **Convert JSON**.
   3. On the **File** tab, check the **Source is defined in a field?** option. Next, select the **mapReduceJSON** option in the select box of **Get source from field**.
   4. On the **Fields** tab, we will map the JSON to Fields in the PDI stream. The definition should be like what is shown in this screenshot:

   ![Json Input Step Configuration]

   5. Click on **OK** to finish the configuration.

7. Now, let's define the fields that we want to see as the output of the transformation. From the **Transform** category folder, find the **Select values** step and drag and drop it into the working area in the right side view.

8. Create a hop between the **Convert JSON** step and the **Select values** step.
   1. Double-click on the **Select Values** step to open the configuration dialog.
   2. Set the **Step Name** property to **OUTPUT**.
3. On the Select & Alter tab, click on the Get fields to select button. This will populate the table with all the available fields in the stream. Remove the mapReduceJSON field; it isn't necessary anymore, since we have converted it into individual fields in the PDI stream.

4. Click on OK to finish the configuration.

9. When you run the transformation, be sure to set the CUSTOMER_NAME parameter in the Run dialog. This will be used by the Get Customer Name step and to filter the map function.

![Diagram](image.png)

**How it works...**

In this example, we executed a transformation that takes CUSTOMER_NAME as a parameter. This value is then sent to User Defined Java Class and used in the Java code within. The code in User Defined Java Class is a simple Map and Reduce JavaScript function that we are sending to the MongoDB server.

The output of this step is a single JSON row that needs to be parsed into fields in the PDI Stream. To do this, we used the JSON input step and mapped the JSON string to individual stream fields.

If you want to know more about User Defined Java Class, you can find out more in the documentation at [http://wiki.pentaho.com/display/EAI/User+Defined+Java+Class](http://wiki.pentaho.com/display/EAI/User+Defined+Java+Class).

**There's more...**

When we talk about map and reduce functions, it is almost mandatory to talk about Hadoop, an open source software framework for storage and processing of datasets that uses a MapReduce engine.

PDI provides integration with Hadoop using PDI job steps and transformation steps. You can find more documentation about this on the Pentaho website. Personally, I recommend these two tutorials:

- [http://wiki.pentaho.com/display/BAD/Using+Pentaho+MapReduce+to+Parse+Weblog+Data](http://wiki.pentaho.com/display/BAD/Using+Pentaho+MapReduce+to+Parse+Weblog+Data)
- [http://wiki.pentaho.com/display/BAD/Using+Pentaho+MapReduce+to+Generate+an+Aggregate+Dataset](http://wiki.pentaho.com/display/BAD/Using+Pentaho+MapReduce+to+Generate+an+Aggregate+Dataset)
Working with jobs and filtering MongoDB data using parameters and variables

In this recipe, we guide you through creating two PDI jobs. One uses variables and the other uses parameters. In a PDI process, jobs orchestrate other jobs and transformations in a coordinated way to realize the main business process. These jobs use the transformation created in the last recipe but with some changes, as described in this recipe.

So, in this recipe, we are going to create two different jobs, which will be used to send data to a subtransformation. The subtransformation that we will use will be a copy of the transformation in the previous recipe.

Getting ready

To get ready for this recipe, you need to start your ETL development environment Spoon, and make sure you have the MongoDB server started with the data inserted in the last recipes.

How to do it...

Let's start using jobs and variables. We can orchestrate the ETL to run in different ways. In this simple case, we are just using the customer name. Perform the following steps:

1. Let's copy and paste the transformation created in the previous recipe and save it as chapter1-mongodb-map-reduce-writelog.ktr.
2. Open that transformation using Spoon, and from the Utility category folder, find the Write to log step. Drag and drop it into the working area in the right-side view.
   1. Create a hop between the OUTPUT step and the Write to log step.
   2. Double-click on the Write to Log step to open the configuration dialog.
   3. Set Step Name to MapReduce.
   4. Click on the Get Fields button.
   5. Click on OK to finish the configuration.
3. Let's create a new empty job.
   1. Click on the New file button from the toolbar menu and select the Job item entry. Alternatively from menu bar, go to File | New | Job.
   2. Open the Job properties dialog by pressing Ctrl + J or by right-clicking on the right-hand-side working area and selecting Job settings.
   3. Select the Job tab. Set Job Name to Job Parameters.
4. Select the **Parameters** tab and add a **Parameter** entry with the name as **CUSTOMER_NAME**. Click on **OK**.

5. Save the Job with the name **job-parameters**.

4. From the **General** category folder, find the **START**, **Transformation**, and **Success** steps and drag and drop them into the working area in the right-side view.
   1. Create a hop between the **START** step and the **Transformation** step.
   2. Then, create a hop from the **Transformation** step to the **Success** step.
   3. Double-click on the **Transformation** step to open the configuration dialog
   4. Change the **Name of job entry** property to **MapReduce Transf**.
   5. Click on the transformation button of the **Transformation filename** field and select the transformation file that you copied before in your filesystem. Also select the **chapter1-mongodb-map-reduce-writelog.ktr** file.
   6. Select the **Parameters** tab. By default, the **Pass all parameters values down to the sub-transformation** option is checked, which means our job parameter will be passed to the transformation.
   7. Click on **OK** to finish.
   8. Run the job and analyze the results and check the logs on the **Logging** tab.

Now let's do a quick and simple example using variables:

1. Copy and paste the **chapter1-mongodb-map-reduce-writelog** transformation. Save it as **chapter1-mongodb-map-reduce-writelog-without-parameter**.

2. Open the transformation with Spoon and remove the parameter from **Transformation properties**.

3. Copy and paste the last job. Save it as **job-variables**.
   1. Open the job with Spoon.
   2. In **Job properties**, change the job name to **Job Variables**. From the **Parameters** tab, remove the **CUSTOMER_NAME** parameter. Select the parameter, right-click on it and select **Delete selected lines**, or just press delete on your keyboard.
   3. Click on **OK** to finish.

4. From the **General** category folder, find the **Set variables** step and drag and drop it into the working area in the right-side view.
   1. Remove the hop from between the **START** step and **MapReduce Transf** step.
   2. Create a hop between the **START** step and the **Set variables** step.
3. Then, create a hop between Set Variables and the MapReduce Transf step.
4. Double-click on the Set Variables step to open the configuration dialog.
5. Set the Step name property to Set CUSTOMER_NAME.
6. On Variables, create a new variable with the CUSTOMER_NAME name. Set the value to an existing client in the database and the Scope type to Valid in the root job.
7. Click on OK to finish the configuration.

5. On the MapReduce Transf transformation step, change the file location for the transformation file to the transformation without the parameter.
6. Run the job and analyze the results, checking the logs in the Logging tab.

**How it works…**

Most ETL solutions created in Pentaho Data Integration will be sets of jobs and transformations.

Transformations are workflows with an orchestration of actions that manipulate data using essentially input, transformation, and output steps.

Jobs are workflows with an orchestration of tasks that can be order execution failure or success.

Variables and parameters are extremely useful functions that we can use to create dynamic jobs and transformations.
Where to buy this book

You can buy Pentaho Analytics for MongoDB Cookbook from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.