Quick answers to common problems

Drupal 7 Views Cookbook

Over 50 recipes to master the creation of views using the Drupal Views 3 module

J. Ayen Green

[PACKT] open source* PUBLISHING
community experience distilled

Drupal 7 Views Cookbook

Over 50 recipes to master the creation of views using the Drupal Views 3 module

J. Ayen Green

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Drupal 7 Views Cookbook

First published: March 2012

Second published: July 2016

Production Reference: 2140716

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84951-434-7

www.packtpub.com

Cover Image by Duraid Fatouhi (duraidfatouhi@yahoo.com)

Credits

Author

J. Ayen Green

Reviewers

Dave Hall

Dlair Kadhem

Deepak Vohra

Acquisition Editor

Usha Iyer

Lead Technical Editor

Meeta Rajani

Technical Editors

Siddhi Rane

Sunith Shetty

Copy Editors

Leonard D'Silva

Aaron Rosario

Neha Shetty

Project Coordinator

Shubhanjan Chatterjee

Proofreader

Aaron Nash

Indexer

Hemangini Bari

Production Coordinator

Shantanu N. Zagade

Cover Work

Shantanu N. Zagade

About the Author

J. Ayen Green (`@accidentalcoder`, `theAccidentalCoder.com`) has developed software since inventing the abacus, and has created websites since [insert name du jour] created the Web. He is a writer and columnist of sorts, a poet of metered sorts, husband, father, friend, and rascal (in the nicest possible way). When not plugged in, Green enjoys nature, dogs, horses, and other critters, riding his Harley, kayaking, spicy food, the arts, and other cultures. He and his wife, Sofía-Aileen, reside in New York City.

This was my third title for Packt, yet was a unique experience. If I may be allowed to make a resolution outside of New Year's, it is to never again start a book about software that is still alpha code (as were both Drupal 7 and Views 3 during my first draft) or undergoing a UI change (as was Views 3 during my second draft. Despite the pitfalls, I had unending support from my publisher. I thank those responsible for a ride smoother than it might have been: Siddhi Rane, Edward Gordon, Chaitanya Apte, Meeta Rajani, Mehreen Shaikh, and Neha Mallik, and the rest of the editorial staff, Rebecca Sawant and Shubhanjan Chatterjee, project coordinators, and all those who have provided production services. Special thanks to Harshit Jhaveri and Sarah Crofton for hearing my pleas for revision.

My technical reviewers showed patience and endurance beyond the normal call of duty.

My wife, Sofia-Aileen, weathered this project and my curmudgeon-like orneriness with cheery, occasionally sarcastic, aplomb. Thanks for knowing what I need before I do.

My thanks to Dries Buytaert for Drupal, to Angie Byron for getting Drupal 7 out the door as quickly as humanly possible, and an especially hardy thank you to Earl Miles, a.k.a merlinofchaos, for Views and his kind and patient assistance the several times I was at wit's end, as well as the team that brought the new UI to life.

About the Reviewers

Dave Hall has worked as an open source consultant and advocate, specializing in web applications, for over a decade. He is currently working as an Architect and Lead Developer for enterprise clients pushing the boundaries of what is possible with Drupal.

Dave has a keen interest in performance, scalability, and security. In 2009, he designed, deployed, and maintained more than 2000 production Drupal 6 sites for a single client.

Dave has contributed to numerous open source projects, including Drupal core, phpGroupWare, StatusNet, and PEAR.

Dlair Kadhem immigrated to the United Kingdom in 1996 having fled war-torn Iraq to seek a better life. In 2005, he went on to graduate with a degree in Electronics Systems Engineering from the University of Essex. Upon completing his degree, he decided to further explore the world of computing, specializing in online services and web applications.

Dlair always had a fascination for creativity and technology. He first encountered computers in 1997 at the age of 13. In the following year, he launched his first website using free web space provided by Freeserve and went on to create ground-breaking online communities.

Dlair spent the early part of his career working in web design, online marketing, and software development. Having gained valuable industry experience, Dlair founded his own business in 2006 with the clear vision of bringing people and technology together through innovation and open source technology. Some of his clients include the BBC, Bauer Publishing, Croydon Council, Department of Health, Harrods, London College of Communication, NHS, and Red Bee Media.

With a keen interest in the evolution of technology, Dlair is currently focused on the rapid innovation taking place in the world of handheld devices and how they affect everyday life. His goal is to build a revolutionary business to create and develop products and services that will increase people's quality of living.

Deepak Vohra is a consultant and a principal member of the NuBean.com software company. Deepak is a Sun Certified Java Programmer and Web Component Developer, and has worked in the fields of XML and Java programming and J2EE for over five years. Deepak is the co-author of the *Apress* book *Pro XML Development with Java Technology* and was the technical reviewer for the *O'Reilly* book *WebLogic: The Definitive Guide*. Deepak was also the technical reviewer for the *Course Technology PTR* book *Ruby Programming for the Absolute Beginner*, and the technical editor for the *Manning Publications* book *Prototype and Scriptaculous in Action*. Deepak is also the author of the *Packt Publishing* books *JDBC 4.0 and Oracle JDeveloper for J2EE Development*, *Processing XML Documents with Oracle JDeveloper 11g*, and *EJB 3.0 Database Persistence with Oracle Fusion Middleware 11g*.

www.PacktPub.com

eBooks, discount offers, and more

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `customercare@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

This book is dedicated to my children, who will never read it, but will think it's cool, nonetheless. This revision is dedicated to my revised children, who are both current and well-tested.

Table of Contents

Preface

Views is a contributed module, originally written by Earl Miles, who is known in the Drupal community as *merlinofchaos*, and is maintained by him and others in the Drupal community.

Views 1 was written during the Summer of Code in 2005, and was available for Drupal 4.6, 4.7, and in 2006 for Drupal 5. For those still running a Drupal 5 site, there is a *Drupal 5 Views Recipes* book version from Packt Publishing.

Views 2 was first released in 2008 for Drupal 6, and was a major improvement on an already very useful module. There isn't a Views 2 recipes book, but you can find many good examples of using Views 2 in *Drupal 6 Attachment Views* from Packt Publishing.

Views 3 was released on December 18, 2011. The original version of this book was based on Views 3 from the period of its being a design on paper through to version 7.x-3.0. At the time of writing of this revision, the current version, and the version that this book relates to, is 7.x-3.13.

What is a view?

Perhaps you've just installed Drupal and the default website that it creates. You've added a few articles and assigned a descriptive term to each: a category. Now, you would like to present the visitors with a page containing articles of a specific category. How do you do that? The short answer is...you can't...yet.

So, you decide to put that idea aside, and instead, present all articles, but sorted by their titles. How do you do it? The short answer, again, is...you can't...yet. (Prior to Drupal 7, the content type Article was called *Story*.)

The fact is that of the laundry list of the thousands of modules available for Drupal, painstaking thought goes into deciding which of them will be present in core, that is, in the code present when first installed, before anything else is added. Generally, the philosophy has been that only the mission-critical functions should be present, that keeping the base platform light and fast is preferable to bloating it with functionality that can be added via contributed modules. Enter Views.

We'll be exploring the capabilities of the Views module throughout the book, so for now, here is a short in-a-nutshell definition of what this module offers.

The Views module provides the capability, via program code or the included user interface sub-module, to define the criteria by which to select content, process it, manipulate it, and format its presentation. It is, at its heart, a query generator, with many additional functional layers.

Many would say that a fully functional Drupal site would be almost impossible to produce without the use of the Views module. Don't take that as a challenge. Of course it would be possible to write custom modules to produce a rich site without using the Views module. Should you bother? It's a "it depends" answer, with many variables. More likely, you could decide that a hybrid environment is needed, with some functionality provided by the Views module, and some via custom modules that do the database queries themselves, especially if the queries are so complex as to exceed the capabilities of Views.

From a MySQL perspective

If you're not familiar with MySQL, it stands for My Structured Query Language, and is the database type most used with Drupal. The database used by a Drupal website contains Drupal's settings as well as the content added to the website.

So, let's say that we have a table in our database, and it is called node, and in this table we keep whatever content we've added to the website. If we wanted to retrieve all content from this table, the command given to MySQL would be:

```
SELECT * FROM node;
```

Which would return all the data stored in that table, each piece of content being a row, a record. If we wanted to retrieve only blog content, the command might be:

```
SELECT * FROM node WHERE node_type='blog';node_type='blog';
```

If we wanted to sort the records by the title of the blog entries, we could use this command:

```
SELECT * FROM node WHERE type='blog' ORDER BY title;
```

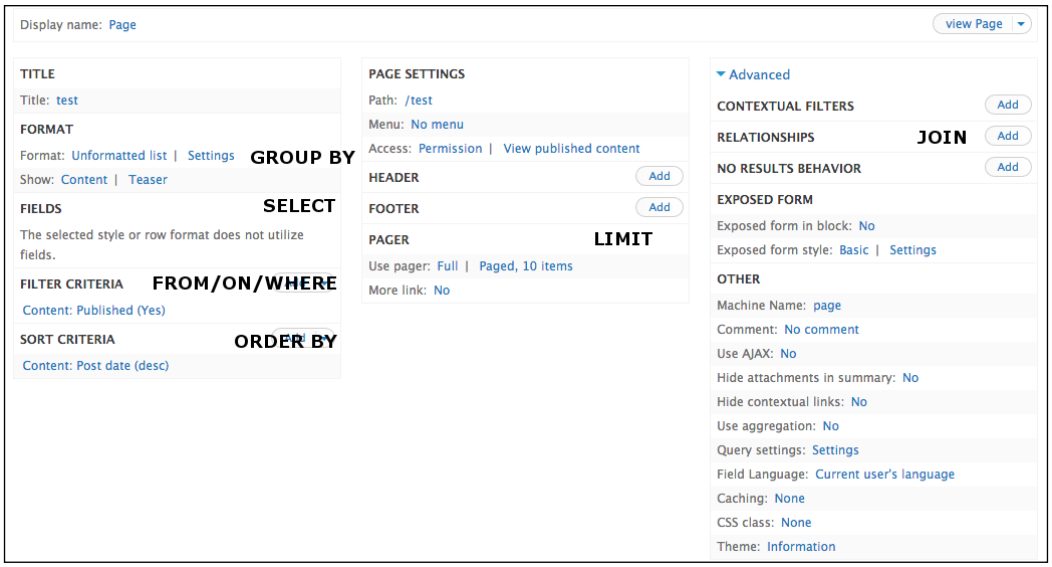
Specifying that only three records are desired would be:

```
SELECT * FROM node WHERE type='blog' ORDER BY title LIMIT 0,3;
```

And finally, if there were another table, `blog-topic`, and the `blog-topic` table used the same identifying value as the node table, *nid* (like a driver's license number), and we want to return its data along with the node data, we would relate the two records to each other, like so:

```
SELECT * FROM node JOIN blog-topic ON node.nid=blog-topic.nid WHERE node_
type='blog' ORDER BY title LIMIT 0,3;
```

Views does all that for you, as well as giving you many options for formatting its output to suit your needs. The following image shows the Views 3 user interface and how the various widgets relate to a typical example of MySQL syntax.



The term View comes up in other places in computing, such as with SQL, but in the context of Drupal, it almost always refers to a dynamic display or feed created with the Views module.

Views is a particularly versatile module in terms of interactions with the developer, who will use it in any or all of three ways:

- Via the UI (user interface) for creating views that are editable by the admin or other authorized users
- From within a custom module, creating and/or invoking a view programmatically
- Indirectly, using modules that, themselves, create programmatic views

Views offers many of the tools necessary for meeting your needs:

- Template hints and model templates
- Several types of default views
- Various display types to meet the needs of the layout, such as page, block, and attachment
- A number of output formats, such as tabular, grid, and list
- Hooks, which allow your code to respond to a Views event
- Pluggable features such as handlers and formatters
- Instantaneous AJAX previewing

And much more. It's no wonder that Views is consistently one of the most popular contributed modules downloaded from Drupal.org (`https://www.drupal.org/project/views`).

We've covered the background of the Views module, and some of what it offers. The most important thing, of course, is being able to create a quick view when you need one. Until you're adept at creating one from scratch, some instructions will help, and they start on the next page.

Introduction

In the beginning, there was the concept pitch to have a book on Views available for purchase coinciding with the release of Drupal 7. This seemed like a clever idea at the time when the pitch is abstract, before the writing begins. Things change.

The problem with having the book on the shelves simultaneous to Drupal being released is that, working backwards from that day to when the writing began, Drupal 7 was in Alpha. Even more awkward was the fact that while Drupal 7 was in Alpha, Views for Drupal 7 was somewhere off on the horizon. After all, with Alpha being the time when architecture and feature functions are not yet nailed down, it would be somewhat difficult to nail down a contributed module deeply interwoven with both. Then, how do you begin developing the book?

Simple. You use Views for Drupal 6, with the intention of making whatever changes are necessary once the version for Drupal 7 is available.

That was my modus operandi. I plodded along, happily, creating "recipes" with step-by-step instructions and screen captures, and soon found my manuscript over 80% complete...and then came the chat:

Me: "I'm getting near the end of the book. When do you think Views will be ready?"

Earl: "Did you know that the Views UI is being redesigned?"

Me: "Redesigned, as in a little different, or vastly different?"

Earl: "'Vastly' is a good description."

Me: "Can I get my hands on the code?"

Earl: "Not yet, but I can point you at the designs."

So, off I went and looked at the designs. Have you ever experienced a vastly (good word here, too) sinking feeling, like remembering you left the note from your girlfriend or boyfriend on the kitchen table and it's very likely that while you're realizing you did, your mother is reading it? Yeah, like that. In terms of a manuscript where 99% of it is detailed instructions regarding the Views UI along with screen shots of the same, perhaps 98% of it needed to be rewritten.

I plodded along, nowhere near as happy as I had been during the first draft, and created a second one. It's not quite the same as creating a second draft of a novel, because all of the Views being presented as "recipes" need to be created one step at a time. It's laborious the first time...tedious the second. Still, the designs were easy to follow, and I persevered.

Finally, I finished the second draft. Remember what I wrote, earlier, about Alpha being the time when architecture and feature functions are not yet nailed down? Well, the same applies to the Views redesign. When the code became available, it was apparent that it had diverged in places from the design, and while the divergence wasn't vast, it was substantial. That introduced the need for a third draft, in places.

A need for a third draft was enough on its own, but the issue was greatly compounded by the fact that the galleys had already been produced, which means the manuscript had been styled to print, and was now in PDF form.

The result that was with so many versions of the files flying by, in many cases the wrong versions were printed (it would be nice to have the manuscript in Git).

I have heard from many readers in various stages, ranging from annoyance to apoplexy, about issues with the usability of the book. I created errata pages on my blog, and though that was the best I could do at the time, it's likely that referring to an errata page made things more annoying rather than less so.

Thus, with the greatest and sincerest apology, and the weight finally gone from my shoulders, here again is the Views cookbook for Drupal 7, revised and tested, the way it was meant to be, originally.

–J. Ayen Green

What this book covers

Chapter 1, *Modifying Default Views*, gives an introduction to the Views UI by modifying some of the views that come with the module in order to make useful versions of them.

Chapter 2, *Basic Custom Views*, covers creating elementary views and how to get them to provide the information you need.

Chapter 3, *Intermediate Custom Views*, goes beyond the basics to introduce concepts such as presenting teasers for a specific type of content, adding a header and footer, using AJAX for page changes, and producing custom links.

Chapter 4, *Creating Advanced Views*, covers advanced topics such as the use of multiple displays, using dynamic filters with depth, and restricting access to views.

Chapter 5, *Theming Views*, shows you the various ways to manipulate the output of a view so that it has the look that you need.

Chapter 6, *Programmatic Views*, shows how to create a view from within the module code rather than using the UI.

Chapter 7, *But Wait – There's More*, covers some of the tools for administering your views environment.

Appendix A, *Installing Views*, provides instructions for installing the Views module.

Appendix B, *Bundles*, gives instructions for creating the various content types and other Drupal elements used in the recipes.

Appendix C, *Resources*, provides the information needed to make obtain the recipes' ingredients, Views, content types and code, already prepared, rather than having to cook them yourself.

What you need for this book

You will need a reasonably advanced computer and an Internet connection. All software required to do the recipes can be freely obtained from `drupal.org`. If you will be using an existing Drupal site rather than creating one, you will need administrative access to it.

Who this book is for

This book is for developers or technically proficient users who are fairly comfortable with the concepts behind websites and the Drupal environment.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows. "Enter `Alternate front page` for the view name and click on the **Continue** button."

A block of code is set as follows:

```
<style type="text/css">
#cc-container {
  width: 180px;
}
.cc-odd, .cc-even {
  padding: 6px;
  border: 4px solid black;
  width: 120px;
  position: relative;
  text-align: center;
}
.cc-odd {
  left: 0;
  background-color: #aaa;
}
.cc-even {
  left: 60px;
  background-color: #eee;
}
.cc-value {
  font-size: 36px;
}
</style>
```

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

```
<?php foreach ($rows as $id => $row): ?>
  <div class="cc-<?php echo ($ctr % 2) ? 'odd' : 'even'; ?>">
      <?php $ctr--; ?>
      <div class="cc-value"><?php echo $ctr; ?></div>
      <div class="<?php print $classes_array[$id]; ?>">
        <?php print $row; ?>
      </div>
  </div>
<?php endforeach; ?>
```

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: "Click on the **Title** link in the **Title** box and change the title to **Recent article comments**, and then click on the **Update** 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 disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

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

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `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

You can download the example code files for this book 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.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.

5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

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 could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

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

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

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

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Modifying Default Views

In this chapter, we will cover:

- Selecting recent comments for a specific node type
- Focusing the archive view
- Changing the front page view
- Selecting the glossary view entries for a specific user
- Creating an attached menu for the taxonomy term view
- Reporting a tracker activity for a certain user role

Introduction

The Views module comes with a number of useful predefined views. Not only are they there for you to use, but they are ready to be edited to meet whatever special needs arise.

Since these views are ready to use the moment the module is enabled, the steps necessary to make some changes to them are less than those needed to create new custom views, and so using them is a logical choice for our first chapter.

Selecting recent comments for a specific node type

The **Recent comments** view provides a block containing comments that link to a page providing additional comment content. The default behavior of this view is to select comments from all content types. We will edit this view to enable us to display comments for a specific content type.

Getting ready

1. Ensure that:
    1. Your site has at least two types of content (the default **Page** and **Article** types are fine) and that you have access to both.
    2. The content you wish to use has comments.
2. Navigate to the **Views** page (`admin/structure/views`) and click on the **Enable** link for the **Recent comments** view.
3. Click the down arrow beside the **Edit** link for the same view, and from the drop-down menu click on the **Clone** link.
4. Enter `Article comments recent` for the view name and click on the **Continue** button.

How to do it...

We will now have been placed in an edit window for the new cloned view that we have created.

1. Click on the **Add** button in the **Filter Criteria** section.
2. A window titled **Add filter criteria** will open.
3. Click on the filter select box, which currently shows **All**, and select **Content**.
4. Scroll down to **Content: Type** and check the box. At the bottom of the window, you will now see **Selected: Content: Type**. Click on the **Apply (all displays)** button at the bottom of the **Add filter criteria** box to reveal a new window, titled **Configure filter criterion: Content: Type**.
5. In the **Content Types** section, click on the checkbox next to the content type that you want the view to select. In our case, we will click on the box for article.
6. Click on the **Apply (all displays)** button.
7. Click on the **Recent comments** link in the **Title** section and change the title to `Recent article comments`, then click on the **Apply (all displays)** button.
8. Click on the path link `/comments/recent` in the **Page Settings** section.
9. Change the path to `article-comments/recent` (you don't need to enter the leading slash, it's provided) and click on the **Apply** button.
10. At the top of the page, click on the **Block** display button.
11. Click on the block name **None** link in the **Block Settings** section which opens a window titled **Block: Block admin description**.
12. Change the block admin description to the `Recent article comments` view and click on the **Apply** button.

13. Now back in the main view window, click on the **Save** button at the top right of the screen.
14. Navigate to `article-comments/recent`.

How it works...

Most of the views that you will create will probably be Node views, views that use nodes as the primary source of data. This view uses a different entity type: comments. The relationship that was already in place in the view links each selected comment to the applicable content.

The filter that was already present when we cloned the default view, **Published** or **Admin**, limits the selection of comments to published content, unless the user has admin capability, in which case all comments will be selected. We added another filter, **Content: Type**, which we configured to further limit the selection to comments made on articles. Therefore, even though there was another piece of content with a comment, it was not displayed, because it was not an article. However, if we run the original view (`comments/recent`), we see the result of not having added the additional filter:

There's more...

As you have seen, more than one filter can be applied to the selection of content. One additional filter to consider is **Comment: Approved**, which limits the displayed comments to those that have been approved, or not, depending on the chosen setting.

Later in the book, we will modify this view so that instead of only showing Articles, we can specify which type of content by changing the URL.

Focusing the archive view

The **Archive** view displays a list of links that are the months in which content was published.

Each link leads to a page that presents teasers of each of the pieces of content published that month. We will add a filter to the view so that only the current user's content is considered.

Getting ready

1. Ensure that your site has content posted by more than one author (for testing purposes).
2. Navigate to the **Views** page (`admin/structure/views`) and click the **Enable** link for the **Archive** view.
3. Click on the down arrow beside the **Edit** link for the same view, and from the drop-down menu, click on the **Clone** link.
4. Enter `Archive for current user` as the view name and click on the **Continue** button.

How to do it...

We will edit the clone we have created, and make some modifications to it to provide a new view.

1. Click on the **Advanced** link.
2. Click on the **Add** button in the **Relationships** section.
3. Check the box for **Content: Author** and click on the **Apply (all displays)** button.
4. In the **Configure Relationship** window that appears, click on the **Apply (all displays)** button.
5. Click on the **Add** button in the **Filter Criteria** section and a window titled **Add filter criteria** will open.
6. Check the box for **User: Current**, then click on the **Apply (all displays)** button.
7. A window titled **Configure filter criterion** will open. Click on the radio button for **Yes** in answer to **Is the logged in user**, then click on the **Apply (all displays)** button.
8. Click on the **Monthly archive** link in the **Title** section, change the title to `User monthly archive` and click on the **Apply (all displays)** button.
9. Click on the `/archive` link for the path in the **Page Settings** section, change the path to `user-archive` (no leading slash) and click on the **Apply** button.
10. Click on the **Block** button at the top of the page.
11. Click on **None** for **Block name** in the **Block Settings** section, change the block admin description to `User archive` and click on the **Apply** button.
12. Click on the **Save** button.
13. Navigate to `user-archive`:

How it works...

This is an example of a view that uses content as the data source. The original filter limits the selection of content titles to those that are published, and produces a links list of the months in which they were published. We added another filter, which further limits the selection to content created by the current user.

Changing the front page view

The composition of the front (home) page in Drupal can be themed to alter its appearance, but the structure and sources of its content are determined for you. If you leave things as they are, the content area of the front page will consist of whatever nodes you have promoted, in any quantity up to the maximum that is set in the site configuration, and that's pretty much the extent of your control... or is it?

We're going to create a new front page.

Getting ready

For this recipe, we're going to be using a custom content type, *Gallery*, the details of which are given in *Appendix B*, *Bundles*. Feel free to duplicate it, create a content type more meaningful to you, or just use one of the existing content types. You can also enable this content type as a Feature (see *Appendix C*, *Resources*). The content type also makes use of an image style so that the image in each piece of content can be presented in a uniform manner.

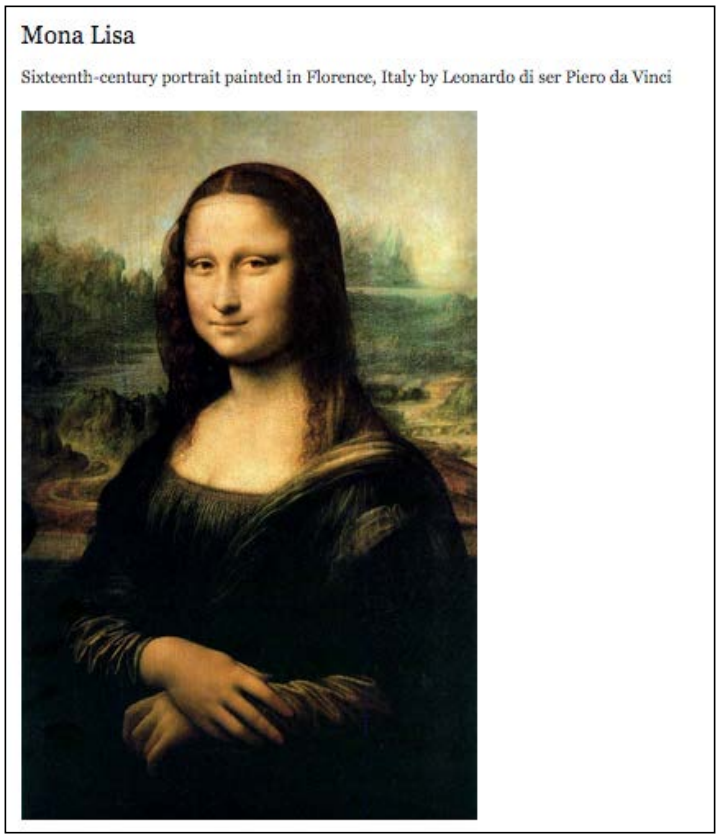
1. Create two entries of the content type you choose to use.
2. Navigate to the **Views** page (`admin/structure/views`) and click on the **Enable** link for the **Front page** view.
3. Click on the down arrow beside the **Edit** link for the same view, and from the drop-down menu, click on the **Clone** link.
4. Enter `Alternate front page` for the view name and click on the **Continue** button.

How to do it...

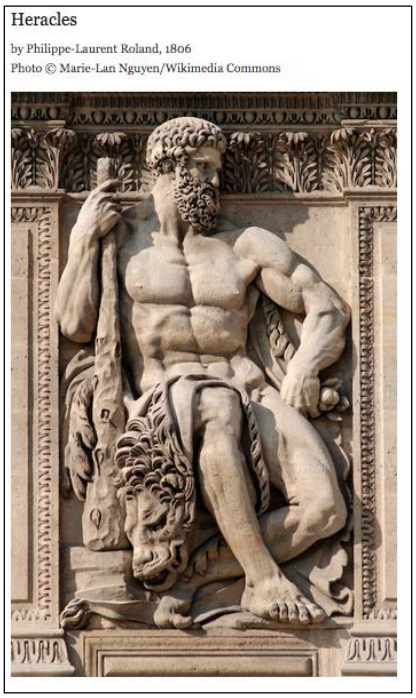
We will edit the clone we have created, and make some modifications to it to provide a new view.

1. Click on **Full** next to **Use pager** in the **Pager** section, select **Paged output**, **mini pager**, and click on the **Apply (all displays)** button.
2. Change **Items per page** to `1` and click on the **Apply (all displays)** button.
3. Click on the **Add** button in the **Filter Criteria** section.
4. Scroll down to **Content: Type**, check the box, and click on the **Apply (all displays)** button.

5. In the **Configure filter criterion** window that opens, check the box for the **Gallery** content type and click on the **Apply (all displays)** button.
6. Click on the **Save** button.
7. Navigate to the site information page (`admin/config/system/site-information`).
8. In the text box for the path to the default front page, change **node** to **frontpage** and click on the **Save configuration** button.
9. Navigate to the site home page to view the page:



10. Clicking the pager will load the next piece of content:



How it works...

The default version of the front page view is designed to provide precisely what the Drupal front page does. We created a view that shows only content of the custom *Gallery* type, composed primarily of an image, with one piece of content per page. Then we let Drupal know that this new view should be used as our front page.

Selecting the glossary view entries for a specific user

The glossary view presents a list of all content organized by the first letter of the title. This is a convenient view for site visitors that know the name of the content they are seeking, or who simply want to browse. We're going to give the user the ability to browse the content created by a specific author.

Getting ready

1. Ensure that your site has content posted by more than one author (for testing purposes).
2. Navigate to the **Views** page (`admin/structure/views`) and click on the **Enable** link for the **Glossary** view.
3. Click on the down arrow beside the **Edit** link for the same view, and from the drop-down menu, click on the **Clone** link.
4. Enter `Author glossary` for the view name and click on the **Continue** button.

How to do it...

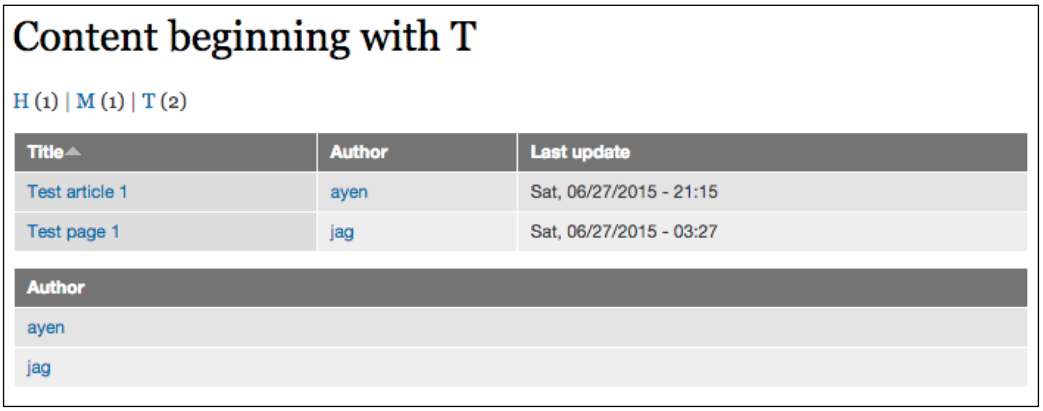
The glossary view, and now the clone that we created, has two displays: the page display that lists content, and an attachment display that lists each letter of the alphabet for which there is content, along with the number of applicable pieces. We will make some changes to the existing displays and create an additional one that will provide the author list.

1. Click on the **+Add** button in the **Displays** section at the top of the page and select **Attachment** from the list.
2. Click on the **Attachment** link next to display name and change the name to `New attachment`, so we can tell one display button from the other, and click on the **Apply** button.
3. Click on the **Add** button in the **Sort criteria** section, select **This attachment (override)** from the **For** select box, check the box for **User: Name** and click on the **Apply (this display)** button, and in the configuration window that opens, click on the **Apply (this display)** button.
4. In the **Attachment Settings** section, click on **Before** next to **Attachment position**, select the radio button for **After**, and click on the **Apply** button.
5. Click on the **Not defined** link next to **Attach to** in the **Attachment Settings** section, check the box for **Page**, and click on the **Apply** button.
6. In the **Fields** section, click the **Content: Title** link, select **This attachment (override)** from the **For** select box, and click on the **Remove** button.
7. Do the same and remove the **Content: Updated date** field.
8. Click on the **Advanced** link to open that section, then click on the **Settings** link next to **Query settings** in the **Other** section, select **This attachment (override)** from the **For** select box, check the box for **Distinct**, check the box below it for **Pure Distinct**, and click on the **Apply (this display)** button.

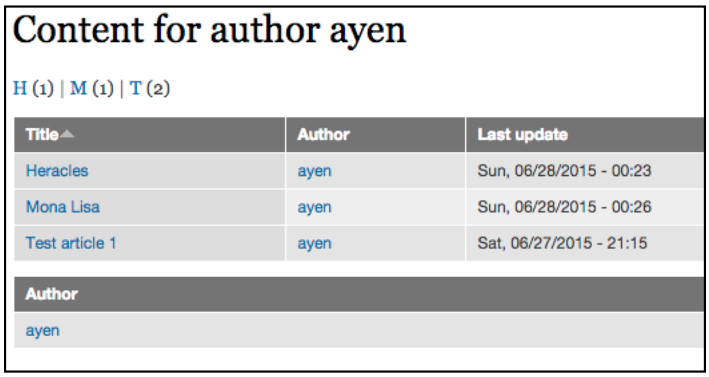
   That is the work needed for the attachment itself. Now, we need to make a few minor changes to the settings from the page display.

9. Click on the **Page** display button.

10. Click on the **Advanced** link to expand that section.
11. In the **Other** section, click on the **Yes** link for **Use AJAX**, select **No**, and click on the **Apply** button.
12. Click on the **Content: Title** link in the **Contextual filters** section, select **This page (override)** from the **For** select box, and in the section titled **When the filter value is Not in the URL** ensure provide default value of type fixed value of **a** is the setting.
13. In the section **When the filter value is in the URL**, check the box for **Override** title, enter `Content beginning with %1`, and click on the **Apply (this display)** button.
14. Click on the **Add** button in the **Contextual filters** section, select **This page (override)** from the **For** select box, scroll down and check the box for **User: Name**, and click on the **Apply (this display)** button.
15. In the subsequent configuration window, in the **When the filter value Is in the URL** section, check the box for **Override title**, enter `Content for author %2`, and click on the **Apply (this display)** button.
16. Click on the **(author) User: Name** link in the **Fields** section, ensure that the checkbox for **Link this field to its user** is not checked, scroll down and click the **Rewrite Results** link, check the box for **Output this field as a link**, enter `author-glossary/all/[name]` as the link path, then click on the **Apply (all displays)** button.
17. Click on the `/glossary` link next to path in the **Page settings** section, change the path to `author-glossary` (no leading slash) and click on the **Apply** button.
18. Click on the **Normal: Glossary** link next to **Menu** in the same section, change the title from **Glossary** to `Author glossary`, and click on the **Apply** button.
19. Click on the **Save** button.
20. Navigate to the home page. First invoke the view simply by click on the **Author Glossary** link in the navigation menu, and then making a selection of one of the letters shown. Notice the new attachment below the list of content. Yours will vary from the following image, based on the authors on your site:



21. Then, click on the name of an author in the author attachment.



How it works...

We added a new attachment display to the existing view, giving it three displays. The first is the page display, which is the section of the view containing the information about content. The second is the original attachment, which appears first on the page, and is formatted as a summary of the available titles using the first letter of the title, giving a total for each letter. The new attachment appears last, and contains the names of each author who has content on the site.

We specified that the new attachment would inherit two arguments from the page display (which receives the arguments), those being the additional portions of the URL. The first argument is the title of a piece of content, or "all" to specify that all should be used. The second argument is the name of an author, or is omitted entirely for all authors to be selected. We turned off AJAX processing of the links, because when using AJAX the page is not reloaded, so the new arguments are not picked up for the title.

The link produced for each author name links back to the same view, providing "all" as the title being searched (so, all titles) and the author's name. Thus, the page URL when selected from the menu item appears as `author_glossary` with no arguments, in which case all records are retrieved; when an author link is clicked, `a` is inserted as a default; and `author_glossary/all/author_name`, where all titles belonging to that author are retrieved.

There's more...

Attachment displays are a boon to the value of your views and your site. You can find several examples of their power in my book *Drupal 6 Attachment Views*, also by Packt Publishing.

Creating an attached menu for the taxonomy term view

In this recipe, we will be using an attachment display as a menu.

Getting ready

1. Ensure that some of your content has taxonomy terms (tags) assigned to it.
2. Navigate to the **Views** page (`admin/structure/views`) and click on the **Enable** link for the **Taxonomy term** view.
3. Click on the down arrow beside the **Edit** link for the same view, and from the drop-down menu, click on the **Clone** link.
4. Enter `Taxonomy term menu` for the view name and click on the **Continue** button.
5. The view has an additional display that we do not need, so let's remove it. Click on the button for the **Feed** display, then the down arrow beside the **clone Feed** link, and delete the feed.
6. Click on the **Save** button.

How to do it...

We will be adding an attachment display that will be used as a menu for the view.

1. Click on the **+ Add** button in the **Displays** section and select **Attachment**.
2. Click on **Content** beside **Show** in the **Format** section, select **This attachment (override)** in the **For** select box, change the setting to **Fields**, click on the **Apply (this display)** button, and then the **Apply (this display)** button in the subsequent configuration form.
3. Click on the **Advanced** link to expand that section, click on the link for each of the **Contextual filters**, and then the **Remove** button.
4. Click on the **Add** button in the **Relationships** section, select **This attachment (override)** from the **For** select box, scroll down and check the box for **Content: Tags (field_tags)**, and click on the **Apply (this display)** button and again in the subsequent configuration window.
5. Click on the **Add** button in the **Fields** section, select **This attachment (override)** from the **For** select box, scroll down to **Taxonomy term: Term ID** and check the box, then click on the **Apply (this display)** button, and in the subsequent configuration window, check the box for **Exclude from display** and click on the **Apply (this display)** button.

6. Again, click on the **Add** button in the **Fields** section, select **This attachment (override)** from the **For** select box, scroll down to **Taxonomy term: Name** and check the box, then click on the **Apply (this display)** button.
7. In the subsequent settings box, uncheck the **Create a label** checkbox and click on the **Rewrite Results** link to expand that section.
8. Check the box for **Output this field as a link**, and enter **taxonomy/term-menu/[tid]** as the link path.
9. Click on the **No Results Behavior** link to expand that section, check the box for **Hide if empty**, leaving the **Hide rewriting if empty** box checked as well, and click on the **Apply (this display)** button.
10. Click on the link for each of the two entries in the **Sort criteria** section, select **This attachment (override)**, and click on the **Remove** button.
11. Click on the **Add** icon to add a sort criterion, check the box for **Taxonomy term: Name**, and click on the **Apply (this display)** button.
12. Ensure **Ascending** is selected in the subsequent configuration window, and click on the **Apply (this display)** button.

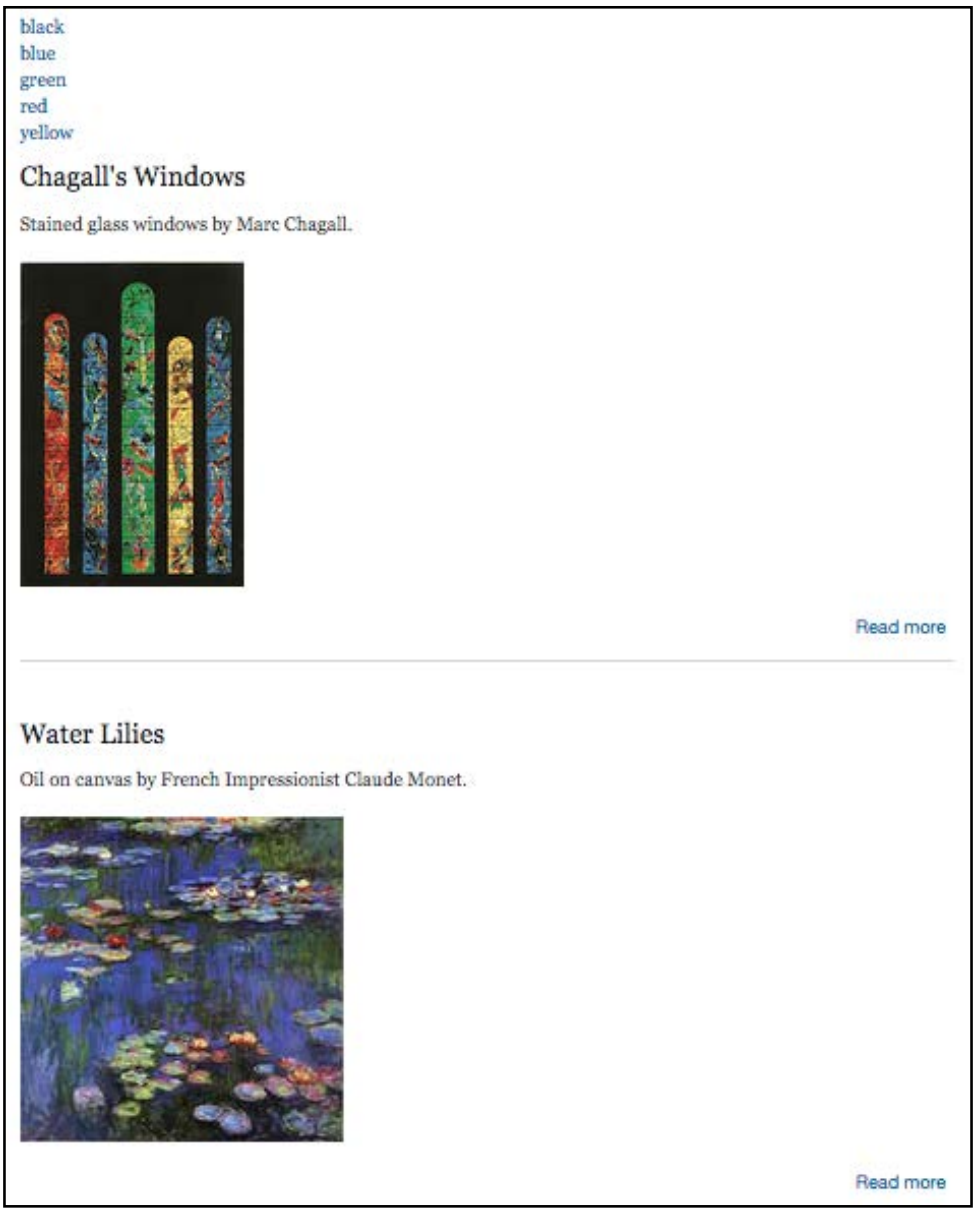
    Those changes took care of the data requirements of the new attachment display. Now, we need to make some changes to the structural parts of it before we finish.

13. Click on the **10 items** link next to the **Display a specified number of items** link in the **Pager** section, change the **10** to **0**, select **This attachment (override)** from the **For** select box, and click on the **Apply (this display)** button.
14. Click on the **None** link in the **Title** section, select **This attachment (override)** from the **For** select box, enter `Terms as the title`, then click on the **Apply (this display)** button.
15. In the **Attachment Settings** section, click on the **Yes** link next to **Inherit contextual filters**, uncheck the check box for **Inherit**, then click on the **Apply** button.
16. Click on the **Not defined** link next to **Attach to** in the **Attachment Settings** section, check the box for **Page**, and click on the **Apply** button.

    Almost done now. The original view had dynamic filters that we removed. We need to add a new one now.

17. Click on the **Page** button in the **Displays** section.
18. Click on the **Advanced** link to expand that section and then the **Add** button for **Contextual filters**, select **This page (override)** from the **For** select box, check the box for **Content: Has taxonomy term ID**, and click on the **Apply (this display)** button.
19. In the subsequent configuration window, in the **When the filter value is Not in the URL** section, select **Provide default value**, in the **Type** select box select **Fixed value**, enter `all` in the **Fixed value** textbox, then click on the **Apply (this display)** button.

20. Click on the path link next to **Path** in the **Page Settings** section, change the path to `taxonomy/term-menu`, and click on the **Apply** button.
21. Click on the **Save** button and navigate to `taxonomy/term-menu` to see the results. The taxonomy terms will be listed along with all content, but clicking one of the taxonomy terms will result in only content tagged with that term being shown:



How it works...

Content can have taxonomy terms, often known as "tags," associated with it. We took a view that uses a node's taxonomy in a different way and edited it to present the terms associated with published nodes as a menu from which we can choose one as a filter. Most of the work we did was removing settings we would not need or adding new ones. The three primary changes were adding the attachment display to use as a menu, changing the contextual filter for the page display to be the taxonomy term that we select via its link, and setting the term to output in the attachment display as a link pointing back to our view with the term name appended.

Drupal treats additional information in a URL as arguments, so, for example, if a page address is `my/page` and the URL is `my/page/2/even` the values `2` and `even` are processed as arguments, or parameters, to be passed into the view.

There's more...

The next steps for a real site would be to theme the attachment display to list the terms horizontally, perhaps add a pager for both it and the page display, and maybe change the position setting for the attachment display to *after* rather than *before* or have it appear side by side with the content.

Reporting a tracker activity for a certain user role

The default tracker view lists posts created by users. We will create an exposed filter to allow the admin to filter the view by user role. Filters require certain criteria to be met in order for records to be selected. Often, the value of the filter is one that can remain constant, such as requiring that content be published in order to be visible. There are times, however, when you will want the ability to change the value that is filtered. Rather than having to edit the view each time you want this value to change, you can simply elect to have the view provide a widget with which the value can be supplied.

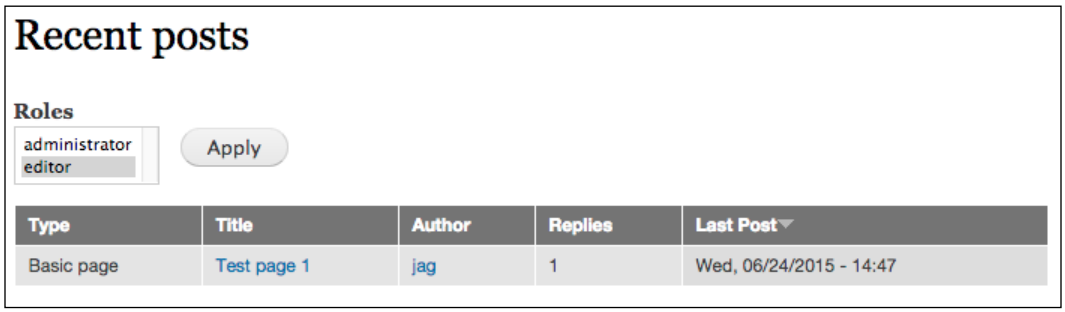
Getting ready

1. Ensure that your site has content posted by more than one author, and that the authors represent more than one role.
2. Navigate to the **Views** page (`admin/structure/views`) and click on the **Enable** link for the **Tracker** view.
3. Click on the down arrow beside the **Edit** link for the same view, and from the drop-down menu, click on the **Clone** link.
4. Enter `Tracker role` for the view name and click on the **Continue** button.

How to do it...

1. Click on the **Add** button in the **Filter criteria** section.
2. Check the **User: Roles** checkbox and click on the **Add (all displays)** button.
3. In the new configuration window that opens, check the box for **Expose this filter to visitors...**, check the box for **Allow multiple selections**, and click on the **Apply (this display)** button.
4. Click on **/tracker** next to **Path** in the **Page Settings** section, change the path to `tracker-role` (no leading slash) and click on the **Apply** button.
5. Click on **Normal: Recent posts** next to **Menu** in the **Page Settings** section, change the **Title** to `Recent posts with selectable role` and click on the **Apply** button.
6. Click on the **Advanced** link to expand that section.
7. Click on the link for **Content: User posted or commented** in the **Contextual filters** section, select **This page (override)** from the **For** select box, and then click on the **Remove** button.
8. Click on the **Add** button in the **Contextual filters** section, scroll down and check the box for **User: Roles**, then click on **Apply (this display)**.
9. In the **When the filter value is Not in the URL** section, ensure that **Display all results for the specified field** is selected, and click on the **Apply (this display)** button.
10. Click on the **Save** button.

11. Navigate to **tracker-role** and you will see the exposed filter as shown in the following screenshot, which you can use to choose to see nodes created by users of a specific role:



How it works...

We changed the contextual filter for the view to use the author's role. Because this is a view based on content and not users, the information will come from content records and not user records, which means that the list will only contain roles for which content exists. In choosing the role in the exposed filter, its `rid` (role ID) gets passed in the URL back to the view, which then lists only the content that has been created by a user with that role.

2
Basic Custom Views

In this chapter, we will cover:

- Selecting all content
- Creating a paged block display
- Creating a dynamic links display block
- Creating a random ad block
- Using a view content filter
- Providing a user view for admins
- Winning the argument
- Using views to create a bulleted list
- Creating bulleted lists using multiple content types

Introduction

In this chapter, we're going to begin creating custom views. We'll be working with simple examples, which you can then alter, expand on, or combine to suit your own needs.

Selecting all content

Normally, you will want to select only published content, because you want users to see only nodes that have been published. However, as an admin, you could very well want to view unpublished content as well. Fortunately, there is a way to account for both requirements using a filter made just for this job.

Getting ready

Ensure that your site has both published and unpublished content.

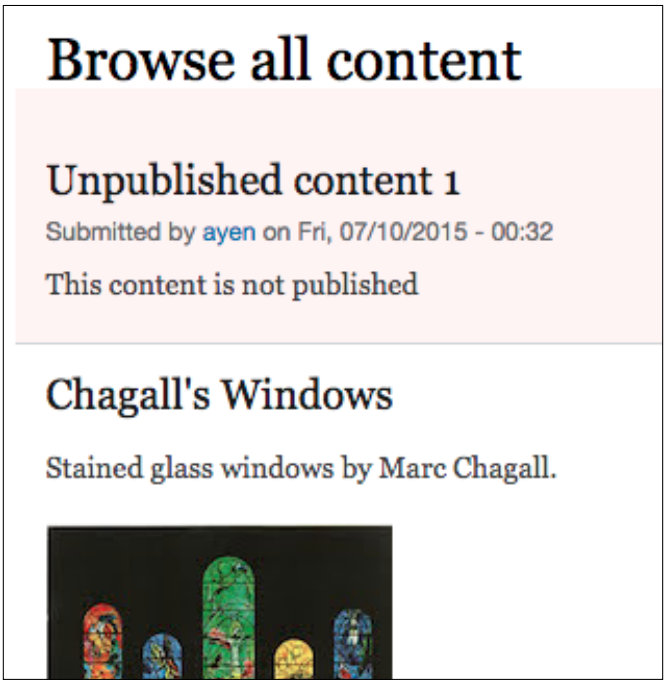
How to do it...

On the views list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+Add new view** link.
3. Enter `Browse all content` as the **View name**. Check the box for **Description** and enter `Browse all content if admin, all published if not`.
4. In the **Create a page** section, enter `Browse all content` as the page title.
5. In the **Display format** select boxes, select **Unformatted list**, **teasers**, **without links**, and **without comments**, respectively.
6. Check the **Create a menu** link checkbox, in the **Menu** select box, select **Main menu** and leave `Browse all content` as the link text, then click the **Continue and edit** button.

On the view edit page:

1. Click the **Content: Published (Yes)** link in the **Filter Criteria** section, and then click the **Remove** button.
2. Click the **Add** button in the **Filter Criteria** section, check the box next to **Content: Published or Admin**, and click the **Apply (all displays)** button in this window and the window that opens subsequently.
3. Click the **Save** button at the top of the screen.
4. Navigate to the front page.
5. Click the **Browse all content** menu tab while logged in as the admin—the unpublished content will typically show as pink unless this has been overridden in your theme.



6. Log out and click the **Browse content link** again—you are now seeing this view as a typical site visitor would, and only the published content shows.

How it works...

By default, a content view will select all available content that is published, sorting backwards from the most recent. We changed our view to select all content, published or not, if the user requesting the view is the admin, but only published content otherwise. When creating the view, we changed the style of the output to present the content as teasers, and finally, we added it to the site's main menu.

There's more...

The main concept to take away from this view is the use of a filter. There are many criteria on which a view can be filtered, and we will use several throughout the book. More than one filter can be applied to the same display. For example, in addition to the filter we added, we could have also required the content to have been promoted to the front page, or selected only specific content types.

Creating a paged block display

The output presented by a view is typically seen in the content area of the page, but views are also capable of creating displays that appear as blocks in any selectable region of your theme, including the content area. We're going to create a block that provides a paged content display for a specific content type.

Block displays

The next few recipes make use of block displays. The block displays of views are not necessarily any different in appearance than other blocks. The difference is that unlike static blocks, their content is dynamic. Technically, the block does not exist until the page is created (an exception being that if caching is used, the block's contents may be cached). At that point, the data for it is retrieved from the database when the block is rendered.

Getting ready

1. This recipe uses a custom content type, **Sponsor**, the details of which are in *Appendix B*, *Bundles*.
2. Create a node or two of this content type.

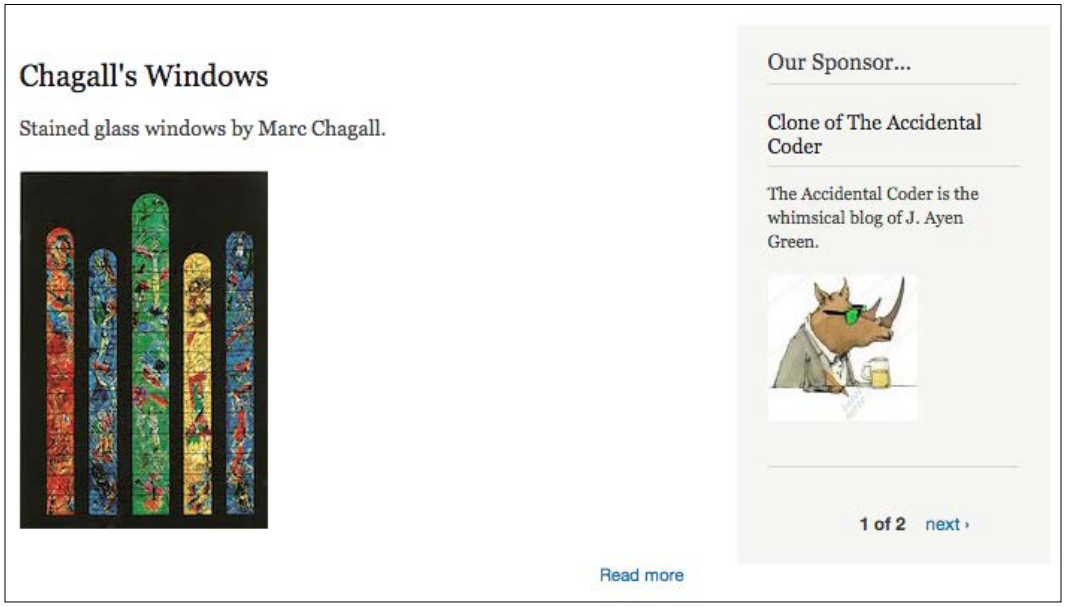
How to do it...

1. Navigate to the views list (`admin/structure/views`). Click the **+Add new view** link. Enter `Sponsors` as the **View name**. Check the **Description** box and enter `Views related to Event sponsors`.
2. Uncheck the box for **Create a page**.
3. Check the box for **Create a block**, enter `Our Sponsor...` as the **Block Title**.
4. In the **Display format** select boxes, select **Unformatted list**, **Teasers** (which will cause two more boxes to appear), **without links**, and **without comments**, respectively. Click the **Continue & edit** button.

That creates the view, and now we need to have it do something, so on the view edit page:

1. Click the **Add** button in the **Filter Criteria** section, check the box for **Content: Published or admin** and the one for **Content: Type**, and click the **Apply (all displays)** button.
2. Check the box for **Apply (all displays)**.
3. Check the box for **Sponsor** and click the **Apply (all displays)** button.

4. Click the link for **Content: Published (Yes)** in the **Filters** section, and then the **Remove** button.
5. Click **Display a specific number of items | 5 items** next to **Use pager** in the **Pager** section, select paged output, mini pager, and click the **Apply (this display)** button.
6. Set the **Pager ID** to **1**, the **Items to display** to **1**, and click the **Apply (this display)** button.
7. Click **None** in the **Block Settings** section, enter `Sponsor ad` in the text box, and click the **Apply** button.
8. Click the **Save** button.
9. Navigate to the **Blocks** admin page (`admin/structure/block`).
10. Scroll down to **Sponsor ad** in the **Disabled** section.
11. Select **Sidebar second** from the select box.
12. Click the **Save** blocks button.
13. Navigate to the front page to view the block.



How it works...

We created a block display for a content view, and limited its selection to sponsor content that is published (unless the user is the admin).

We changed the pager number to 1, because it is likely that the content area will be using pager 0, and each pager on a page must have a unique number.

We specified that one record be shown at a time, with a mini pager below it for navigation. We assigned the block to the right (second) sidebar. If there are 10 nodes that meet our criteria, each will be shown in turn as the next link in the pager is clicked.

Creating a dynamic links display block

Views are capable of providing displays that appear as blocks in any selectable region of your theme. We're going to create a block that provides a dynamic list of links for a specific content type.

Getting ready

This recipe is based on the view created in the previous recipe.

How to do it...

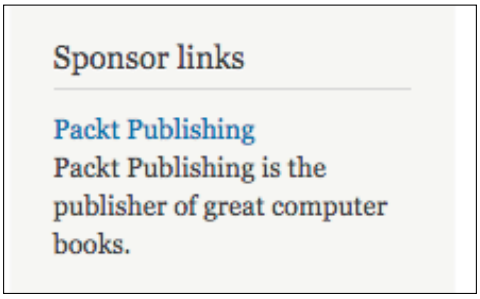
On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **Edit** link for the **Sponsors** view.

On the view edit page:

1. Click the **+ Add** button next to the **Block** display button at the top of the page, and choose **Block** from the dropdown list.
2. Click the **None** link beside **Block name** in the **Block Settings** section and enter `Sponsor links`, then click **Apply**.
3. Click the **Block** link in the next to **Display** name, enter `Sponsor links` for the name, and click **Apply**.
4. Click **Our sponsors** in the **Title** section, select **This block (override)** from the **For** select box, change the title to **Sponsor links**, and click **Apply (this display)**.
5. Click the **Content** link next to **Show** in the **Format** section, select **This block (override)** from the **For** select box, select **Fields**, and click the **Apply (this display)** button.
6. Click **Apply (this display)**.

7. Click the **Add** button in the **Fields** section, select **This block (override)** from the **For** select box, check the box for **Content: Body**, and click the **Apply (this display)** button.
8. Uncheck the **Create a label** checkbox, select the **Summary or trimmed from the Formatter** select box, and click **Apply (this display)**.
9. Click **Mini** in the **Pager** section, select **This block (override)** from the **For** select box, select **Display a specified number of items**, and click **Apply (this display)**.
10. Set **Items to display** to **5**, **Offset** to **1**, and click the **Apply (this display)** button.
11. Click the **Save** button. Navigate to the **Blocks** admin page (`admin/structure/block`).
12. Scroll down to **Sponsor links** in the **Disabled** section. Select **Sidebar second** from the select box, and click the **Save blocks** button.
13. Navigate to the front page to view the block.



How it works...

We used the **Sponsor** view we had created in the preceding recipe, created an additional block display, and limited its selection to the sponsor content types that are published (or all, in the case of the user being admin). We set the maximum records to be shown to 5. We then set the offset to 1. The reason for this is that the **Sponsor ad** block from the preceding recipe will display the most recent sponsor node, so that content does not need to be included in this block; the offset setting will have views skip one record and start the list with the second most-recent sponsor.

Creating a random ad block

Anyone who has spent time on the Web has seen advertisements, often in the form of a banner. In a Drupal site, an ad is typically in a block, which is most often a region of the screen other than the main content area. Adding content to a block manually results in a static ad, which could be all that is required, but if dynamic ads are needed, creating a view with a block display is a good way to go. We're going to create a block that presents an ad for a randomly-selected product, and have it display within the content area.

Getting ready

1. This recipe uses a custom content type, **Product**, the details of which are in *Appendix B*, *Bundles*.
2. Create at least one node of this content type—more if you want to see the random selection at work.

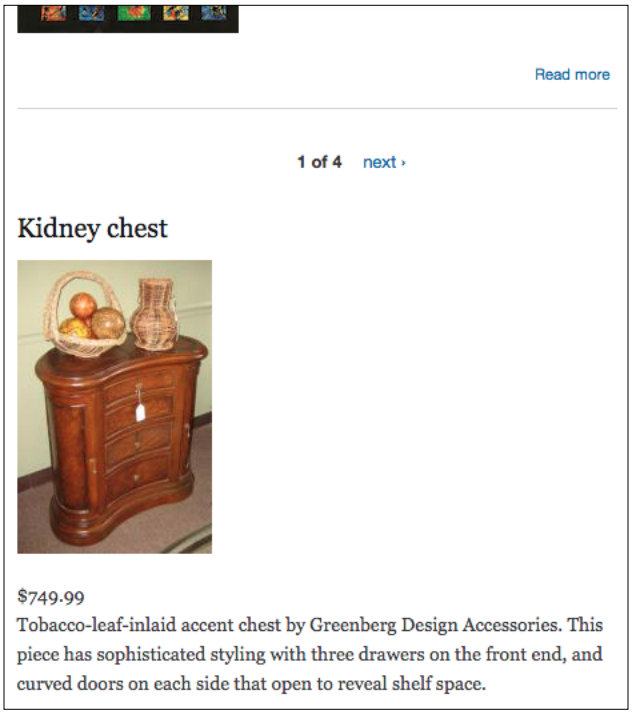
How to do it...

On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`). Click the **+Add new view** link. Enter `Product ads` as the **View name**.
2. Check the **Description** box and enter `Random product ad block`.
3. Select **Content** from the **Show** list and select of type to be **Product**.
4. Uncheck the box for **Create a page**.
5. Check the box for **Create a block**.
6. In the **Display format** select boxes, select **unformatted list**, **of teasers**, **without links**, and **without comments**, respectively, set the **Items per page** to **1**, and click the **Continue & edit** button.

That creates the view, and now we just need to add a few settings:

1. Click the **Content: Post date (desc)** link in the **Sort criteria** section, and click the **Remove** button.
2. Click the **Add** button in the **Sort criteria** section, scroll down and check the box for **Global: Random**, and click the **Apply (all displays)** button.
3. Click the **Apply (all displays)** button.
4. Click the **None** link for block name in the **Block** settings section, enter `Random ad` in the textbox and click **Apply**.
5. Click the **Save** button.
6. Navigate to the **Blocks** admin page (`admin/structure/block`).
7. Scroll down to **Random ad** in the **Disabled** section, select **Content** from the select box, and click the **Save blocks** button.
8. Navigate to the front page to view the block.



How it works...

We created a view and limited its selection to product content. We set the sort option to random, which causes Drupal to select a random record from those available. We set the record quantity to 1, so that only one product is shown. We specified that the format be a teaser, that way we see the title, description, price, and image. At that point, if we had created a page display the view would have been displayed on its own page. Instead, because we created a block display and assigned the block to the content area, it shows below the main content.

There's more...

In *Chapter 5*, *Theming Views*, we will cover theming a view. The current format of this view cries out for theming, such as making the price larger and floating the text next to the image.

Using a view content filter

Often, it can be overwhelming to navigate a large quantity of content, browsing through it page by page. Filtering allows the selection of content to be reduced using specific criteria. We will create a view that allows the user to filter content.

Getting ready

1. This recipe uses a custom content type, **Home**, the details of which are in *Appendix B*, *Bundles*.
2. Create at least two nodes of this content type, each with a different zip code.

How to do it...

On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+Add new view** link.
3. Enter `Homes for sale` as the **View name**.
4. Check the **Description** box and enter `Homes for sale`.
5. Select **Content** from the **Show** list and select **of type** to be **Home**.
6. In the **Display format** select boxes in the **Create a page** section, select **Unformatted list**, **teasers**, **with links**, and **without comments**, respectively, then click the **Continue & edit** button.

Having created the view, we need to add a few settings:

1. Click the **Add** button in the **Filters** section.
2. Scroll down and check the checkbox next to **Content: Zip code (field_zip_code)**, then click the **Apply (all displays)** button.
3. In the configuration window for **Content: Zip code (field_zip_code)**, click the **Expose this filter to visitors** to allow them to change its button, ensure that the checkbox for **Required** is unchecked, change **Label** to **Zip code**, check the **Expose** operator checkbox, then click the **Apply (all displays)** button.
4. Click the **Save** button.
5. Navigate to **homes-for-sale**.

   We can select **Is not empty (Not NULL)** from the **Zip code** filter select box, to specify that we want any content with a zip code, but there is currently a bug in that filter choice and selecting it causes the filter to disappear except for the **Apply** button. Instead, we will select **Length is longer than** and enter `4` into the textbox, signifying we want any zip code of at least 5 characters, and click the **Apply** button. We could also specify a zip code using another filter, such as **Is equal to**, to further limit the display.

Homes for sale

Zip code

Length is longer than

Apply

4

M999999

On 640 acres, with 43 rooms in 11,000 square feet. No indoor plumbing or central air.

Zip code: 229021234
35 000 000.00

Read more

WH888888

132 rooms including 35 bathrooms. 55,000 square feet on 18 acres. Several previous residents. Guard booth and gate. Large kitchen.

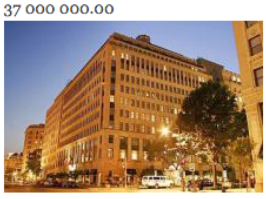
Zip code: 20006
17 760 000.00

Read more

DH777777

273 rooms plus kitchens, maintenance, reception rooms and storage areas. All the modern conveniences.

Zip code: 20006
37 000 000.00



Read more

How it works...

A view is essentially primarily SQL statements, in terms of the data acquisition. In effect, a filter results in an equivalent `WHERE` clause being added to the SQL statement to gather a subset of the available data. So, what we've done is create a view that gathers all content. We reduced the number of nodes the view will receive from the database by specifying that they must be of the content type **Homes**. Finally, we provided the means to select a subset of those records by exposing a filter allowing the user to specify a zip code. By unlocking the operator in the filter, the user is free to request, for example, a specific zip code, zip codes greater than a specified value, all zip codes not equal to another, or every zip code.

Providing a user view for admins

Not every view needs to contain data visible to the general public. The user permissions functionality of views doesn't automatically provide the granularity to allow only certain views to be seen. There is a way to do it, though, and it is quite simple. We'll create a view that shows user data, and limit access to it to admins.

How to do it...

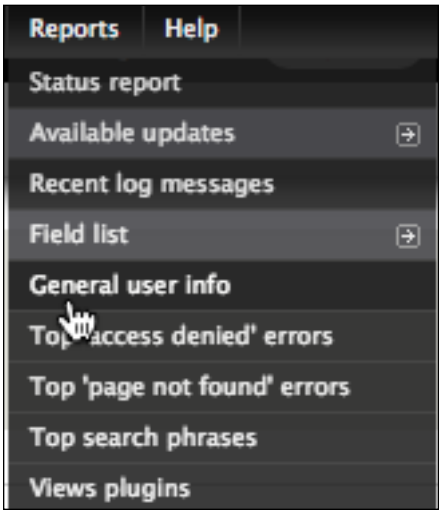
On the views list page:

1. Navigate to the views list (`admin/structure/views`). Click the **+Add new view** link.
2. Enter `Users` as the **View name**.
3. Check the **Description** box and enter `User info`.
4. Select **Users** from the **Show** select box.
5. Enter `General user info` for the page title in the **Create a page** section, `admin/reports/general-user-info` as the **Path**, and in the **Display format** select boxes, select **Unformatted list** and **Fields**, then click the **Continue & edit** button.

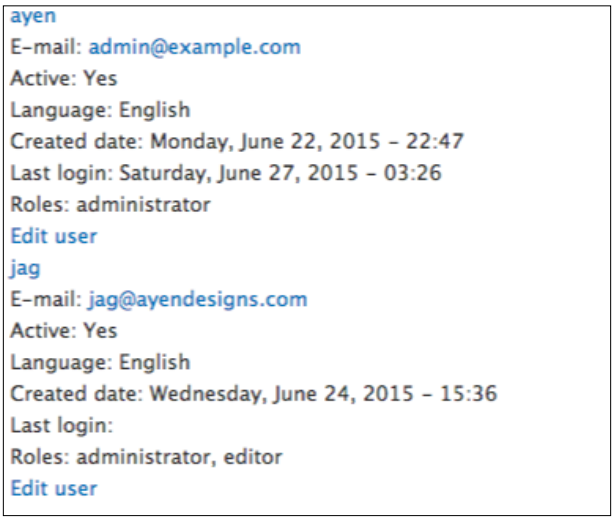
We've created the view framework and will now configure its settings to select the required information.

1. Check the **No menu** link for **Menu** in the **Page Settings** section, select **Normal menu entry** as the **Type**, enter `General user info` in the **Title** textbox, select **Management** from the **Menu** select box, then click the **Apply** button.
2. Click the **Permission** link next to **Access** in the same section, select **Role**, and click the **Apply (all displays)** button.
3. Check the checkbox for administrator, then click the **Apply (this display)** button.

4. Click the **Add** button in the **Fields** box, check the checkboxes for **User: Active**, **User: Created date**, **User: E-mail**, **User: Edit link**, **User: Language**, **User: Last login**, and **User: Roles**, then click the **Apply (all displays)** button.
5. Click the **Apply (all displays)** button for the **User: Active**, **User: Created date**, and **User: E-mail** fields.
6. For the **User: Edit link** field, uncheck the **Create a label** checkbox, enter `Edit user` in the **Text to display** textbox and click the **Apply (all displays)** button.
7. In the **User: Language** window, uncheck the **Link this field to its user** checkbox and click the **Apply (all displays)** button.
8. For the **User: Last login** field and the **User: Roles** field, click the **Apply (all displays)** button.
9. Click the down arrow next to the **Add** button in the **Fields** section and select the **Rearrange** icon to the left, drag **User: E-mail** under **User: Name**, **User: Created date** to directly above **User: Last login**, **User: Edit link** to the bottom, and click the **Apply (all displays)** button.
10. Click the link for **User: Created date (desc)** in the **Sort criteria** section and then the **Remove** button.
11. Click the **Add** button in the **Sort criteria** box, check the checkbox for **User: Name**, and click the **Apply (all displays)** button and then again for the subsequent window.
12. Click **Not set** next to **Access:** in the **Page settings** box, click the **Role** radio button, click the **Administrator** radio button, and click the **Update** button.
13. Click the **Save** button.
14. Navigate to the new report by selecting it from the admin menu, in which case it will display as an overlay.



15. Or you can navigate to it by entering the URL (`admin/reports/general-user-info`).



16. Try logging out and accessing it via the URL.

How it works...

We started by creating a user view, rather than a content view as we have been doing. We did not set a filter, because in the case of a user view there are not content types to be concerned with and we wanted to retrieve all user records. We selected several fields from the user entity and put them in a sensible order. We gave the view a path and assigned it to the admin **Reports** menu. Finally, we specified that access to the view be limited to users with the administrator role. Three of the fields will be links without us having to do anything special to make them so: the edit link, the user name (a link to the user record), and the e-mail address (a mailto: link).

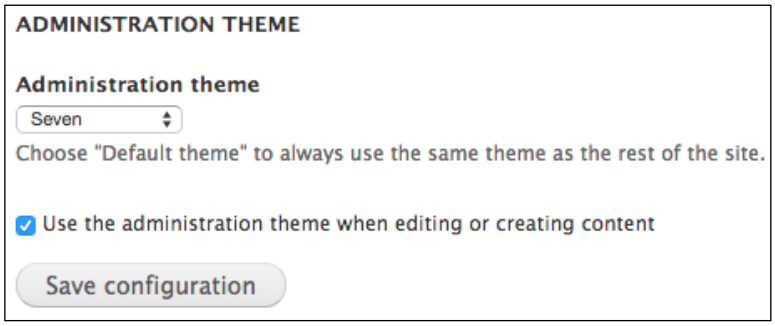
There's more...

There is another method available for restricting access, and that is to select **Permission** instead of **Role** as the access setting. That choice provides a select box that lists existing permissions from which to choose which permission the user needs in order to access the view. There is, however, no *add new permission* choice offered, so the usefulness depends on whether an existing permission already exists. Instead of selecting the administrator role, we could have selected the **Access all Views** permission. Usually, this is restricted to admins, but not necessarily, so the downside of using that would be that if that permission were ever given to users other than admins the view would no longer be restricted to admins. Having selected a role in the view means that administration of access to the view is done in the view rather than through the permission system.

The output leaves much to be desired, aesthetically. In fact, with there being no spacing between user records, the very utility of the display is lessened. We will be learning about using a table format in the next chapter. Using that here would be an immediate improvement on this default output. In *Chapter 5*, *Theming Views*, we will explore the *theming* of views, which is yet another way to improve the aesthetics.

For now, let's make a couple of quick edits to give the view more utility. The site theme already has a CSS file, and we could make the changes directly to it, but if we do that, the next time the theme has an update our changes will be overwritten. Instead, we'll add a CSS file.

The theme that applies to our report is the **Administration theme**, because reports are part of the admin interface. That theme is selected at the lower left of the page at `admin/appearance`:



I'm using the Seven theme which comes with D7:

1. Edit `seven.info` in the `themes/seven` directory, and find the following line:

   ```
   stylesheets[screen][] = style.css
   ```

2. Replace it with this line:

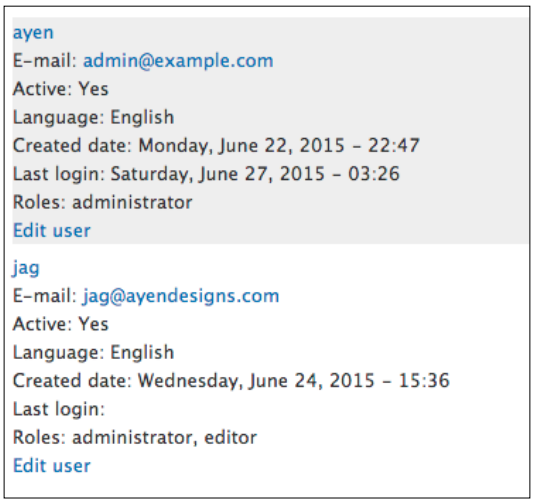
   ```
   stylesheets[screen][] = overrides.css
   ```

3. Open a new file, `overrides.css`, and enter the following into it:

   ```
   .page-admin-reports-general-user-info .views-row {
       margin-bottom: 6px;
   }
   .page-admin-reports-general-user-info .views-row-odd {
       background-color: #eee;
   }
   ```

4. Clear the cache (`admin/config/development/performance`, or `drush cc all` if you have Drush), return to the report, and reload the page.

Now the report is much easier to read:



Winning the argument

The node ID (`nid`), the value that is unique to each piece of content in combination with a URL, provides a direct path to that content, but what if we want a query where the path represents a selection that is not a specific node? The answer: dynamic filters (arguments). We will create a view that can interpret links containing a dynamic filter.

Getting ready

1. This recipe uses a custom content type, **Home**, the details of which are in *Appendix B*, *Bundles.*
2. Create at least two nodes of this content type, each having a different zip code.

How to do it...

On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+Add new view** link.
3. Enter `Homes` links as the **View name**.
4. Check the **Description** box and enter `Homes for sale`.
5. Select **Content** from the **Show** select box, **Home** from the **of type** select box, and set **sorted by** to **Title**.
6. Enter `zip/%` as the **Path**.
7. In the **Display format** select boxes, select **Unformatted list**, **teasers**, **without links** and **without comments**, respectively, then click the **Continue & edit** button.

We will define the general criteria for selecting records:

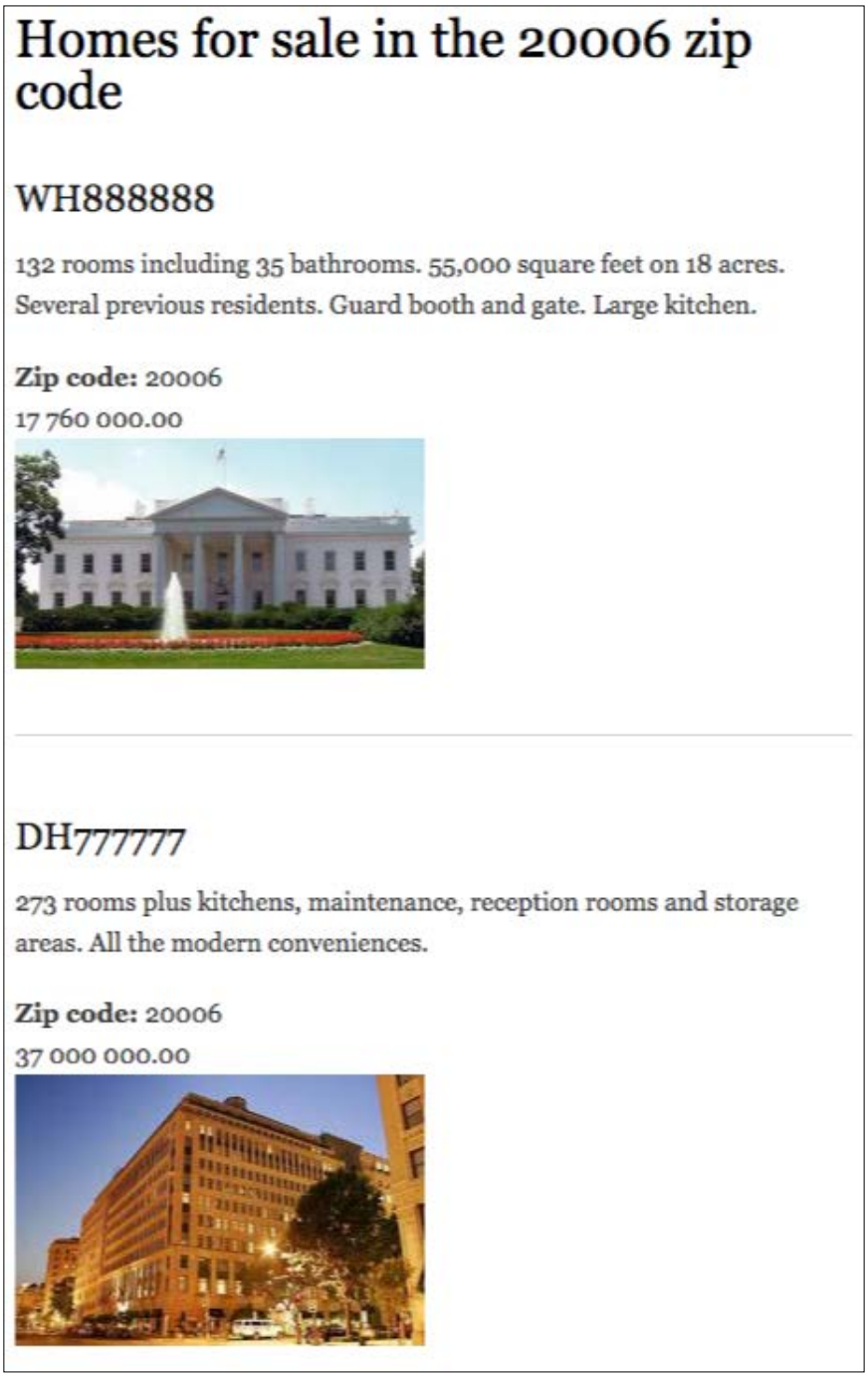
1. Click the **Advanced** link to expand that section.
2. Click the **Add** button in the **Contextual Filters** box, check the checkbox next to **Content: Zip code (field_zip_code)** (note: there are two fields with similar names), and click the **Apply (all displays)** button.
3. In the resulting configuration box, in the section titled `WHEN THE FILTER IS IN THE URL OR A DEFAULT IS PROVIDED`, check the checkbox for **Override title** and enter `Homes for sale in the %1 zip code` in the textbox that appears below it, then click the **Apply (all displays)** button.

We will create the block from which the user will select the filter value:

1. Click the **+ Add** button next to **Page** in the **Displays** list.
2. Select **Block** from the list that drops down.
3. Click **Home links** in the **Title** section, select **This block (override)** in the **For** select box, enter `Zip codes with homes for sale:` in the textbox, and click the **Apply (this display)** button.
4. Click the **Content** link next to **Show** in the **Format** section, select **This block (override)** from the **For** select box, click the radio button for **Fields**, and click the **Apply (this display)** button.

5. Click the **Apply (this display)** button in the subsequent window.
6. Click the **Content: Title** link in the **Fields** section, select **This block (override)** from the **For** select box, and click the **Remove** button.
7. Click the **Add** button in the **Fields** section, ensure **This block (override)** is selected in the **For** select box, scroll down and check the checkbox for **Content: Zip code**, and click the **Apply (this display)** button.
8. Uncheck the **Create a label** checkbox, click the **Rewrite Results** link to expand that section, check the checkbox for **Output this field as a link**, type `zip/[field_zip_code]` into the link path textbox, then click the **Apply (this display)** button.
9. Click the **Content: Title (desc)** link in the **Sort criteria** section, select **This block (override)** from the **For** select box, and click the **Remove** button.
10. Click the **Add** button in the **Sort criteria** section, scroll down and check the checkbox for **Content: Zip code (field_zip_code)**, and click the **Apply (this display)** button.
11. Click the **None** link next to **Block name** in the **Block Settings** section, type `Zip codes` in the textbox, and click the **Apply** button.
12. Click the **Content: Zip code** link in the **Contextual Filters** section, select **This block (override)** from the **For** select box, and click the **Remove** button.
13. Click the **Settings** link next to **Query settings** in the **Other** section, select **This box (override)** from the **For** select box, check the **Distinct** checkbox, then click the **Apply (this display)** button.
14. Click the **Save** button.
15. Navigate to `admin/structure/block`, at the bottom of the **Disabled** list click the select box in the **Zip codes** row, select **Sidebar second**, and click the **Save blocks** button.
16. Navigate to the home page and you will find the block with the zip code list in the right rail:

17. Click one of the zip codes:



How it works...

We created a basic content view, but instead of controlling the content selection with an exposed filter through which a user provides the filtering value, we used a dynamic filter that allows the URL to provide the filter argument and created a block that presents zip codes that link to our view with the selected zip code as an argument. The end result is a URI value becoming a `WHERE` clause in the SQL that retrieves the content, but with this method the selection can come from links selected by the user.

There is a deficiency in the **Views** module's ability to limit content selection to distinct values... it doesn't always work. If you have more than one piece of content with the same zip code, you might see it appear more than once in the selection list. This is purely a cosmetic issue, as the links resolve to the same URL.

There's more...

The decision to use a dynamic filter or an exposed filter would be based on the intention. If it is to allow the user to define a zip code, we would instead have used an exposed filter. The argument capabilities provided by views is quite sophisticated, with the ability to use multiple arguments, default arguments, various argument validation methods, and more. Look for arguments to appear in subsequent recipes.

Using views to create a bulleted list

There are a number of ways in which content can be presented with views. Sometimes, the need is quite simple, as is the case in this recipe. We will create a view that produces a block with a basic bulleted list.

Getting ready

This recipe requires only that there is published content available.

How to do it...

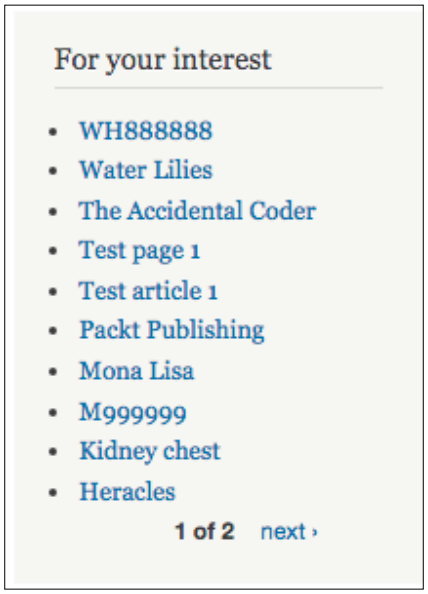
On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+Add new view** link.
3. Enter `Content topics` as the **View name**.
4. Check the **Description** box and enter `Bulleted list of topics`.
5. Set the **sorted by** select box to **Title**.
6. Uncheck the box for **Create a page**.

7. Check the box for **Create a block**.
8. Enter `For your interest` as the **Block title**, and in the **Display format** select boxes, select **HTML list** and **titles** respectively, then click the **Continue & edit** button.

We have created the basic framework of our new view, and now need to establish the settings:

1. Click the **Content: Title (desc)** link in the **Sort criteria** section and then the **Remove** button.
2. Click the **Add** button in the **Sort criteria** section, scroll down and check the checkbox next to **Global: Random**, and click the **Apply (all displays)** button.
3. Click the **Apply (all displays)** button again.
4. Click the **None** link next to **Block name** in the **Block Settings** section, enter **Contents bullet** list into the textbox, and click the **Apply** button.
5. Click the **Display a specified number of items** link in the **Pager** section, select **Paged output**, **mini-pager**, and click the **Apply (all displays)** button.
6. Enter `10` into the **Items to display** textbox, `1` into the **Pager ID** textbox, and click the **Apply (all displays)** button.
7. Click the **Save** button.
8. Navigate to the **Blocks** admin page (`admin/structure/block`), scroll down to the **Disabled** section and set **Content bullets list** to **Sidebar first**, and click the **Save blocks** button.
9. Navigate to the home page to view the block.



How it works...

We created a very simple block display in a content view, which selects all published content of any type. We set the sort criteria to be random, so that any published content might appear on the first "page" of the block. We selected one field to be shown, the title, which becomes a link to the content, itself. We also set the number of items displayed to be 10, with a mini-pager present for paging. We set the ID of the pager to 1, because all pagers on a page must have a unique ID, and since this is a block, there's a good chance that ID 0 will be used by content elsewhere on the page. The essential step for this recipe was setting the output format to that of an HTML unordered list.

There's more...

You might not want the block to contain items already present on the front page, particularly if you set the block to only appear on the front page (accomplished by configuring the block and setting it to only appear on page `<front>`). In that case, an additional filter can be set for **Content: Promoted** to **front page** and set to only select records that are not promoted to the front page, or alternatively, by setting the offset in the pager to one greater than the number of items that appear on the front page.

Our list appears as a standard bulleted list because that is the default formatting for an unordered list as defined in the active theme. By changing the definition of the `<ul>` (unordered list) and `<li>` (list item) tags in the theme CSS file, the bullets can be changed to another type, to images, removed altogether, or the items displayed inline rather than on separate lines.

Creating bulleted lists using multiple content types

There are a number of ways in which content can be presented with views. Sometimes, the needs are quite simple, as is the case in this recipe. We will create a view that produces a block with a basic bulleted list, similar to the previous recipe. However, in this case we will not have simply one long list, but will create a list grouped under headings.

Getting ready

This recipe requires only that there is published content of more than one content type available.

How to do it...

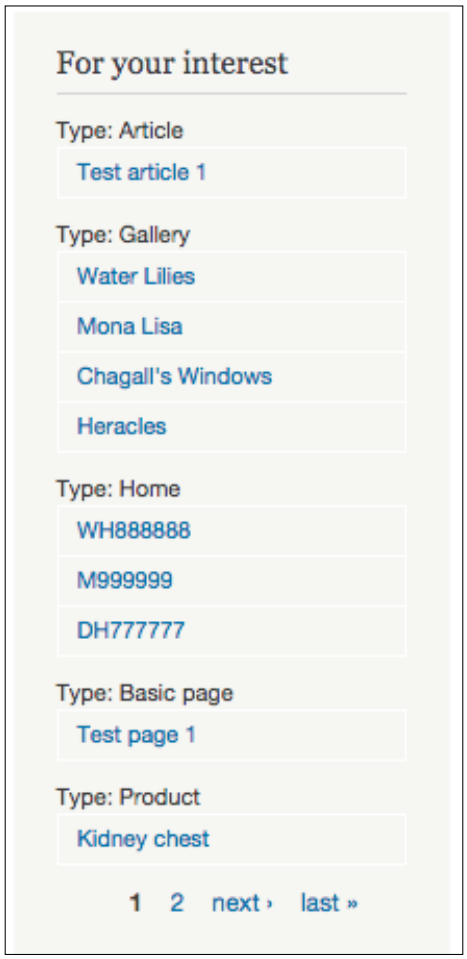
On the **Views** list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+Add new view** link.
3. Enter `Grouped content topics` as the **View name**.
4. Check the **Description** box and enter `Bulleted list of topics`.
5. Uncheck the checkbox for **Create a page**.
6. Check the box for **Create a block**.
7. Enter `For your interest` as the **Block title**, and in the **Display format** select boxes, select **HTML list** and **titles**, enter `10` as the **Items per page**, check the **Use a pager** checkbox, then click the **Continue & edit** button.

We have created the basic framework of our new view, and now need to establish the settings:

1. Click the **Add** button in the in the **Fields** section and check the checkbox for **Content: Type**, then click the **Apply (all displays)** button.
2. In the **Content: Type** configuration window, check the checkbox to **Exclude from display**, and click the **Apply (all displays)** button.
3. Click the **Content: Post date (desc)** link in the **Sort criteria** section and the **Remove** button.
4. Click the **Add** button in the **Sort criteria** section, scroll down and check the checkboxes for **Content: Type** and **Global: Random**, click the **Apply (all displays)** button, and then again in the subsequent configuration window for each field.
5. Click the **HTML list** link in the **Format** section, change the formatting to **Table** and click **Apply (all displays)**. Set **Grouping field Nr.1** to **Type** and click **Apply (all displays)**.
6. Click **None** next to **Block name** in the **Block Settings** section, enter `Grouped contents bullet list` in the textbox, and click the **Apply** button.
7. Click the **Advanced** link to expand that section, click the **No link** next to **Use aggregation**, check the **Aggregate** checkbox, and click the **Apply (all displays)** button.
8. Click the **Save** button.
9. Navigate to the **Blocks** admin page (`admin/structure/block`), scroll down to the **Disabled** section and set the select box for **Grouped content bullets list** to **Sidebar first**, then click the **Save blocks** button.

10. Navigate to the home page to view the block:



How it works...

In this recipe, we essentially created the same view that we did in the previous recipe, except for a few changes. We specified that the primary sort be the content type. We also added **Content: Type** as a field. We set it to be excluded from the display, so it would not be listed in the table as row data, but would be available for use as a heading. The most important change was specifying that the table be aggregated by content type, which in turn segregated the output records.

3
Intermediate Custom Views

In this chapter, we will cover:

- Selecting content teasers based on type and contents
- Displaying a table of entity fields
- Sortable table with a header and footer
- Changing pages the AJAX way
- Understanding relationships
- Grouping in a query
- Content within content
- Producing custom links
- Proving a negative with a filter and an argument

Introduction

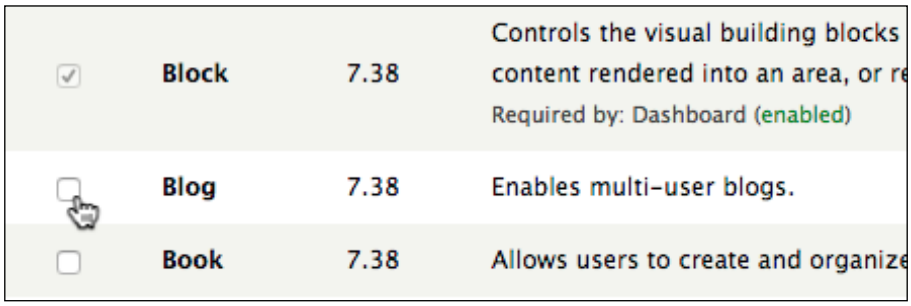
In this chapter, we will continue creating custom views. The examples will be somewhat more complex than in the previous chapter.

Selecting content teasers based on type and contents

Sometimes, simply filtering by content type is not sufficient. Filters can be combined to provide more than one selection criteria. We can select based on metadata, such as content type, but also based on the content itself. In this recipe, we are going to select entries that are blog posts having the same topic, as determined by the title or the category terms.

Getting ready

1. If it's not already enabled, enable the core **Blog** module at `admin/modules`:



2. Edit the blog content type (created by the blog module) at `admin/structure/types/manage/blog`:
    1. Click the **MANAGE FIELDS** tab.
    2. In the **Add existing field** section, select **Term reference: field_tags** as the **Field to share**, ensure **Autocomplete term widget** is the form element, click the **Save** button, and then the **Save settings** button.
3. Add a few blog posts using a small number of tags, and at least one other piece of non-blog content with one of the tag terms in the content title.

For this example, we'll be using the tag **food**. Replace **food** in the instructions with whichever tag you use in creating your content. Make sure that you add the term to the taxonomy vocabulary.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and Click the **+Add new view** link.
2. Enter `Blog posts` as the **View name**, check the box for **Description**, and enter `Subsets of blog posts` into the textbox.
3. The **Show** options should be **Content**, **Blog entry**, and **Newest first**.
4. In the **Create a page** section, change the **Page title** to `Food blog posts` and the **Path** from `blog-posts` to `blog-posts/food`.
5. In the **Display format** select boxes, change **with links** to **without links**.
6. Click the **Continue & edit** button.

At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

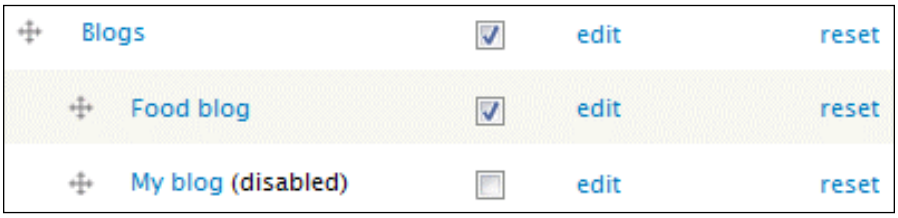
1. Click the **Page** link next to **Display name** at the top of the page, change **Page** to **Food**, and click the **Apply** button.
2. Click the **Add** button beside **Filter Criteria**, scroll down and check the checkbox next to **Content: Tags (field_tags)** and **Content: Title**, and click the **Apply (all displays)** button.
3. In the **Configure extra settings for filter criterion Content: Tags**, select **Autocomplete** and click the **Apply and continue** button.
4. Check the checkbox for **Reduce duplicates**, enter `food` in the **Select terms** textbox and click **Apply (all displays)** button.
5. In the **Configure filter criterion: Content: Title** configuration window, change the **Operator** select box to **Contains**, enter `food` in the **Value** textbox, and click the **Apply (all displays)** button.

All filters are being evaluated as *AND* conditions, meaning that all have to be true for a record to be selected. We want the filters for content type and for content title to be *OR* conditions, meaning if either is true the record will be selected. To do this, we must reorganize the filters.

6. Click the arrow next to the **FILTER CRITERIA Add** button and select **and/or**.

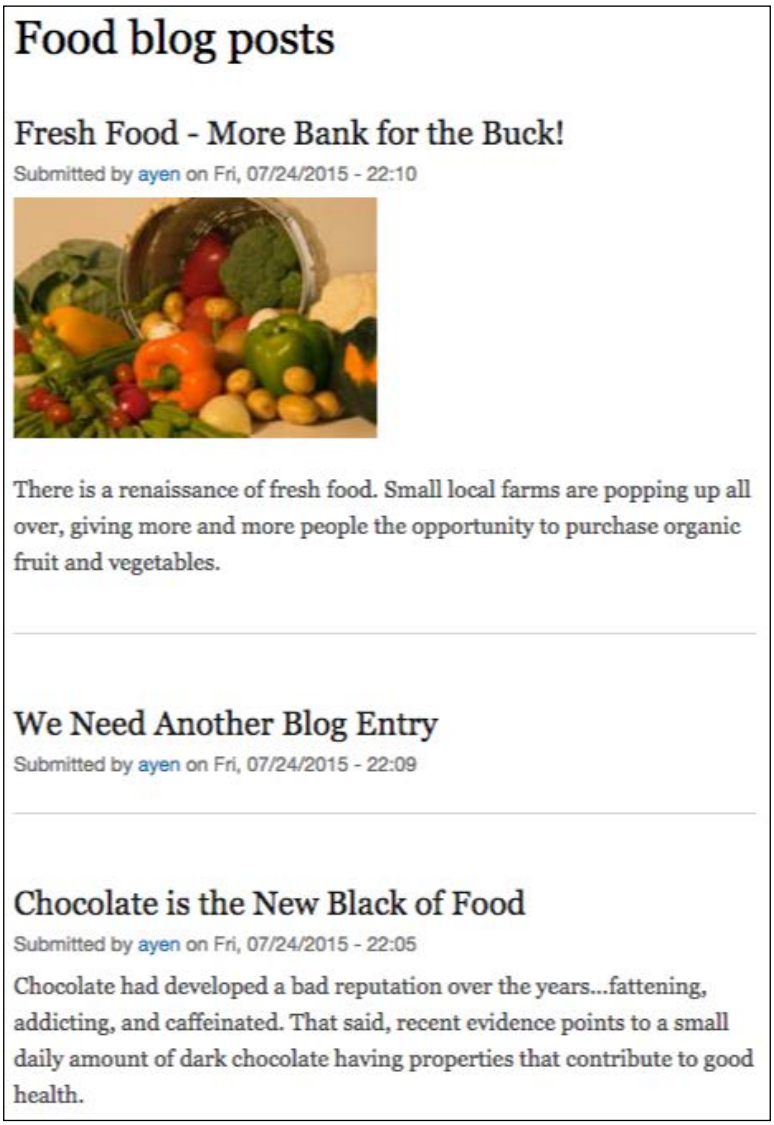
7. Click the **Create new filter group** link, drag the **Content: Tags** and **Content: Title** filters down to the lower group area, then change the **Operator: And** setting to their left to **Or**, then click the **Apply (all displays)** button. The **Filter Criteria** section should now look like this:

8. Click **No menu** next to **Menu** in the **Page settings** section, click the radio button next to **Normal menu entry**, enter `Food blog` in the **Title** textbox, and click the **Apply** button.
9. Click the **Save** button at the top of the page.
10. Across from the **Displays** list, click the down-arrow next to **edit view name/ description** and select **analyze view**.
11. The results box should indicate that there is nothing to report. Click the **Ok** button.
12. Navigate to the menu editor (`admin/structure/menu`) and click the **list links** link for the **Navigation** menu.
13. Uncheck the checkbox for the **My blog** link (further down the list), drag **Food blog** up to under **Blogs** and make sure it's indented the same as the **My blog** entry, enable **Blogs** (the **disabled** status will not change until saved) and click the **Save configuration** button, and that portion of the menu list should now look like the following screenshot:



14. Click the **edit** link for **Blogs**, check the checkbox for **Show as expanded**, and click the **Save** button.

15. Navigate to the front page and click the **Food blog** menu link:



How it works...

By default, a content view will select all available content with the filtering being that the content is published. It will sort starting from the most recently published.

We changed our view to also filter the content selection based on the type being blog, and also to filter on the content being tagged with the term food, as well as the title containing the word `food`. This filtering would normally result only in blog content tagged with the term `food` and a title containing the word `food` being selected. However, we gave the filters an *or* relationship to each other, so the selection became blog content tagged with the term `food` OR content with `food` in the title. This results in the article about fresh food being included because the word `food` is present in its title, as well as content without the word `food` present at all, because it was tagged with the term `food`.

We checked the checkbox for **Reduce duplicates** for the **Tags** field, because the current version of the **Views** module at the time of writing has a bug that causes only one of the *OR* conditions to be evaluated, due to the type of SQL join being incorrect. Checking that checkbox forces the type of join used to be the correct one.

We assigned a path to our page display of `mydomain.com/blog/food` and added it to the site's navigation menu. We clicked the checkbox for **Show as expanded** so that the submenu choice would be visible.

There's more...

The main concept to take away from this view is that with an *OR* relationship, *any* of the criteria can be true, as opposed to an *AND* relationship, in which *all* criteria need to be true. We will see *OR* filters again later in the book.

Displaying a table of entity fields

It is common for the output of a view to be in the form of an article or a teaser. That isn't the only format in which a view can present content, however. Sometimes, a user is able to intuit the data more easily with a more recognizable format, such as a table, and that is what we will create in this recipe.

Getting ready

1. This recipe uses a custom content type, **Ingredient**, the details of which are in *Appendix B*, *Bundles*.
2. Create a few pieces of content of this type.

How to do it...

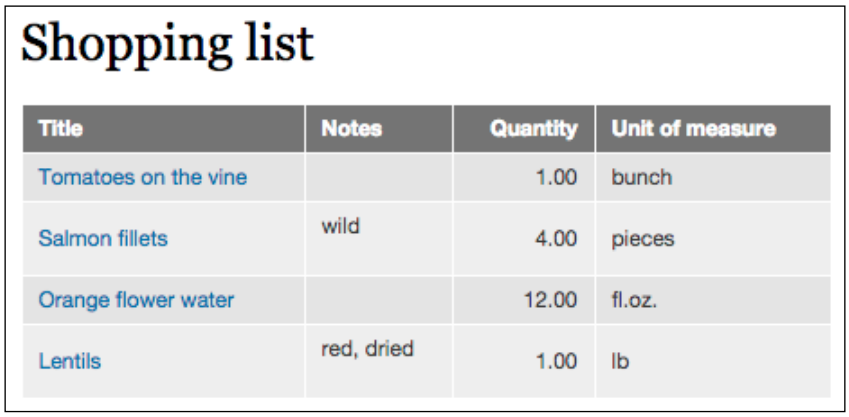
On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Shopping list` as the **View name**, check the box for **Description**, and enter `A list of bulk ingredients` in the textbox.
3. The **Show** option select boxes should be **Content**, **Ingredient**, and **Title**.
4. For **Display format** in the **Create a page** section, select **Table** (the adjacent select boxes will disappear).
5. Click the **Continue & edit** button.

At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Add** button next to **Fields**, check the boxes for **Content: Body**, **Content: Measure**, and **Content: Quantity**, and click the **Apply (all displays)** button.
2. In the **Configure field: Content: Body** window, change the label from **Body** to **Notes**, then click the **Apply (all displays)** button.
3. In the **Configure field: Content: Measure** window, change the label to `Unit of measure` and click the **Apply (all displays)** button.
4. In the **Configure field: Content: Quantity** box, just click the **Apply (all displays)** button without making any changes.
5. Click the arrow next to the **Add** button in the **Fields** section and select **Rearrange**; in the resulting window, drag **Content: Quantity** to above **Content: Measure**, and click the **Apply (all displays)** button.
6. Click the **Settings** link next to **Table** in the **Format** section, select **Right** for the **Align** setting for **Quantity**, and click the **Apply (all displays)** button.
7. Click the **Save** button at the top of the page.
8. Across from the **Displays** list click the arrow next to **edit view name and description** and select **analyze view**.
9. The results box should indicate that there is nothing to report. Click the **Ok** button.

10. Navigate to `/shopping-list` to see the view:



Shopping list

Title	Notes	Quantity	Unit of measure
Tomatoes on the vine		1.00	bunch
Salmon fillets	wild	4.00	pieces
Orange flower water		12.00	fl.oz.
Lentils	red, dried	1.00	lb

How it works...

The table format style exposes the content fields as an HTML table. The table format option cannot be saved unless at least one field has been selected for the view, but the **Title** field is selected by default.

There's more...

We took the default options for the table style. One tremendous benefit of using this styling is that when editing the style options for the table, each of the field columns has a checkbox that, when checked, makes that column sortable by clicking its title. For example, if you have a price field in your data, and you set that checkbox sortable, the price column will be a clickable link that when clicked, will sort the table rows by price. We'll do this in the next recipe.

Sortable table with a header and footer

It might seem like the only text in a view display can be the title, the fields, and their headings, if any, but there is no such limitation. Two forms of text that can be added to views are a header and footer. We will create a view that uses both.

Getting ready

1. This recipe uses a custom content type, **Product**, the details of which are in *Appendix B*, *Bundles*.
2. Create some content of this type.

How to do it...

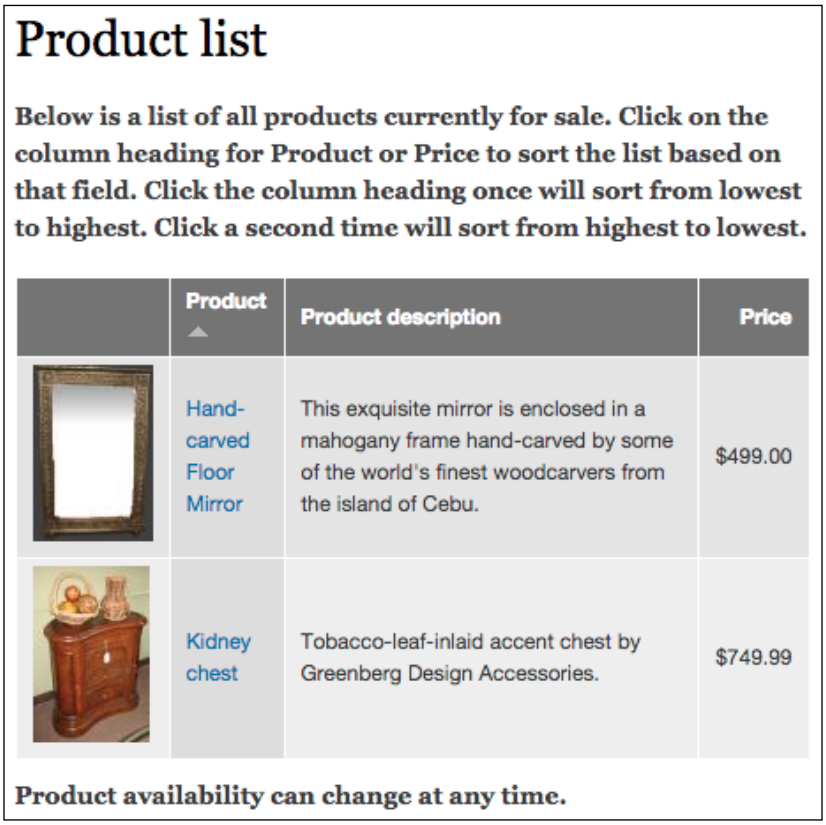
On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Product list` as the **View name**, check the checkbox for **Description**, and enter `A list of products for sale` in the textbox.
3. The **Show** options should be **Content**, **Product**, and **Title**.
4. In the first select box of **Display format** in the **Create a page** section, select **Table** (the adjacent select boxes will disappear).
5. Click the **Continue & edit** button.

At this point we have defined the overall view, and now we can adjust its settings:

1. Click the **Add** button next to **Fields**, check the boxes for **Content: Body**, **Content: Product image,** and **Content: Product price**, then click the **Apply (all displays)** button.
2. In the **Configure field Content: Body** window, change the label to `Product description`, select **Summary or Trimmed** in the **Formatter** select box, change **600** to `200` in **Trim length**, click the **Rewrite Results** link and check the checkbox for **Strip HTML tags**, then click the **Apply (all displays)** button.
3. In the **Configure field Content: Product image** window, uncheck the **Create a label** checkbox, select **Thumbnail (100x100)** from the **Image style** select box, then click the **Apply (all displays)** button.
4. In the **Configure field Content: Product price** window, change the label to `Price`, then click the **Apply (all displays)** button.
5. Click the link for **Content: Title** in the **Fields** section, enter `Product` as the label, then click the **Apply (all displays)** button.
6. Click the arrow next to the fields **Add** button and select **Rearrange**; in the resulting window drag **Content: Product** image to above **Content: Title** and click the **Apply (all displays)** button.
7. Click the **Settings** link next to **Table** in the **Format** section, check the **Sortable** checkbox for **Product** and for **Price**, click the radio button to make **Product** the default sort item, select **Right** for the **Align** setting for **Price**, and click the **Apply (all displays)** button.
8. Click the **Add** button for **Header**, check the checkbox for **Global: Text area**, then click the **Apply (all displays)** button.

9. In the large textbox, enter `<strong>Below is a list of all products currently for sale. Click on the column heading for Product or Price to sort the list based on that field. Click the column heading once will sort from lowest to highest. Click a second time will sort from highest to lowest. </strong>` as the header text and click the **Apply (all displays)** button.
10. Click the **Add** button for **Footer**, check the checkbox for **Global: Text area**, then click the **Apply (all displays)** button.
11. Enter `<strong>Product availability can change at any time.</strong>` as the footer text and click the **Apply (all displays)** button.
12. Click the **Save** button at the top of the page.
13. Across from the **Displays** list, click the arrow next to `edit view name and description` and select **analyze view**.
14. The results box should indicate that there is nothing to report. Click the **Ok** button.
15. Navigate to `/product-list` to see the view:



Product list

Below is a list of all products currently for sale. Click on the column heading for Product or Price to sort the list based on that field. Click the column heading once will sort from lowest to highest. Click a second time will sort from highest to lowest.

	Product	Product description	Price
	Hand-carved Floor Mirror	This exquisite mirror is enclosed in a mahogany frame hand-carved by some of the world's finest woodcarvers from the island of Cebu.	$499.00
	Kidney chest	Tobacco-leaf-inlaid accent chest by Greenberg Design Accessories.	$749.99

Product availability can change at any time.

How it works...

We created a basic field-driven content view, and used a table for the display style. The data in the table includes a product image thumbnail, the product name as a link to the full content, a trimmed version of the product description, and the product price. We reordered the fields to show the image first. We configured the table to sort initially by the product name, and made the product name and price columns selectable as sort keys by clicking the applicable column heading. We added a header with some formatting, and a footer.

Go ahead, click **Price**. You might have to click it a second time to see a difference if the output was already, coincidentally, in price order.

There's more...

The widget for creating a header or footer allows HTML to be entered, and, depending on the permissions granted the view creator, to select from a number of input formats. Thus, there are many possibilities, including having an image present within either. There is an option to show or suppress headers or footers if there is no content to show and the header/footer makes no sense in that context. There is also a way to use a view as the contents of the header or footer for those cases where the text needs to be dynamic.

Changing pages the AJAX way

In the normal scheme of things, changing pages with a pager results in the entire web page being reloaded. This can be an annoyance for the site visitor, particularly if the page loads result in advertising being loaded from remote sites with a resulting delay. Paging through 20 pages of output can be irksome, and more so if the paging is being done in a block rather than the main content area. Fortunately, there is a way to make the experience more enjoyable for the user, and that is by using AJAX for the page changes.

Getting ready

This recipe uses any content type that defines field: image and contains an uploaded image.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Photo gallery` as the **View name**, check the checkbox for **Description**, and enter `Gallery of images transitioned via AJAX` in the textbox.
3. Uncheck the checkbox for **Create a page**.
4. Check the checkbox for **Create a block**.
5. Change the selection in the **Display format 'of'** select box from **titles (linked)** to **fields**.
6. Change **Items per page** to **1**.
7. Click the **Continue & edit** button.

At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Add** button for **Filter Criteria**, check the box for **Content: Image (field_image:fid)** and click the **Apply (all displays)** button.
2. In the subsequent configuration window, change **Operator** to **Is not empty (NOT NULL)** and click the **Apply (all displays)** button.
3. Click the **Content: Title** link in the **Fields** section, and click the **Remove** button.
4. Click the **Add** button for **Fields**, check the checkbox for **Content: Image**, and click the **Apply (all displays)** button.
5. In the subsequent configuration window, uncheck the **Create a label** checkbox, select the **Image style** of **Thumbnail (100x100)**, and click the **Apply (all displays)** button.
6. Click the **Advanced** link to expand that section.
7. Click **No** next to **Use AJAX**, change the setting to **Yes**, then click the **Apply (all displays)** button.
8. Click **None** next to **Block name** in the **Block settings** section, enter `AJAX gallery` in the textbox, and click the **Apply** button.
9. Click **Display a specific number of items** in the **Pager** section, click the radio button for **Page output**, **mini pager**, and click **Apply (all displays)**.
10. In the subsequent configuration window, set the **Pager ID** to **2** and click the **Apply (all displays)** button.
11. Click the **Save** button.

12. Navigate to `admin/structure/block`, scroll down to the **AJAX gallery** entry, select **Sidebar first**, and click the **Save blocks** button.
13. Navigate to the home page to see the view. Using the pager to change the page will result in an AJAX page change, as shown in the following screenshot:

How it works...

We created a content view with a block display that selects any published content in which there is an uploaded image. We did this by specifying that the ID of `field_image` cannot be null. We set the pager to allow one row per page, which means one image per page. The key in this recipe is specifying the use of AJAX, which then uses that service to transition the images in situ without reloading the page with each change.

Understanding relationships

A typical view will select information for each row from one data source, such as a node of content that meets the criteria of whatever filters are in place. How, though, do we handle a situation where the data for each row is *not* all in one place, where some of the data needs to come from another piece of content?

Enter the relationship, the widget in views that allows us to relate one data source to another. We will create a view that lists university courses from one content type, and draws the college department name from another.

Getting ready

1. This recipe uses two custom content types, **Course** and **Department**, the details of which are in *Appendix B*, *Bundles*.
2. Create some content of each type, entering **Department** content first.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Course list` as the **View name**, check the box for **Description**, and enter `A list of courses` in the textbox.
3. The **Show** options should be **Content**, **Course**, and **Title**.
4. In the **Page title** box, enter `Course list`.
5. For the **Path**, enter `course-list`.
6. In **Display format**, select **Table** (the adjacent select boxes will disappear).
7. Click the **Continue & edit** button.

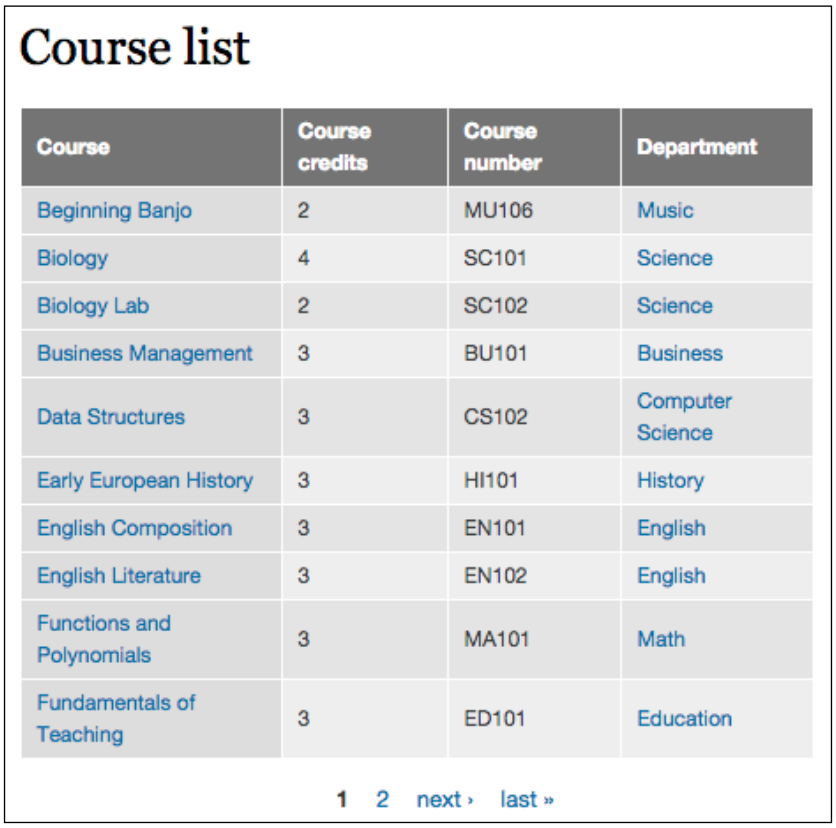
At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Content: Title (Title)** link in the **Fields** section, change the **Label** from **Title** to `Course`, then click the **Apply (all displays)** button.
2. Click the **Add** button next to **Fields**, check the checkboxes for **Content: Course credits** and **Content: Course number**, then click the **Apply (all displays)** button.
3. In the **Configure field Content: Course credits** overlay, click the **Apply (all displays)** button.
4. In the **Configure field Content: Course number** overlay, click the **Apply (all displays)** button.

Now we will create the relationship that links the selected **Course** to a **Department**:

1. Click the **Advanced** link to reveal the advanced settings.
2. Click the **Add** button beside **Relationships**, check the checkbox for **Content: Department (field_department)** and click the **Apply (all displays)** button, then click the **Apply (all displays)** button in the subsequent window.
3. Click the **Add** button next to **Fields**, check the checkbox for **Content: Title**, and click the **Apply (all displays)** button.
4. In the **Configure field Content: Title** window, change the **Relationship** setting to **field_department**, change the label from **Title** to `Department`, and click the **Apply (all displays)** button.
5. Click the **Settings** link in the **Format** section, click the radio button for **Default sort** for **Course**, and then click the **Apply (all displays)** button.
6. Click the **Save** button at the top of the page.

7. Navigate to `/course-list` to see the view:



Course list

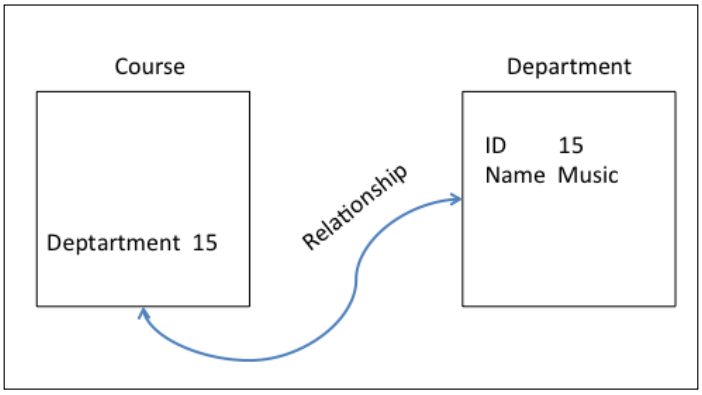
Course	Course credits	Course number	Department
Beginning Banjo	2	MU106	Music
Biology	4	SC101	Science
Biology Lab	2	SC102	Science
Business Management	3	BU101	Business
Data Structures	3	CS102	Computer Science
Early European History	3	HI101	History
English Composition	3	EN101	English
English Literature	3	EN102	English
Functions and Polynomials	3	MA101	Math
Fundamentals of Teaching	3	ED101	Education

1 2 next › last »

How it works...

We created a content view that displays the fields for a specific content type, **Course**. We selected the table style to display the data. We wanted to display the department name along with the course, but the department name is contained in the content type **Department** rather than **Course**. When entering the content, there were radio buttons labeled with the **Department** names from which to select the department, but once selected, only the ID of the **Department** content, a reference or pointer, was stored in the **Course** content, and *not* the department title.

So, we needed a way to access the **Department** content referred to in the **Course** content, and we did that by implementing the relationship between the two within our view. Then, when we chose the **Content: Title** field for the second time, but specified that the second instance use the relationship to **Department**, which resulted in the **Department** title rather than the **Course** title.



There's more...

A relationship, in SQL parlance, is a **join**. Thus, when we establish a relationship between **Course** and **Department** using a node reference (its `nid`) in the course content, the resulting SQL fragment would be:

```
LEFT JOIN department ON department.nid = course.department_ref
```

And the field that we display will be selected as `department.title` instead of `course.title`, because we identified it as using the `field_department` relationship. We are not limited to one relationship, but can, if we need, have several.

Why use another content type?

Why do that instead of simply embedding the department name in the course record? Were we to embed the actual text and the name later changed, we would need to change it in every course record containing it. So, why not define a select field in the course record, and enter the department names in one place from which to choose? That way, it only has to be changed once, in the select's available values. That would be acceptable, but our department content contains the department phone number, chairman's name, and degree programs, and not just the department name... information useful for other views...so all of that information would need to be duplicated in every course record as well.

Grouping in a query

If you are familiar with spreadsheets or reports, then you're also familiar with the concept of grouping: segregating data items by a higher level value, such as students in a class. We will create a view that lists departments and the college courses within them.

Getting ready

1. This recipe uses two custom content types, **Course** and **Department**, the details of which are in *Appendix B*, *Bundles*.
2. Create some content of each type.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Department course list` as the **View name**, check the checkbox for **Description**, and enter `A list of departments and courses` in the textbox.

3. The **Show** options should be **Content**, **Course**, and **Title**.
4. In **Display format**, select **Table** (the adjacent select boxes will disappear).
5. Click the **Continue & edit** button.

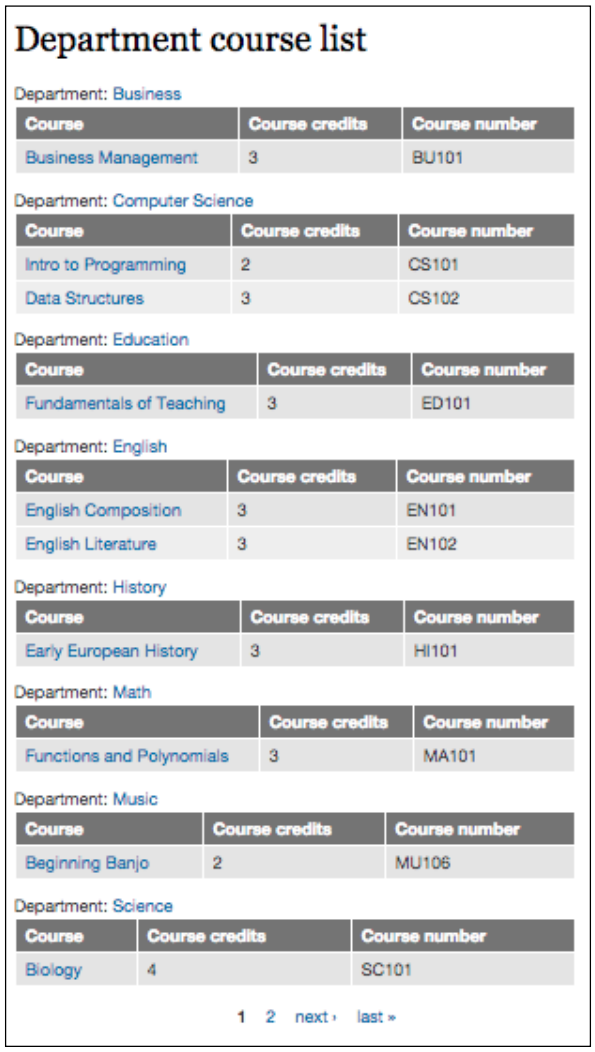
At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Content: Title (Title)** link in the **Fields** section, change the **Label** from **Title** to `Course`, then click the **Apply (all displays)** button.
2. Click the **Add** button next to **Fields**, check the checkboxes for **Content: Course credits** and **Content: Course number**, and click the **Apply (all displays)** button.
3. In the **Configure field Content: Course credits** overlay, click the **Apply (all displays)** button.
4. In the **Configure field Fields: Course number** overlay, click the **Apply (all displays)** button.

Now we will create the relationship that links the selected **Course** node to a **Department** node:

1. Click the **Advanced** link to reveal the advanced settings.
2. Click the **Add** button beside **Relationships**, check the checkbox for **Content: Department (field_department)**, click the **Add (all displays)** button, then click the **Apply (all displays)** button in the subsequent window.
3. Click the **Add** button next to **Fields**, check the box for **Content: Title**, and click the **Apply (all displays)** button.
4. In the **Configure field Content: Title** window, change the **Relationship** setting to **field_department**, change the label from **Title to Department**, check the checkbox to **Exclude from display**, and click the **Apply (all displays)** button.
5. Click the **Settings** link in the **Format** section and click the radio button for **Default sort** for **field_department Department**, select that same field from the **Grouping field Nr 1** select box, then click the **Apply (all displays)** button.
6. Click the **Save** button at the top of the page.

7. Navigate to `/department-course-list` to see the view:



Department course list

Department: Business

Course	Course credits	Course number
Business Management	3	BU101

Department: Computer Science

Course	Course credits	Course number
Intro to Programming	2	CS101
Data Structures	3	CS102

Department: Education

Course	Course credits	Course number
Fundamentals of Teaching	3	ED101

Department: English

Course	Course credits	Course number
English Composition	3	EN101
English Literature	3	EN102

Department: History

Course	Course credits	Course number
Early European History	3	HI101

Department: Math

Course	Course credits	Course number
Functions and Polynomials	3	MA101

Department: Music

Course	Course credits	Course number
Beginning Banjo	2	MU106

Department: Science

Course	Course credits	Course number
Biology	4	SC101

1 2 next › last »

How it works...

We created a content view that displays fields from two content types, **Course** and **Department**. We selected the table style to display the data. We wanted to display department information along with the course, but the department name is contained in another content type, **Department**. The **Course** content type contains a node reference to the applicable **Department** node, but the actual string containing the name of that department is in the **Department** content and not the **Course** content.

So, we needed a way to access the content of the department that is referred to from the course, and we did that by establishing a relationship between the two. That would have simply given us a list of courses, but we wanted the list to be courses within each department, so we elected to *group* the records by **Department**.

Content within content

We can think of catalogs, real estate listings, and the like, as containers of similar content. For example, real estate listings may have some introductory text and other information that appears once within the page and then a series of similarly-formatted home listings.

We can replicate this with a view. A content type can contain unique information, as well as node references and pointers, which point to the home listings. This recipe allows us to create just such a pairing.

Getting ready

1. This recipe uses two custom content types, **Home** and **Real Estate flier**, the details of which are in *Appendix B*, *Bundles*.
2. Create some content of each type.

How to do it...

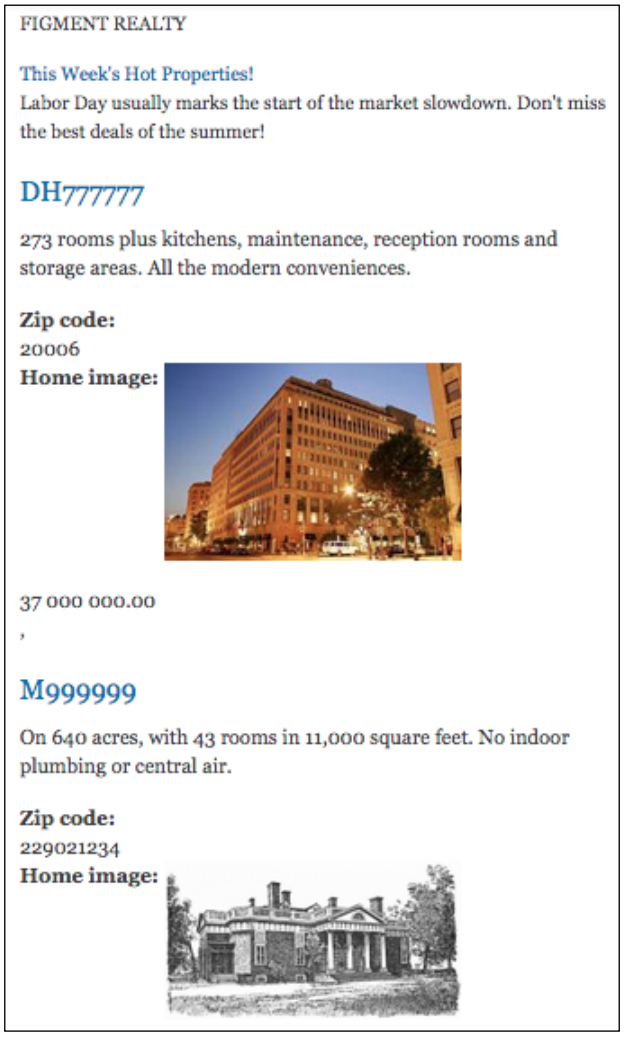
On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Real estate flier` as the **View name**, check the box for **Description**, and enter `Weekly real estate deals` into the textbox.
3. The **Show** options should be **Content**, **Real Estate flier**, and **Newest first**.
4. Set the **teasers** select box to **fields** in **Display format** and click the **Continue & edit** button.

At this point we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Add** button next to **Fields**, check the boxes for **Content: Body** and **Content: Property**, and click the **Apply (all displays)** button.
2. In the **Configure field Content: Body** window, uncheck the **Create a label** checkbox and click the **Apply (all displays)** button.

3. In the **Content: Property** window, uncheck the **Create a label** checkbox, select **Rendered node** from the **Formatter** select box, and click the **Apply (all displays)** button.
4. Click the **Add** button in the **Header** section, check the box for **Global: Text area**, and click the **Apply (all displays)** button.
5. Enter `<h1>FIGMENT REALTY</h1>` in the large textbox and click the **Apply (all displays)** button.
6. Click the **Save** button at the top of the page.
7. Navigate to `/real-estate-flier` to see the view:



How it works...

We created a somewhat standard content view, except that the content type that we selected, **Real Estate flier**, contains a field that, itself, contains multiple content references. This means that this field in the Real Estate flier contains an array of node IDs, each one being the ID of content of the type **Home**. In this way, we created a flier that contains any number of real estate entries, the inclusion of which is based solely on those we specifically want to include, rather than selection criteria that can be expressed in a filter setting.

There's more...

We specified that the entire content body should be displayed for each home. We could have specified that a teaser be shown instead, but then had we wanted to print the flier, each home would have been a text excerpt with a readmore link, which would have little value in print. We could have created two displays, one for print and one for browser, with the former showing the complete node and the latter a teaser.

Producing custom links

It is common to see teaser content with a title and/or a readmore link being a link to the full content body, or an image performing the same function. This is easily accomplished in a view by checking the checkbox for **Link this field to its node** in the field settings. Sometimes, though, we want to provide a link to a *view* instead, or a link to a callback in a custom module, or perhaps link to another site altogether. We will create a view that provides custom links.

Getting ready

1. This recipe uses any of the content types listed in *Appendix B*, *Bundles*.
2. If you have not already done so, create a node for at least some of the content types.

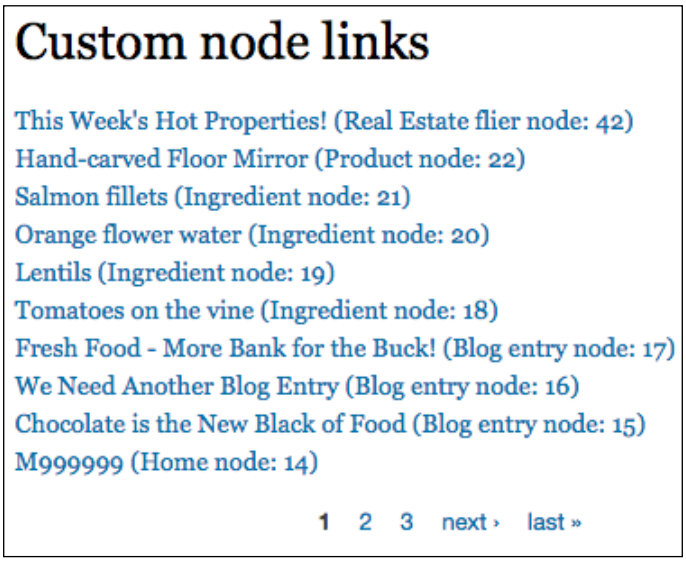
How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Custom node links` as the **View name**, check the box for **Description**, and enter `An index of site content, with each entry being a custom link` in the textbox.
3. Change **Display format** to **Unformatted list** of **Fields** and click the **Continue & edit** button.

At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Add** button next to **Fields**, check the boxes for **Content: Nid** and **Content: Type**, and click the **Apply (all displays)** button.
2. In both subsequent configuration windows, uncheck the **Create a label** checkbox, check the **Exclude from display** checkbox, and click the **Apply (all displays)** button.
3. Click the downarrow next to the **Add** button in the **Fields** section and select **Rearrange**, then drag the **Title** field to the bottom of the list and click the **Apply (all displays)** button.
4. Click the **Content: Title** link in the **Fields** section, uncheck the **Link this field to the original piece of content** checkbox, and click the **Rewrite Results** link.
5. Check the **Rewrite the output of this field** checkbox, and in the **Text** textbox that appears, enter `[title] ([type] node: [nid])`.
6. Check the **Output this field as a link** checkbox, in the **Link path** textbox enter `http://www.example.com/myview/[nid]`, and then click the **Apply (all displays)** button.
7. Click the **Save** button at the top of the page.
8. Navigate to `custom-node-links` to see the view:



How it works...

We created a node view that selects all published content. We chose to include the fields for the node ID (`nid`) and the content type, but specified that their value not be displayed. The reason for this is that we only wanted their values available to be used as replacement tags for another field.

We then reordered the fields, because fields don't appear in the replacement tag list unless they appear in the field list prior to the field to use them. Since we wanted both the content type and content `nid` values available to the content title field, they both had to precede it.

Finally, we rewrote the output value of the title field to include the content type and `nid` in the link text, and changed the link value to point to a view in another (but fake) domain, with `nid` as an argument.

There's more...

In my use of the view, I discovered that because of the order in which I created content, page 1 of the view output was all college courses. To avoid this, I added a filter for content type, and set the filter to exclude any content of the type **Course** or **Department**.

Proving a negative with a filter and an argument

In most every content site, the user will view only published content. There can be unpublished content present, though; content that is in-process, awaiting editing, waiting for its publishing date or season to come around, and so on.

Users with the permission to edit content often get to it by navigating to that content's display and clicking an edit tab. If the content is unpublished, however, there usually is no getting to it from the site's frontend rather than navigating to it in the admin content list.

It would be nice if users with the permissions to create content could access their own unpublished content in an easy manner, and they can! We will build a view that provides the user with a block, which lists their unpublished content, and even allows them to filter dynamically on the content type.

Getting ready

1. This recipe uses nodes of any content type, including those listed in *Appendix B*, *Bundles*.
2. Ensure that there are some unpublished nodes present.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Unpublished content` as the **View name**.
3. Uncheck the **Create a page** checkbox, check the **Create a block** checkbox, and click the **Continue & edit** button.

At this point we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click **None** next to **Block name** in the **Block settings** section, enter `Your unpublished content` in the textbox, and click the **Apply** button.
2. Click **Content: Published (Yes)** in the **Filter criteria** section, change **Yes** to **No**, and click the **Apply (all displays)** button.
3. Click the **Advanced** link to reveal that section.
4. Click the **Add** button for **Relationships**, then check the checkbox for **Content: Author** and click the **Apply (all displays)** button, and the same, again, in the subsequent configuration window.
5. Click **No** next to **Use AJAX**, change the setting to **Yes**, and click the **Apply (all displays)** button.
6. Click the **Add** button in the **Filter criteria** section, check the checkbox next to **Content: Type**, the checkbox next to **User: Current**, and click the **Apply (all displays)** button.
7. In the **Configure filter criterion: Content: Type** configuration window, check the **Expose this filter** checkbox, the **Is one of** radio button under **Operator**, the **Select all** checkbox in **Content types**, the **Allow multiple selections** and **Remember the last selection** checkboxes, and click the **Apply (all displays)** button.
8. In the **Configure filter criterion: User: Current** configuration window, click the **Yes** radio button for **Is the logged in user**, and click the **Apply (all displays)** button.
9. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Edit link**, and click the **Apply (all displays)** button.
10. In the **Configure field: Content: Edit link** settings window, uncheck the checkbox for **Create a label**, click the **Rewrite Results** link, check the **Rewrite the output of this field** checkbox, enter `(Edit)` in the **Text** textarea that appears, check the **Output this field as a link** checkbox, and click the **Apply (all displays)** button.
11. Click the **Settings** link next to **Fields** in the **Format** section, check both the checkboxes in the **Inline fields** section, and click the **Apply (all displays)** button.

12. Click the **Save** button.
13. Navigate to the blocks admin page (`admin/structure/block`), scroll down to **Your unpublished content** in the **Disabled** section, select a region for the block (I'm using **Sidebar second**) from the select box, and click the **Save blocks** button.
14. Navigate to the front page to view the block:

How it works...

We created a content view with a block display, and configured it to select unpublished content. We added the content type as a filter with all values selected, but exposed the filter for the user to make a selection. We also specified that the content must belong to the current user. The default setting for the title is to be a link to its content, and we added an edit link and specified that its title be `edit`. In this way, the user can elect to click the title and view the content, or click the edit link and edit the content.

The AJAX setting must be set to **Yes**, as we did, for an exposed filter to appear within a block.

4
Creating Advanced Views

In this chapter, we will cover:

- A view with multiple personalities
- A marketing bundle
- Filtering with OR
- Page, block, and attachment
- Teaming two content lists
- Related content—adding depth to a term ID
- Related content–adding depth to a term
- Limited visibility

Introduction

In this chapter, we will explore the capability Views has to provide multiple displays, sometimes simultaneously. Right out of the box, Views can create page, block, attachment, and feed displays, and with displays being pluggable, contributed modules can add to that list. This chapter introduces the basics of using multiple displays, with further capabilities theming, presented in *Chapter 5*, *Theming Views*, delves further into this topic, and the concepts apply despite the book being for Drupal 6.

A view with multiple personalities

It might seem that given the challenge of presenting different content types in different ways there are two choices: create a separate view for each or write code in a module or template file. Depending on the need, the latter might be a reasonable option, but if all that is required is minor differences in content selection, sort order, or the fields to be presented, there is another alternative available.

We're going to create a view, where most of the settings are made in an umbrella manner, and then create a display containing some minor variances for each content type.

Getting ready

- We'll be using the blog entry and home content types
- Enable the blog module if you have not done so
- Details for the home content type are in *Appendix B*, *Bundles*
- Create some entries for each content type

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Chameleon` as **View name**, check the box for **Description**, and enter `View title and Content type displays` in the textbox.
3. The select box settings for **Show**, **of type**, and **sorted by** should be **Content**, **All**, and **Title**.
4. In the **Create a page** section, change the **Page title** from **Chameleon** to `Blog entries`, and the **Path** to `content/blog-entries`.
5. In the **Display format** field, change **Teasers** to **Titles (linked)**—the adjacent select boxes will disappear.
6. Click the **Continue & edit** button.

At this point, we have defined most of the information for one display, and now we can continue on the view edit page:

1. Click the **Page** link next to **Display name** and change **Name** to `Blogs`, then click the **Apply** button.
2. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Post date**, and click the **Apply (all displays)** button.

3. In the **Configure field: Content: Post date** window, change the label from **Post date** to `Posted`, then click the **Apply (all displays)** button.
4. Click the **Settings** link for **Show** in the **Format** section, check the box for each field listed beneath **Inline fields**, and click the **Apply (all displays)** button.
5. Click the **Add** button in the **Sort Criteria** section, check the box for **Content: Post date**, and click the **Apply (all displays)** button.
6. Select **Sort descending** and click the **Apply (all displays)** button.
7. Click the downarrow next to the **Sort Criteria Add** button and select **Rearrange**, in the resulting window drag the **Content: Title** field to below **Content: Post date** and click the **Apply (all displays)** button.
8. Click the **Add** button in the **Filter criteria** section, check the checkbox for **Content: Type**, and click the **Apply (all displays)** button.
9. Check the box for **Blog entry** and click the **Apply (all displays)** button.

That completes the blog display, but this view will have more than one display. Now we will create the homes display:

1. Click the **+Add** button next to the **Blogs** button in the **Displays** section at the top of the page and select **Page**.
2. Click the **Page** link next to **Display name** and change the name to `Homes`, then click the **Apply** button.
3. Click **Blog entries** next to **Title**, change **All displays** to **This page (override)**, enter `Homes` in the textbox, and click the **Apply (this display)** button.
4. Click **No path is set** next to **Path** in the **Page settings** box, enter `content/homes` as the path, and click the Apply button.
5. Click the **Content type (= Blog entry)** link in the **Filters section**, change **All displays to This page (override)**, uncheck the checkbox for **Blog entry**, check the box for **Home**, and click the **Apply (this display)** button.
6. Click the **Content: Post date (Posted)** link in the **Fields** section, change the select box to **This page (override)**, and click the **Remove** button.
7. Click the **Add** button in the **Fields** section, check the checkboxes for **Content: Home image** and **Content: House price**, and click the **Apply (this display)** button.
8. Uncheck the **Create a label** checkbox, select **Medium (220x22)** in the **Image style** select box and **Content** in the **Link image to** select box, and click the **Apply (this display)** button.
9. Uncheck the checkbox for **Create a label**, change the **Thousand marker** to **Comma**, the **Scale** from **2** to **0**, and click the **Apply (this display)** button.
10. Click the **Add** button in the **Sort criteria** section, select **This page (override)** in the select box, check the box next to **Content: House price**, and click the **Apply (this display)** button, and then again in the subsequent window.

11. Click on the sort field links for **Content: Post date**, ensure that **This page (override)** is selected, and click the **Remove** button. Do the same for **Content: Title**.
12. Click the **Save** button at the top of the page.
13. Navigate to the `/content/blog-entries` link to see the following image:

14. Navigate to the `/content/homes` link to see the following image:

How it works...

We created a content view and two page displays within it, one each for the **Blog entry** and **Home** content types. Each display, so each content type, has its own filters, sort criteria, field selection, and title.

There's more...

There is more than one way to accomplish the results we see here. We could have created a template for each content type, and one view display that selects records based on an argument. Templates are discussed in the next chapter. The decision should be based on whether there are likely to be changes to the view later on, and who is going to be maintaining it; if a developer or themer, then perhaps a template is the way to go, but if an editor with little or no programming experience, then the method we used would be better.

A variation on this would be to create each display as a block instead of a page, and to assign each to the content area on the front page. If doing so, and if using a pager for each, remember that every pager on a page must have a unique ID number, so be sure to change the pager ID setting for each.

A marketing bundle

Each view's display type has its own utility, and can be used to create a set of related displays to fulfill a purpose. In this recipe, we will create a marketing bundle using a page display for a sales presence landing page, a block display for an ad, and a feed with which customers can stay updated about new content.

Getting ready

- We'll be using the **Product** content type (details are in *Appendix B*, *Bundles*)
- At least one piece of product content will be needed

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Marketing bundle` as the view name, check the box for **Description**, and enter `Landing page, ad and feed` in the textbox.
3. The **Show** options should be **Content**, **Product**, and **Title**.

4. In the **Page title** textbox, enter `Greenberg Design Accessories Brings You the Best Bargains!`.
5. Enter `bargains` in the **Path** textbox.
6. For **Display format**, select **Grid** and **fields** (the other two select boxes will disappear).
7. Change the **Items to display** field to **3**.

Next, we will enter the settings to create a block:

1. Check the checkbox for **Create a block**.
2. For the block title, enter `Another Great Greenberg Design Accessories Product`.
3. Change **titles (linked)** to **fields** in **Display format** and change **Items per page** to **1**.
4. Click the **Continue & edit** button.

At this point, we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Settings** link next to **Grid**, select **This page (override)** in the select box, change **Columns** from **4** to **3**, and click the **Apply (this display)** button.
2. Click the **Add button in the Fields** section, select **This page (override)** from the select box, check the checkbox for **Content: Product image**, and click the **Apply (this display)** button.
3. In the **Configure field: Content: Product image** window, select **This page (override)** from the select box, uncheck the **Create a label** checkbox, select **Medium (220x220)** from the **Image style** select box, select **Content** in the **Link image to select** box, then click the **Apply (this display)** button.
4. Click the link for **Content: Title** in the **Fields** section, uncheck the checkbox for **Link this field to the original piece of content**, click on the **Rewrite Results** link, check the checkbox for **Rewrite the output of this field**, enter `<h2>[title]</h2>` in the **Text** box, then click the **Apply (this display)** button.
5. Click the **Add** button in the **Header** section, select **This page (override**) from the select box, click the **Global: Text area** checkbox, and click the **Apply (this display)** button.
6. Enter `<big>The liquidation area of Greenberg Design Accessories is where you can find amazing bargains in leftover and discontinued products.</big>` in the large textbox and click the **Apply (this display)** button.
7. Click the **Add** button in the **Footer** section, select **This page (override)** from the select box, click the **Global: Text area** checkbox, and click the **Apply (this display)** button.
8. Enter `<small>Greenberg Design Accessories is a division of Acme Holding Corp</small>` in the large text box and click the **Apply (this display)** button.

That takes care of the page display; next we will configure the block display:

1. Click the **Block** button in the **Displays** section.
2. Click the link for **Content: Title (desc)** in the **Sort criteria** box, change the select box to **This block (override)**, and click the **Remove** button.
3. Click the **Add** button in the **Sort criteria** box, check the checkbox for **Global: Random**, click the **Apply (this display)** button, and the same again in the subsequent window.
4. Click the **Content: Title** link in the **Fields** section, change the select box setting to **This block (override)**, and click the **Remove** button.
5. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Product image**, and click the **Apply (this display)** button.
6. Uncheck the **Create a label** checkbox, change the image style to **Thumbnail (100x100)**, and click the **Apply (this display)** button.
7. Click the **None** link in the **Block Settings** section, enter `Marketing bundle ad` in the textbox, and click the **Apply** button.

The final step is to create the feed:

1. Click the **+Add** button in the **Displays** section and click the **Feed** link.
2. Click **No path is set** next to **Path** in the **Feed settings** section, enter `greenbergdesignaccessories/feed` as the path, and click the **Apply** button.
3. Click the title in the **Title** box, change the select box setting to **This feed (override)**, enter `The Newest Bargains from Greenberg Design Accessories` in the textbox, and click the **Apply (this display)** button.
4. Click the **Use site default RSS settings** link for **Show** in the **Format** section, select **Title only** from the **Display type** select box, then click the **Apply** button.
5. Click the **Content: Title** link in the **Sort Criteria** section, change the select box setting to **This feed (override)**, and click the **Remove** button.
6. Click the **Add** button in the **Sort Criteria** section, check the checkbox for **Content: Post date**, and click the **Apply (this display)** button.
7. Change the select box setting to **This feed (override)**, choose **Sort descending**, and click the **Apply (this display)** button.
8. Click the **Save** button.
9. Navigate to the block admin page (`admin/structure/block`), find **Marketing bundle ad** in the **Disabled** section and change the select box setting to **Sidebar first**, then click the **Save blocks** button.

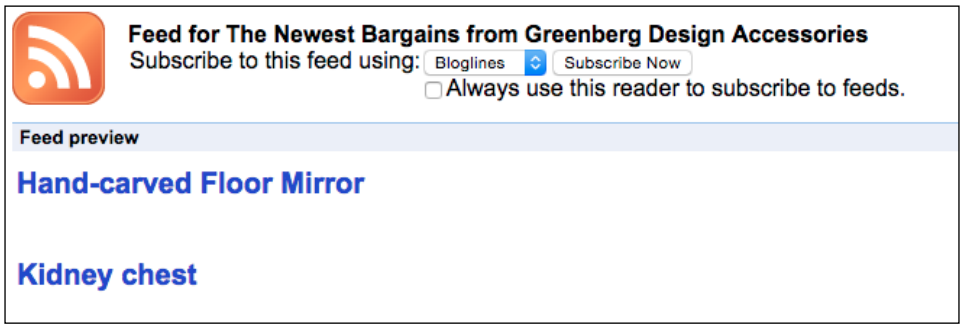
10. Navigate to the home page to see the block shown in the following image:

11. Navigate to the RSS feed (`greenbergdesignaccessories/feed`).
12. If you do not have a RSS feed reader installed in your browser, you will probably see the raw feed XML:

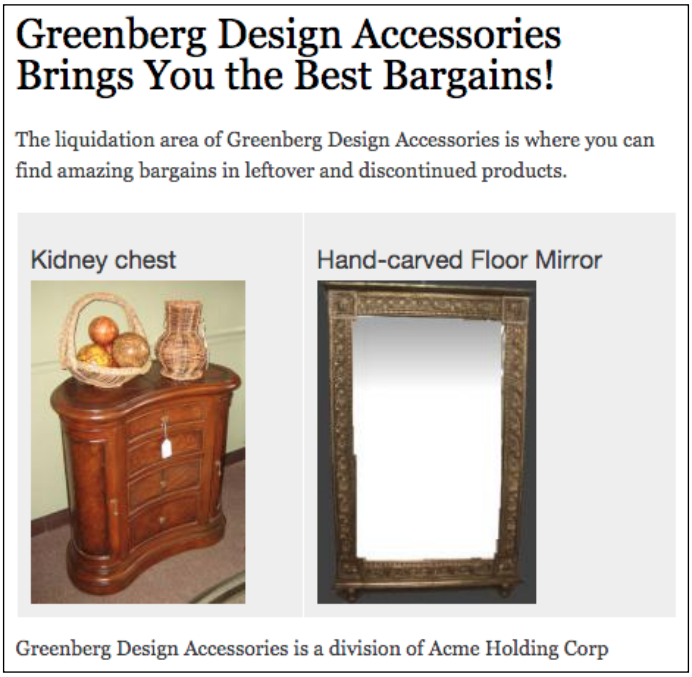
```
<?xml version="1.0" encoding="utf-8" ?><rss version="2.0"
xml:base="http://dev.cookbook/bargains"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:content="http://purl.org/rss/1.0/modules/content/"
xmlns:foaf="http://xmlns.com/foaf/0.1/"
xmlns:og="http://ogp.me/ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:sioc="http://rdfs.org/sioc/ns#"
xmlns:sioct="http://rdfs.org/sioc/types#"
xmlns:skos="http://www.w3.org/2004/02/skos/core#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
   <channel>
   <title>The Newest Bargains from Greenberg Design
   Accessories</title>
   <link>http://dev.cookbook/bargains</link>
   <description></description>
   <language>en</language>
   <item>
     <title>Hand-carved Floor Mirror</title>
     <link>http://dev.cookbook/node/22</link>
     <description></description>
      <pubDate>Sat, 25 Jul 2015 23:36:36 +0000</pubDate>
      <dc:creator>ayen</dc:creator>
      <guid isPermaLink="false">22 at
      http://dev.cookbook</guid>
   </item>
   <item>
     <title>Kidney chest</title>
     <link>http://dev.cookbook/node/11</link>
```

```
        <description></description>
         <pubDate>Sun, 12 Jul 2015 02:49:20 +0000</pubDate>
         <dc:creator>ayen</dc:creator>
         <guid isPermaLink="false">11 at
         http://dev.cookbook</guid>
      </item>
      </channel>
    </rss>
```

13. If you do have a RSS feed reader installed, you will see something similar to the following image:



14. Navigate to the landing page (`/bargains`), which appears as in the following image:



How it works...

We created a node view that selects nodes of the product content type. We then built three displays: a block to use as an advertisement, an RSS feed to allows users to subscribe and keep current with the site, and a landing page that can be used, for example, when the user clicks on an ad at another site. The main idea behind this view is that each display can have its own settings with regards to fields, sort criteria, and even the filter settings, though we used the same filter throughout this example.

Filtering with OR

Filtering is the way to control record selection in a view, whether through contextual filters or... regular filters (there isn't a contrasting term for the latter, but when they are exposed for the user to choose a value on which to filter, they, too, become contextual).

It is possible, and typical, to use two or more filters in conjunction, such as requiring that content be a specific type and be published, or more precisely, that content be of a certain type *AND* be published. When you have *x AND y*, x must be true, and y must be true. If either is false, then entire statement is false and the content will not be selected. So if the content is published but is the wrong type, it will not be selected. Likewise, if it is the correct type but is unpublished, it will not be selected. This is fine, if you intend that to be the requirement, but what if you don't?

Let's use an analogy: buying shoes. Filtering with AND is the equivalent of asking the sales clerk for shoes that are size 8 AND casual AND leather. You insist (albeit nicely) that all three requirements be met. However, you could instead ask for shoes that are size 8 AND are either casual or leather. In other words, they must be your size, but aside from that, they will be acceptable if they are casual, leather, or both.

In this example, the requirement can be written as:

SIZE = 8 AND (STYLE = CASUAL OR MATERIAL = LEATHER)

In prior versions of views, having an *OR* filter was problematic, at best, but Views 3 makes it easy, and in this recipe we will create a view that uses it.

Getting ready

The following will be needed for this recipe:

- One node of the **Ingredient** content type (details are in *Appendix B*, *Bundles*)
- One node of a different content type, with the word `Food` somewhere in the content or title

How to do it...

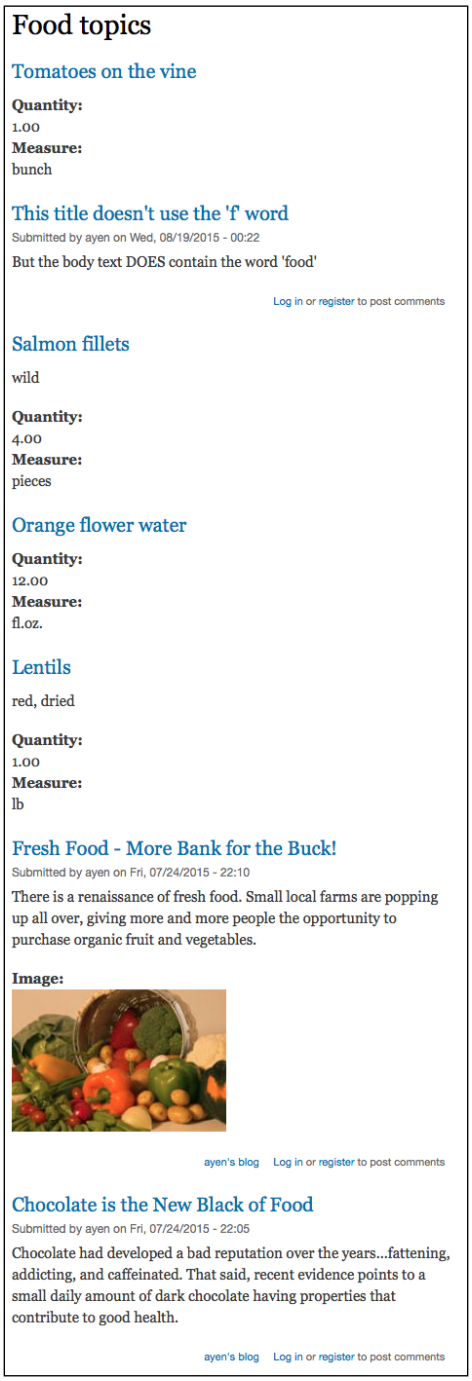
On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Food topics` as the **View name**, check the checkbox for **Description**, and enter `Content that is a recipe OR has 'Food' in the title or body` in the textbox.
3. The **Show** options should be **Show: Content**, of **type: All** and **sorted by: Title**.
4. Check the box for **Include an RSS feed** and change the path from `food-topics.xml` to `food-topics/feed`.
5. Click the **Continue & edit** button.

At this point, we can adjust settings on the view edit page:

1. Click the **Add** button in the **Filter Criteria** section, check the checkboxes for **Content: Body (body)**, **Content: Title**, and **Content: Type**, then click the **Apply (all displays)** button.
2. In the **Content: Body** configuration dialog, change the **Operator** selection to **Contains**, enter `Food` in the **Value** textbox, and click the **Apply (all displays)** button.
3. In the **Content: Title** configuration dialog, change the **Operator** selection to **Contains**, enter `Food` in the **Value** textbox, and click the **Apply (all displays)** button.
4. In the **Content: Type** configuration dialog, check the checkbox for the **Ingredient** content type, then click the **Apply (all displays)** button.
5. Click the down arrow next to the **Add** button in the **Filter Criteria** section and click the **And/Or, Rearrange** link.
6. Click the **+ Create new filter group** link and drag each field with the exception of **Content: Published** to the group at the bottom.
7. Change the lower **Operator** to **Or**, and click the **Apply (all displays)** button.
8. Click the **Save** button at the top of the page.

9. Navigate to the `food-topics` to see the following:



Food topics

Tomatoes on the vine

Quantity:
1.00
Measure:
bunch

This title doesn't use the 'f' word

Submitted by ayen on Wed, 08/19/2015 - 00:22

But the body text DOES contain the word 'food'

Log in or register to post comments

Salmon fillets

wild

Quantity:
4.00
Measure:
pieces

Orange flower water

Quantity:
12.00
Measure:
fl.oz.

Lentils

red, dried

Quantity:
1.00
Measure:
lb

Fresh Food - More Bank for the Buck!

Submitted by ayen on Fri, 07/24/2015 - 22:10

There is a renaissance of fresh food. Small local farms are popping up all over, giving more and more people the opportunity to purchase organic fruit and vegetables.

Image:

ayen's blog Log in or register to post comments

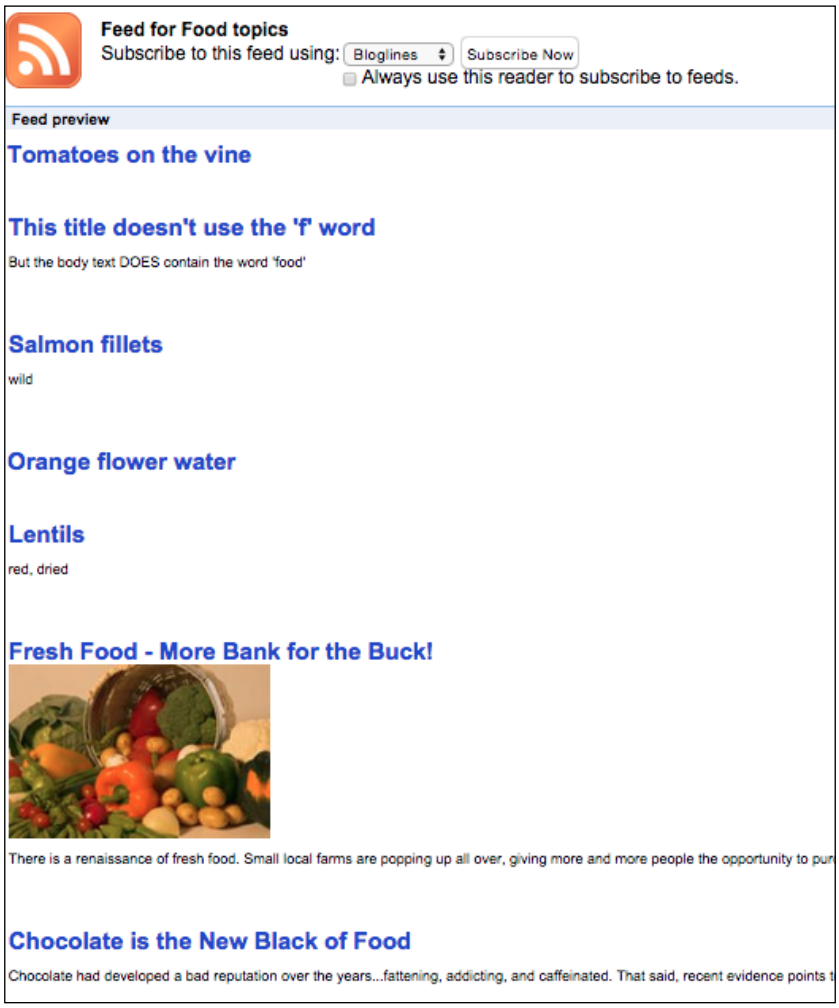
Chocolate is the New Black of Food

Submitted by ayen on Fri, 07/24/2015 - 22:05

Chocolate had developed a bad reputation over the years...fattening, addicting, and caffeinated. That said, recent evidence points to a small daily amount of dark chocolate having properties that contribute to good health.

ayen's blog Log in or register to post comments

10. If you have a RSS feed reader, navigating to the `food-topics/feed` will show the following (otherwise just a dump of the feed content):



How it works...

We accomplished two things in this recipe, the creation of a compound filter, and the creation of a feed. We only changed the path of the feed because of personal preference; the original path would have been fine.

We created the compound filter to accommodate *all/any* logic, where connecting rules with *OR* is the same as stating that *any* of them can be true, while connecting them with *AND* is the same as stating that *all* of them must be true. We put the filters in two groups, with *AND* between them, resulting in the following rule:

Published AND (Ingredient node OR title contains "food" OR body contains"food")

Another way of reading it is:

Content must be published, and one or more of (type=ingredient, body contains "food", title contains "food:) must be true.

Page, block, and attachment

You've probably used desktop applications in which the screen is divided into two or more areas, each working in conjunction with the other. The same functionality is possible on your web page. We will use a page display and an attachment display, arguments, and relationships to create sections that work together, giving more form and function to a view.

Getting ready

- We'll be using the **Employee** and **Extension** content types (details are in *Appendix B, Bundles*)
- Create at least two of each

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Employees and extensions` as the **View name**, check the checkbox for **Description**, and enter `Employee information and extensions` in the textbox.
3. In the **Create a page** section, change **Page title** to `Employee extensions`.
4. Change the entry in the **Path** box to `employees`.
5. For **Display format**, change **teasers** to **fields** (the other two select boxes will disappear).
6. Click the **Continue & edit** button.

At this point let's complete the page, which will display the employee information:

1. Click the **Add** button next to **Fields**, check the checkboxes for **Content: ID** and **Content: Position**, and click the **Apply (all displays)** button.
2. For each of the configure boxes that follow for each of the three fields, simply click the **Apply (all displays)** button.
3. Click the link for **Content: Title** in the **Fields** section, select **This page (override)** from the select box, uncheck the checkbox for **Link this field to the original piece of content**, click on the **Rewrite Results** link to open the dialog, check the checkbox for **Rewrite the output of this field**, enter `<h2>[title]</h2>` in the textbox, then click the **Apply (this display)** button.
4. Click the **Add** button for **Filter Criteria**, check the checkbox for **Content: Type**, and click the **Apply (all displays)** button.
5. Check the checkbox for **Employee** and click the **Apply (this display)** button.
6. Click the **Advanced** link to reveal the advanced criteria.
7. Click the **Add** button for **Contextual Filters**, check the checkbox for **Content: Department (field_department)**, and click the **Apply (all displays)** button.
8. Click the **Apply (all displays)** button.
9. Click the **Add** button for **Relationships**, select **This page (override)** from the **For** select box, check the checkboxes for **Content: Department (field_department)** and **Content: Employee (field_employee) - reverse**, and click the **Apply (this display)** button.
10. Click the **Apply (this display)** button for both configuration windows.
11. Click the **Add** button for **Fields** again, select **This page (override)** from the select box, check the checkbox for **Content: Title**, and click the **Apply (this display)** button.
12. Change the **Relationship** select box setting to **field_employee**, change the **Label** to `Extension`, uncheck the checkbox for linking the field, and click the **Apply (this display)** button.
13. Repeat step 11, select **All displays** from the select box, change the **Relationship** select box setting to **field_department**, change the **Label** to `Department`, then click the **Apply (all displays)** button.

That takes care of the page display, next we will create the block display that will display the departments:

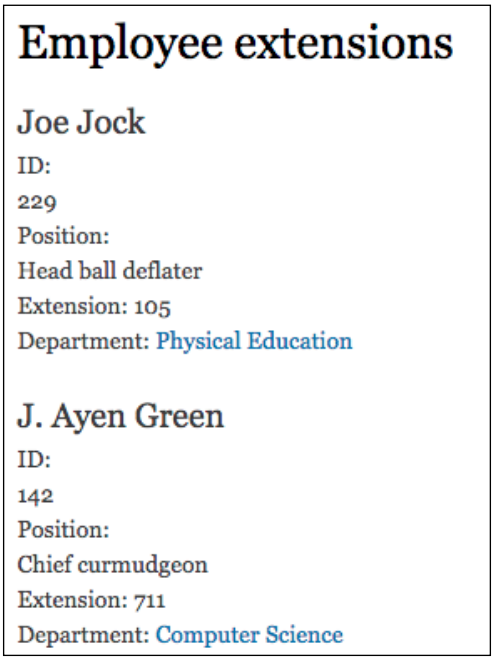
1. Click the **+Add** button in the **Displays** section and select **Block**.
2. Click **Employee extensions** next to **Title**, ensure that **This block (override)** is selected in the **For** select box, enter `Departments` in the textbox, and click the **Apply (this display)** button.

3. Click the link for each field in the **Fields** section, select **This block (override)** from the select box at the top, and click the **Remove** button.
4. Click the **Content: Type (= Employee)** link in the **Filter Criteria** section, select **This block (override)** from the select box, uncheck the **Employee** checkbox, check the **Department** checkbox, and click the **Apply (this display)** button.
5. Click the **Add** button in the **Fields** section, select **This block (override)** from the select box, check the checkboxes for **Content: Nid** and **Content: Title**, and click the **Apply (this display)** button.
6. In the **Content: Nid** configuration window, uncheck the **Create a label** checkbox, check the **Exclude from display** checkbox, and click the **Apply (this display)** button.
7. In the **Content: Title** configuration window, uncheck the **Create a label** checkbox, click the **Rewrite Results** link and check the checkbox for **Output this field as a link**, enter `employees/[nid]` as the **Link path**, and click the **Apply (this display)** button.
8. Click the **None** link for **Block name** in the **Block Settings** section, enter `Department list` in the textbox, and click the **Apply** button.
9. Click the **Content: Department** link in the **Contextual Filters** section, select **This block (override)** from the select box, and click the **Remove** button.

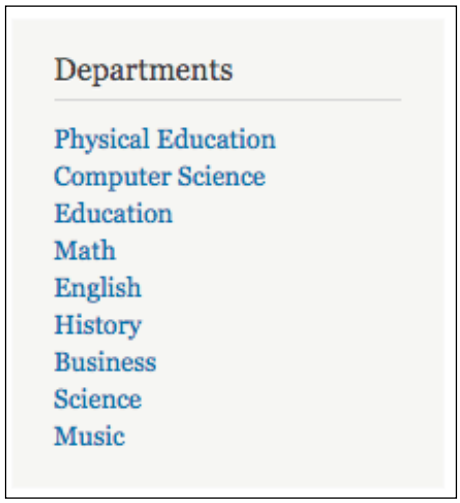
The final step is to create the **Attachment** display that lists employees:

1. Click the **+Add** button in the **Displays** section and click the **Attachment** link.
2. Click the **Content: Department** link in the **Fields** section, select **This attachment (override)** from the select box, and click the **Remove** button.
3. Click the **Content: Position** link in the **Fields** section, select **This attachment (override)** from the select box, and click the **Remove** button.
4. Click the **Not defined** link for **Attach to** in the **Attachment Settings** box, select **Page** and click the **Apply** button.
5. Click the **Before** link for **Attachment position** in the **Attachment Settings** box, change the setting to **After**, and click the **Apply** button.
6. Click the link for **Content: ID (ID)** in the **Fields** section, select **This attachment (override)** from the select box, uncheck the checkbox for **Create a label**, check the box for **Exclude from display** and click the **Apply (this display)** button.
7. Click the downarrow next to the **Add** button in the **Fields** section, click on **Rearrange** and then select **This attachment (override)** from the select box, drag the **Content: ID** field to be above the **Content: Title** field, and click the **Apply (this display)** button.

8. Click the **Content: Title** link in the **Fields** section, click the **Rewrite Results** link and check the checkbox for **Output this field as a link**, enter `employee/all/[field_employee_id]` as the **Link path**, and click the **Apply (this display)** button.
9. Click the **Add** button for **Sort Criteria**, select **This attachment (override)** from the select box, check the checkbox for **Content: Title**, and click the **Apply (this display)** button.
10. Click the **Apply (this display)** button.
11. Click the link for **Content: Post date** in the **Sort Criteria** section, ensure **This attachment (override)** is selected, and click the **Remove** button.
12. Click the **Content: Department** link in the **Contextual Filters** section, select **This attachment (override)** from the select box, and click the **Remove** button.
13. Click the **Save** button.
14. Navigate to the block admin page (`admin/structure/block`), find **Department list** in the **Disabled** section, change the select box setting to **Sidebar first**, then click the **Save blocks** button.
15. Scroll up to the **Sidebar first** section and click the **configure** link for **Department list**; in the section for **Show block on specific pages**, select **Only the listed pages** and enter `employee*` in the textbox; then click the **Save block** button.
16. Navigate to `/employees` to see the components we created: the page, which displays employee information for the current department (*all* by default):



The block, which displays a list of departments:



And the attachment, the following employee information, which displays a list of all employees:

How it works...

You could very well be wondering about the application of this recipe until you give it a closer look. If you hover your mouse over a link in the **Department** list, you will see that it links to the `employees` path, passing the node ID (`nid`) of the department. Clicking a department name will load the `employees` page and show only employees in that department. The employee list, below the employee information, lists all employees.

There are a few things going on in this view that are worth mentioning. The page display determines which employee's information to list based on the argument passed in the URL: the department of the employee(s) to list. If the argument is not passed, such as when the URL was typed into the browser, all employees are listed.

We set the configuration of the department title in the block to be a link back to the employee page that supplies the appropriate department `nid` as an argument.

The employee names that are listed in the attachment below the list of employee information provide links to the employee records themselves, as opposed to the employee page, that is, to information about the employee without regard to a specific department.

The department name in the employees page provides a link to the department content for that department.

The other item of note is the employee extension. It is actually a separate content type, and so a separate table in the database. We retrieved the extension by establishing a relationship between the employee table and the extension table, which in turn creates a table join in the query.

There's more...

There are further things that could be done to improve this view. It certainly needs formatting, particularly to separate the attachment display from the page display, and in this case view templates would probably be the ideal choice. Also, it is likely that this view should be restricted, so consideration should be given to which role(s) should be able to view its content. Lastly, a navigation entry should be added to reach the page initially. This can be done with the **Menu** setting in the **Page Setting** section when viewing the page display.

Teaming two content lists

Sometimes, you want to split one collection of content between two different displays. There is a trick to doing it without the same content appearing in both. In this recipe, we will create a top ten list, but have the top three appear in one display and the remainder appear in the other.

Getting ready

We'll be using the **Country** content type (details are in *Appendix B*, *Bundles*, as is sample data you can use).

How to do it...

Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.

We will first enter the settings to create a page:

1. Enter `Top 10` as the **View name**, check the box for **Description**, and enter `Top 10 Country list` into the textbox.
2. Change the **of type** select box to **Country**.
3. In the **Create a page** section, change the **Display format** setting from **teasers** to **fields** (the other two select boxes will disappear).

4. Click the link for **Content: Title** in the **Fields** section, uncheck the **Link this field to the original piece of content** checkbox, select **This page (override)**, click the link for **Style Settings**, check the checkbox for **Customize field HTML**, select **Strong** from the **HTML element** select box, and click the **Apply (this display)** button.
5. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Area**, and click the **Apply (all displays)** button.
6. Uncheck the **Create a label** checkbox, select **Comma** from the **Thousand marker** select box, select **0** from the **Scale** select box, and click the **Apply (all displays)** button.
7. Click the **Settings** link for **Show** in the **Format** section, check the **Content: Title** and **Content: Area** checkboxes, and click the **Apply (all displays)** button.
8. Click the link for **Content: Post date (desc)** under **Sort criteria** and then the **Remove** button.
9. Click the **Add** button in the **Sort criteria** section, check the checkbox for **Content: area**, and click the **Apply (all displays)** button.
10. Select **Sort descending** and click the **Apply (all displays)** button.
11. Click the link for **Full** in the **Pager** section, select **This page (override)** from the select box, change the setting to **Display a specified number of items**, and click the **Apply (this display)** button.
12. Change the number in **Items per page** to **3** and click the **Apply (this display)** button.

That takes care of the **Page** display: next we will create the **Attachment** display:

1. Check the **+Add** button in the **Displays** section and select **Attachment**.
2. Click the **10 items** link for **Items to display** in the **Pager** section, select **This attachment (override)** from the select box, change **Items to display** to **7** and **Offset** to **3**, then click the **Apply (this display)** button.
3. Click the **Not defined** link for **Attach to** in the **Attachment** section, check the checkbox for **Page**, and click the **Apply** button.
4. Click the **Before** link for **Attachment position** in the **Attachment settings** section and change it to **After**, then click the **Apply** button.
5. Click the **Save** button.
6. Navigate to `/top-10`:

How it works...

One key to this recipe is the ability to specify an offset. For example, suppose you have content that is artwork and you have a place on the home page where you always display the most recent piece. On an interior page, you want to display other pieces, but not the one that is showcased on the home page. In that case, the selection on the interior page would be given an offset of 1, specifying that from the selection results, you want to skip 1 record. Had we not specified an offset of 3, the first three records of the lower display would have been the same as those in the upper display.

For this use of offset to work properly, each display must select the same content and sort it in the same manner.

We chose additional HTML wrapping for the title field for the upper display. The reason we made the appearance of the text for the top 3 different than that of the others was to provide a visual cue. Additionally, CSS would be used to provide further demarcation between the two displays. We also set the formatting to make the two fields inline, so that they displayed side by side.

Related content – adding depth to a term ID

Give the users of your content a useful site and they will come back. One thing that makes a content site more useful is providing the user with a list, one that relates to the content. In this recipe, we will create a display that shows a piece of content, and an attachment display that presents a list of links for related content.

Getting ready

- We'll be using the article content type
- Create the taxonomy terms for this recipe, as shown in *Appendix B*, *Bundles*
- Create a small article for each of the terms

How to do it...

On the view list page, navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.

We will first enter the settings to create a page:

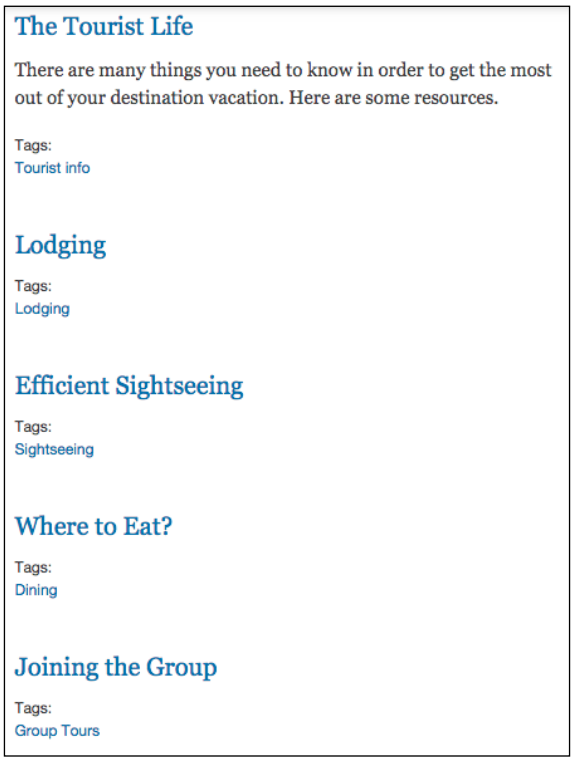
1. Enter `Related content` as the **View name**, check the checkbox for **Description**, and enter `Use term id and depth to provide related content` into the textbox.
2. Change the **Show** select box **of type** to **Article**.
3. In the **Create a page** section, change the **Display format** select box **of** from **Teasers** to **Full posts**, **Items to display** to `1`, and click the **Continue & edit** button.

On the view edit page:

1. Click the **Related content** link next to **Title**, clear the text box, and click the **Apply (all displays)** button.
2. Click the link for **Post date** under **Sort Criteria** and then the **Remove** button.
3. Click the **Advanced** link to reveal the advanced settings.
4. Click the **Add** button for **Contextual Filters**, select **This page (override)** from the select box, check the checkbox for **Content: Nid**, and click the **Apply (this display)** button.
5. Click the **Apply (this display)** button.

That takes care of the page display; next we will create the **Attachment** display:

1. Click **+Add** button in the **Displays** section and select **Attachment**.
2. Click the **Add** button for **Contextual filters**, select **This attachment (override)** from the select box, check the checkboxes for **Content: Has taxonomy term ID (with depth)** and **Global: Null**, then click the **Add and configure contextual filters** button.
3. In the configure box for **Taxonomy: Term ID (with depth)**, select a depth of **2** and click the **Apply** button.
4. In the configure box for **Global: Null**, select **This attachment (override)** from the select box and click the **Apply (this display)** button.
5. Click the **Not defined** link for **Attach to** in the **Attachment settings** section, check the checkbox for **Page**, then click the **Apply** button.
6. Click **Before** next to **Attachment position** in the **Attachment settings** section, select **After**, and click the **Apply** button.
7. The URL for this view will contain two arguments, the `nid` of the content to display and the `tid` (term ID) of the term with which to relate other pieces of content. In my case, the URL is `related-content/92/24`; yours is probably different:



What NID and TID do I use?

The easiest way to determine a node's ID (`nid`) or term's ID (`tid`) is to go to the admin screen that lists nodes (`admin/content`) or terms (`admin/structure/taxonomy/your_vocabulary_name`), find the node or term, hover your mouse over its **edit** link and look at the URL that appears at the bottom of the screen for the number.

How it works...

The secrets of this recipe are inherited arguments (contextual filters) and taxonomy term depth.

The contextual filter defined in the page display, `nid`, is then passed to the attachment display. However, we do not want to use that argument, because we will not be retrieving content based on the `nid` in the attachment, we will be retrieving content based on the `tid`. We have to tell views it will not be used. It's going to be in the URL, but we can tell the attachment that it is something other than it is. In selecting **Global: null** in the first contextual filter position in the attachment, we specified that the argument is null and to ignore it. So, why have it defined at all in the first place? The **page** display uses the argument to determine which content to display.

The second contextual filter is the `tid`. You might notice that we did not define a second argument on the page display. Drupal allows you to put as much extra information in the URL as you would like, and any arguments that are not defined are merely passed on and ignored. So, the page display ignores the second argument and passes it on to the attachment display. Why didn't we do the same kind of thing with the first argument and the attachment display... just omit it? The attachment display would have no way of knowing that the provide argument is meant to be the second one rather than the first.

If we were merely to retrieve content with the same `tid` as the requested content, we would have only that content to show in this example, because I only created one piece of content with the tag tourist info. However, our contextual filter allowed us to define depth.

We could have defined it as 0, meaning leave the choice as whichever `tid` we supplied as an argument, in which case we would have no related content to show, since only the one piece of content has that `tid`. We could also have defined it as negative, so that -1 would include the term's parent, -2 its grandparent, and so on. We defined it as 2, so that children and grandchildren terms would be included. The 2 tells views to select terms that are within 2 levels (generations) of the `tid`. When we added the terms in *Appendix B*, *Bundles*, and specified the relationship, the parent of each term was either tourist info or one of its children, putting all of them within two levels deep of **Tourist** info, so that any content with any of those terms gets selected.

The tag tree looks like this:

Depth:

0 Tourist info

1 Dining

1 Lodging

1 Sightseeing

2 Group tours

Related content – adding depth to a term

This recipe is almost identical to the previous one, with an interesting twist. If you have done the preceding recipe then you can jump ahead to the *How to do it...* section.

Getting ready

- We'll be using the article content type
- Create the taxonomy terms for this recipe, as shown in *Appendix B*, *Bundles*
- Create a small article for each of the terms

How to do it...

On the view list page, click the **+Add new view** link.

We will first enter the settings to create a page:

1. Enter `Related content 2` as the **View name**, check the checkbox for **Description**, enter `Use term id and depth to provide related content` in the textbox.
2. In the **Create a page** section, change the **Display format** select box from **Teasers** to **Full posts**, **Type** to **Articles**, **Items to display** to **1**, and click the **Continue & edit** button.

On the view edit page:

1. Click `Related content 2` next to **Title**, clear the textbox and click the **Apply (all displays)** button.
2. Click the link for **Post date** under **Sort Criteria** and then the **Remove** button.
3. Click the **Advanced** link to reveal the advanced settings.

4. Click the **Add** button for **Contextual filters**, check the checkbox for **Content: Nid**, and click the **Apply (all displays)** button.
5. Select **This page (override)** from the select box and click the **Apply (this display)** button.

That takes care of the page display; next we will create the **Attachment** display:

1. In the **Displays** section, check the button to **+Add** a display and select **Attachment**.
2. In the **Format** section, click the **Content** link, select **This attachment (override)** from the select box, select **Fields**, then click the **Apply (this display)** button.
3. Click the **Apply (this display)** button.
4. Click the **Add** button for **Contextual filters**, select **This attachment (override)** from the select box, check the boxes for **Content: Has taxonomy term ID**, **Content: Has taxonomy term ID depth modifier** and **Global: Null**, then click the **Apply (this display)** button.
5. In the configure box for **Content: Has taxonomy term ID**, click the **Apply (this display)** button.
6. In the configure box for **Content: Has taxonomy term ID** depth modifier, click the **Apply (this display)** button.
7. In the configure box for **Global: Null**, select **This attachment (override)** from the select box, and click the **Apply (this display)** button.
8. Click the drop-down arrow next to **Add** in the **Contextual filter** section and select **Rearrange**. Select **This attachment (override)** from the select box. **Drag Global: Null** to the top position and click the **Apply (this display)** button.
9. Click the **Not defined** link for **Attach to** in the **Attachment settings** section, check the checkbox for **Page**, then click the **Apply** button.
10. Click the link for **Before** in the **Attachment settings** section, change the setting to **After** and click the **Apply** button.
11. The URL for this view will contain three arguments: the `nid` of the content to display, the `tid` (term ID) of the term to relate content to, and the amount of generations preceding the term (negative) or following the term (positive) to include. In my case, the URL is `related-content-2/98/21/-1`; yours is probably different:

How it works...

The secrets of this recipe are inherited arguments (contextual filters) and the taxonomy term depth modifier.

The contextual filter defined in the page display, `nid`, is then passed to the attachment display. However, we do not want to use that argument, because we will not be retrieving content based on the `nid` in the attachment, we will be retrieving content based on the `tid`. We have to tell views it will not be used. It's going to be in the URL, but we can tell the attachment that it is something other than it is. In selecting **Global: null** in the first contextual filter position in the attachment, we specified that the argument is null and to ignore it. So, why have it defined in the first place? The **page** display uses the argument to determine which content to display.

The second contextual filter is the `tid`. You might notice that we did not define a second argument on the page display. Drupal allows you to put as much extra information in the URL as you would like, and any arguments that are not defined are merely passed on and ignored. So, the page display ignores the second argument and passes it on to the attachment display. Why didn't we do the same kind of thing with the first argument and the attachment display... just omit it? The attachment display would have no way of knowing that the provide argument is meant to be the second one rather than the first.

The third argument is a modifier to change the depth setting for selecting taxonomy terms. We could have defined it as 0, meaning leave the choice as whichever `tid` we supplied as an argument; however, we defined it as -1, so that the content tagged with the term's parent is included.

Limited visibility

In this recipe, we're going to create a block that lists the current user's content titles, and a main display to allow viewing selected content.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. We will first enter the settings to create a page.
3. Enter `Limited visibility` as the **View name**, check the checkbox for **Description**, and enter `Present the user's content` in the textbox.

4. The first two **Show** options should be **Content** and **Article**.
5. In the **Create a page** section, for **Display format** select **Unformatted list**, **full posts**, **without links**, and **without comments**.

Next, we will enter the settings to create a block:

1. Check the checkbox for **Create a block**.
2. Click the **Continue & edit** button.

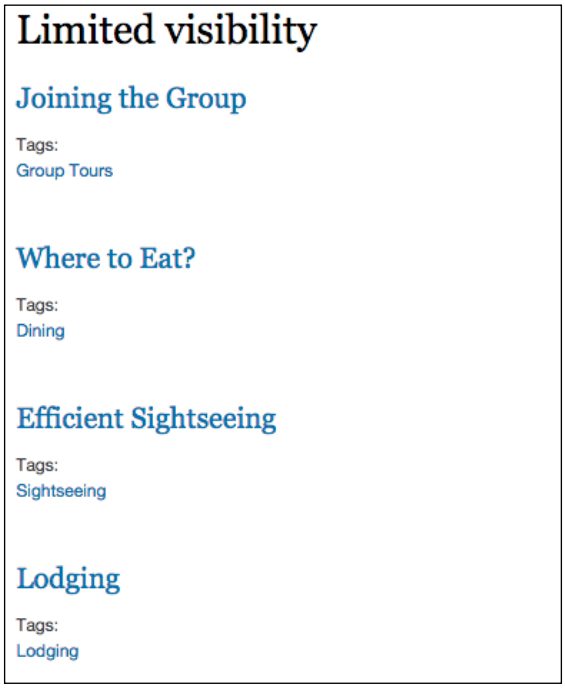
At this point we have defined the overall view, and now we can adjust its settings on the view edit page:

1. Click the **Advanced** link to reveal the advanced settings.
2. Click the **Add** button for **Contextual filters**, select **This page (override)** from the select box, check the checkbox for **Content: Nid**, then click the **Apply (this display)** button.
3. Click the **Apply (this display)** button.

That takes care of the page display; next we will make further adjustments to the block display:

1. Click the **Block** button in the **Displays** section.
2. Click the **Add** button for **Fields**, select **This block (override)** from the select box, check the checkbox for **Content: Nid**, and click the **Apply (this display)** button.
3. In the **Content: Nid** configuration window, uncheck the **Create a label** checkbox, check the **Exclude from display** checkbox, and click the **Apply (this display)** button.
4. Click the downarrow next to the **Add** button in the **Fields** section and select **Rearrange**, then drag the **Content: Nid** field to the top and click the **Apply (this display)** button.
5. Click the link for **Content: Title**, click the link to **Rewrite Results**, check the checkbox to **Output this field as a link**, enter `limited-visibility/[nid]` in the **Link path** textbox, and click the **Apply (this display)** button.
6. Click the **Add** button for **Relationships**, select **This block (override)** from the select box, check the checkbox for **Content: Author**, and click **Apply (this display)** button.
7. Click the **Apply (this display)** button.
8. Click the **Add** button for **Filter Criteria**, select **This block (override)** from the select box, check the checkbox for **User: Current**, and click the **Apply (this display)** button.
9. Click the **Yes** radio button for **Is the logged in user**, and click the **Apply (this display)** button.
10. Click the **None** link next to **Block name** in the **Block settings** section, enter **Limited visibility** into the textbox, and click the **Apply** button.

11. Save the view.
12. Navigate to the block admin page (`admin/structure/block`), find **Limited visibility** in the **Disabled** section, change the select box setting to **Sidebar** first, then click the **Save blocks** button.
13. Navigate to `limited-visibility`:



14. Then note the block, and that it only contains titles for content created by your current login:

How it works...

We created a block display that creates a list of titles for the content created by the currently logged in user, and displays those titles as links to a path with an argument of the `nid` of that content.

Next, we created a page display that will present a piece of content based on the ID that is passed in the URL, the ID that we appended the link with in the block display. Navigating to this URL without providing a `nid` will show all published content.

We could simply have chosen to display each title as a link to its node instead of outputting it as a link to `limited-visibility/<nid>`, but then clicking a title would bring up the node page for that node instead of using the display we created. In this example, it wouldn't make much of a difference, but it would if we choose to format the display differently than the node page.

5
Theming Views

In this chapter, we will cover:

- Changing the page template
- Creating and naming a views template
- Theming a field
- Theming a grid
- Theming a table
- Theming a row
- Theming rows
- Theming an RSS feed
- Theming a block
- Theming a view page
- Theming multiple displays

Introduction

For anything one needs to accomplish in Drupal, there is usually more than one way to do it, and asking a room full of Drupalers the best way will result in many opinions. This is also true with theming and using Views templates.

One can accomplish things in the overall template processing file, `template.php`, or in more specific template files, or even in pre-processing files. In some cases, formatting can be accomplished through CSS, whether inline or as part of a CSS file. In this chapter, we will learn to use templates to better format our views.

The subtheme and images used in this chapter are available for download. The instructions are given in *Appendix C, Resources*.

If you are using a theme that will be receiving updates from a theme developer or as part of a Drupal update, you will want to create a subtheme rather than modify the files of the theme itself; otherwise, your changes will probably be overwritten during any future update. See the article *sub-theme structure and inheritance* at `http://drupal.org/node/225125`.

Changing the page template

Templates are available for various formatting needs, such as nodes and views, but the template near the top of the heap, the page, is the page template. In this recipe, we will change the page template in a small way to illustrate how it is done, moving the footer to the top of the page.

Getting ready

The template file we will be changing is the `page.tpl.php` file in the recipes theme, a subtheme in `sites/all/themes/recipes`.

How to do it...

In your text editor of choice:

1. Edit the page template file `page.tpl.php` (`sites/all/themes/recipes/templates/`).
2. Around line 228, find the following line of code:

   ```
   <div id="footer-wrapper"><div class="section">
   ```

3. We're going to select all the lines from the previous line to this one, around line 245:

   ```
   </div></div> <!-- /.section, /#footer-wrapper -->
   ```

4. Cut the selection, and paste it just before the following code at about line 174:

   ```
   <div id="main-wrapper" class="clearfix"><div id="main"
   class="clearfix">
   ```

5. Save the file and navigate to the front page to find the **Powered by Drupal** line from the footer now at the top of the content section.

How it works...

A template is simply a file that results in HTML, which usually uses variables to allow dynamic elements on the page. The page template in Drupal has sections for the regions that are defined in the theme, such as the header, content area, footer, and sidebars. We simply moved the section for the footer from below the main content area to above it.

While that is not a change you would normally want to make, it serves to show you that if you add a new region to the theme, in addition to adding it to the list of regions in the theme's `.info` file, you need to let the theme know where to print the region's contents.

Now, let's narrow our focus from Drupal theme templates, in general, to Views templates.

Creating and naming a views template

The views template is used to make changes to the overall layout of a view, rather than to a display, table, grid, row, or field within the view. We're going alter a views template to give a view a title and subtitle, and change the color of its background.

Getting ready

The view we will be using is **Course list** from *Chapter 3*, *Intermediate Custom Views*.

How to do it...

1. Edit the course list view (`admin/structure/views/view/course_list/edit`).
2. Click the **Advanced** link to expand that section.
3. Click the **Information** link next to **Theme:** in the **Other** section.
4. In the **Page: Theming information** window, ensure that the theme you are using is selected in the select box.

If you select a different theme, be sure to click the **Change theme** button.

5. If your theme folder doesn't have a `templates` folder (`sites/all/themes/recipes/templates` if you are using the `recipes` theme), create it.
6. If your theme's `templates` folder doesn't have a `views` folder (`sites/all/themes/recipes/templates/views` if you are using the `recipes` theme), create it.
7. Copy the most specific (right-most) filename from the **Display output** line, which in our case is `views-view--course-list--page.tpl.php`.
8. Copy the `views-view.tpl.php` file from the `theme` directory of the `views` module (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`) to the `views template` folder in your theme that contains template files, and name it the name you copied (`views-view--course-list--page.tpl.php`).

Note that in places, the filename has one hyphen and, in other places, two. This is purposeful, and care must be taken when typing the name yourself.

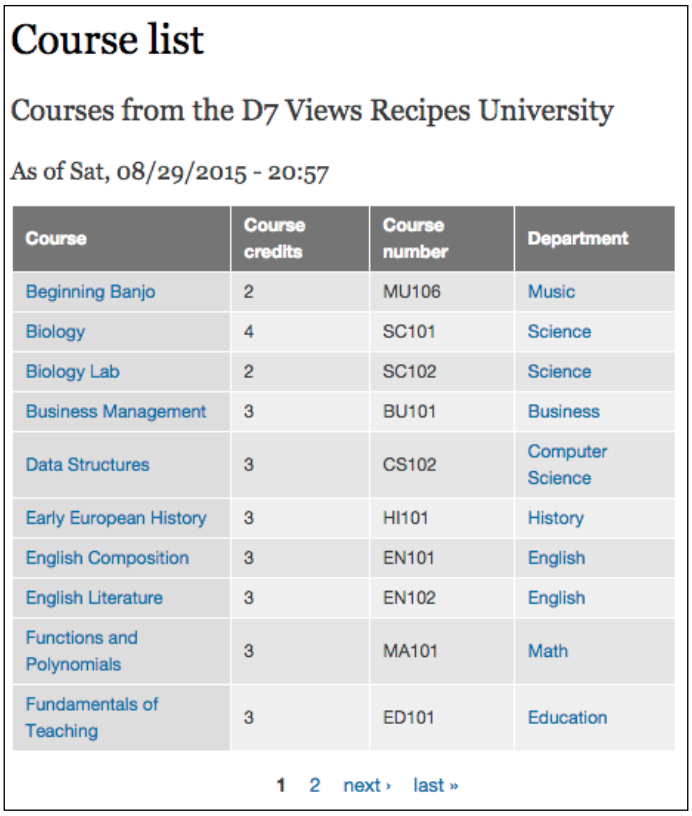
9. Edit the new file. Around line 54, look for:

```
<?php if ($rows): ?>
```

10. Immediately prior to that line, add the following:

```
<h2>Courses from the D7 Views Recipes University</h2>
<h3>As of <?php echo format_date(time(), 'custom', 'd
F,Y');?></h3>
```

11. Save the file, then clear the caches, since this is a new template file (`admin/config/development/performance`), and navigate to the view (`course-list`) to see the resulting headings:



Course list

Courses from the D7 Views Recipes University

As of Sat, 08/29/2015 - 20:57

Course	Course credits	Course number	Department
Beginning Banjo	2	MU106	Music
Biology	4	SC101	Science
Biology Lab	2	SC102	Science
Business Management	3	BU101	Business
Data Structures	3	CS102	Computer Science
Early European History	3	HI101	History
English Composition	3	EN101	English
English Literature	3	EN102	English
Functions and Polynomials	3	MA101	Math
Fundamentals of Teaching	3	ED101	Education

1 2 next › last »

How it works...

We selected the most specific file available to us for a template, which limited the template to the page display of the **Course list** view. If you take a look at the other file names in the **Display outline** list, if we were to choose one we would be dropping the view name, which then would have it apply to all views; or dropping the display name (page), which would have it apply to all current and future displays within our view (such as block, or attachment, or even another page display); or both, which would then impact all displays of all views.

Having selected the appropriate template filename, copying the template model file from the **Views** module template and renaming it, we made the changes to it and applied them. We inserted a title and subtitle, and used a snippet of PHP code to display a formatted date.

There's more...

Inserting HTML and PHP directly into the template file is not always the best approach, but the best one that we can use here, since we're creating quickly executed recipes. If *theming* is new to you, one topic to investigate is that of using the `template.php` file to create variables that can be referred to from within templates, rather than creating them inside a template or using long strings of inline code.

Theming a field

Sometimes we want a view to present data from more than one type of content, but we don't necessarily want all content types to be presented equally. We could want additional information in some cases, less in others, or, as is the case in this recipe, the same data points processed differently depending on the content type.

We're going to create a simple view that displays teasers from all published nodes, and create a template that allows us to display the node title links in varying ways, based on the content type.

Getting ready

This view will need at least one node of the content types **Article**, **Country**, **Course**, **Employee**, and **Extension** (details are given in *Appendix B*, *Bundles*), and one node from any of the other content types.

How to do it...

Create the view with the following steps:

1. Navigate to the views list (`admin/structure/views`) and click the **+Add new view** link.
2. Enter `Themed links` as the **View name**, check the **Description** textbox, and enter `Links themed based on the content type` in the textbox.
3. In the **Display format** select boxes of the **Create a page** section, change **teasers** to **fields**.
4. Change the **Items to display** to **25**, and click the **Continue & edit** button.
5. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Nid**, the checkbox for **Content: Path**, and the checkbox for **Content: Type**, then click the **Apply (all displays)** button.
6. In each of the three configuration windows that subsequently open, uncheck the **Create a label** checkbox, and click the **Apply (all displays)** button.

7. Click the **Advanced** link, and then the **Information** link next to **Theme** in the **Other** section.
8. In the **Page: Theming information** window, ensure that the theme you are using is selected in the select box.

If you select a different theme, be sure to click the **Change theme** button.

9. If your theme folder doesn't have a `templates` folder (`sites/all/themes/recipes/templates` if you are using the `recipes` theme), create it.
10. If your theme's `templates` folder doesn't have a `views` folder (`sites/all/themes/recipes/templates/views` if you are using the `recipes` theme), create it.
11. Scroll down and copy the right-most file name from the **Row style output** line, `views-view-fields--themed-links--page.tpl.php`, then click the **OK** button.
12. Click the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view-fields.tpl.php` and save it into the `views template` folder of the theme you're using (`sites/all/themes/your_theme/templates/views`), renaming it with the filename you copied: `views-view-fields--themed-links--page.tpl.php`.
2. Edit this new file and look around line 27 for:

```
<?php foreach ($fields as $id => $field): ?>
```

3. Immediately prior to this line, add the following:

```
<?php
  global $base_url;
  $t = $base_url . "/" . path_to_theme();
  switch ($fields['type']->raw) {
  case 'article':
    $fields['title']->content = "<img
    src='$t/images/article.png' /> 
    <a href='" . $fields['path']->content . "'>" .
    $fields['title']->content . "</a>";
    break;
  case 'country':
```

```
    $fields['title']->content = "<img
    src='$t/images/country.png' /> "
    . $fields['title']->content;
    break;
  case 'course':
    $fields['title']->content .= " 
    <a href='http://myschool.edu/courses/?course=" .
    $fields['nid']->content . "'><small>(more
    info)</small></a>";
    break;
  case 'extension':
    $fields['title']->content =
    "<a href='" . $fields['path']->content . "'>" .
    $fields['title']->content . "</a> <img
    src='$t/images/phone.png' />";
    break;
  case 'employee':
    $fields['title']->content =
    "<a href='" . $fields['path']->content . "'><img
    src='$t/images/employee.png' border='0' /> " .
    $fields['title']->content . "</a>";
    break;
  default:
    break;
  }
  unset($fields['path']);
  unset($fields['nid']);
  unset($fields['type']);
?>
```

4. Save the file.
5. Clear your caches (`admin/config/development/performance`).

Navigate to `themed-links` to see the following output:

The titles listed and their order could very well vary in your case, since they are being listed in the order in which they were created.

How it works...

The view itself is fairly straightforward. We select all published nodes, and select the fields that we will need later for our theme logic, even though only the title field will be visible.

The reason we did *not* elect to hide the other fields from being displayed by checking the appropriate box in the field's configuration is that we want the content of the field available to us in the `$fields` array in the template. Had we hidden them, we would still have access to their content, but in a more convoluted manner.

When selecting and creating the file to use for the template, the reason we used the fields-level template instead of field-level is that we needed *all* of the selected fields available at the same time. The fields level makes each row (piece of content), available as an array of fields, whereas had we used the field level, only an individual field would have been available at a time. For example, we could be looking at the content title, but would not have access to the content type.

Inside the `fields` template, the collection of rows (of content) is processed in a loop, with each field in the row being displayed. We needed to alter the contents of each row *before* it was displayed, so we placed our code before that loop.

In our code, we changed the value of the title field based on the content type. The `switch` statement was set up to process the specific content types in which we are interested, but leave the others (`default:`) untouched.

The following is a summary of the content types and what we did with the title of each:

- **Article**: A document icon, followed by the title linking to the node
- **Country**: A globe icon followed by the node title
- **Course**: The node title followed by a (more info) link to an external site, passing the content `nid`
- **Employee**: Both an employee icon and the title as links to the employee content
- **Extension**: The extension number as a link to the content, followed by an icon

Having made the changes to the title field, we no longer had a use for the node ID (`nid`), path, or content type fields, so we unset them, so that they would not be displayed as part of the fields content when the field collection was displayed.

There's more...

Any of the changes we made to the title could have been done individually via the field settings in the views UI, but when needing to account for several different content types and formats, that option would not be viable.

While you may never need to make the same kinds of change that we did here, this method can be used to change any included field or fields.

Theming a grid

Views gives you options of the way your output will be structured, such as a list, a table, and a grid. Whatever the structure, you are free to style the output. In this recipe we will style output structured as a grid.

Getting ready

We're going to use clone the view in the *Theming a field* recipe, and create a new display for it. We'll create a table that classifies nodes by content type.

If you have not tried that recipe yet, perform the *Getting ready* section and steps 1-12 from it at this point.

How to do it...

On the view list page, navigate to the views list (`admin/structure/views`), click the downarrow next to the **edit view name/description** button and select **Clone**, change **View name** to `Themed links 2`, and click the **Continue** button.

Create a new view display with the following steps:

1. Click `/themed_links` in the **Page Settings** section, change the path to `themed-links-2`, and click the **Apply** button.
2. Click the **+Add** button in the **Displays** section at the top and select **Page** as the display type.
3. Click the **Page** link next to **Display name** and enter `Grid Page` as the new name, then click the **Apply** button.
4. Click the **Themed links** link next to **Title**, select **This page (override)**, change the title to `Themed grid`, and click the **Apply (this display)** button.
5. Click **No path is set** next to **Path** in the **Page Settings** section, enter `themed-grid` as the path, and click the **Apply** button.
6. Click the link for **Content: Nid** in the **Fields** section, select **This page (override)** from the select box, then click the **Remove** button.
7. Click the link for **Content: Path** in the **Fields** section, select **This page (override)** from the select box, then click the **Remove** button.
8. Click the link for **Content: Type** in the **Fields** section, select T**his page (override)** from the select box, check the checkbox for **Output machine name**, and click the **Apply (this display)** button.

9. Click the **Unformatted list** link next to **Format:** in the **Format** section, select **This page (override)** from the select box, click the **Grid** radio button, and the **Apply (this display)** button.
10. In the **Grid Page: Style options** window, enter `content-type-[type]` in **Row class**, change the **Number of columns** to **3**, then click the **Apply (this display)** button.
11. Click the **Advanced** link, click the **Information** link next to **Theme** in the **Other** section, and copy the right-most file name from the **Style output** line, `views-view-grid--themed-links-2--page-1.tpl.php`, then click the **OK** button.
12. Click the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view-grid.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/recipes`, if you are using the recipe theme) that contains template files, naming it `views-view-grid--themed-links-2—page-1.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 26 for:

   ```
   <?php print $item; ?>
   ```

4. Immediately prior to this line add the following:

   ```
   <?php print $row_number * 3 + $column_number; ?>
   ```

5. Save the file.
6. Edit the CSS file for your theme. In my case it's called `style.css` and is in the `css` folder of the theme.
7. Add the following lines to the end of the file:

   ```
   td.content-type-article {background-color: #ffc0ff}
   td.content-type-blog-entry {background-color: #c0c0ff}
   td.content-type-country {background-color: #c0ffc0}
   td.content-type-course {background-color: #ffffc0}
   td.content-type-department {background-color: #ffc0c0}
   td.content-type-destination {background-color: #ffedc1}
   td.content-type-employee {background-color: #c1ffdb}
   td.content-type-extension {background-color: #dcc1ff}
   td.content-type-gallery {background-color: #dcc472}
   td.content-type-home {background-color: #cadc72}
   td.content-type-ingredient {background-color: #eebabb}
   td.content-type-product {background-color: #dff6f7}
   td.content-type-real-estate-flier {background-color: #f1ae85}
   td.content-type-sponsor {background-color: #edebf3}
   td.content-type-basic-page {background-color: #ffffff}
   ```

8. Save the file and navigate to `themed-grid` to see the following output:

Themed grid

0 Joining the Group article	1 Where to Eat? article	2 Efficient Sightseeing article
3 Lodging article	4 The Tourist Life article	5 Sudan country
6 Kazakhstan country	7 Argentina country	8 India country
9 Australia country	10 Brazil country	11 China country
12 USA country	13 Canada country	14 Russia country
15 711 extension	16 105 extension	17 Joe Jock employee
18 Physical Education department	19 J. Ayen Green employee	20 This title doesn't use the 'f' word article
21 This Week's Hot Properties! real_estate_flier	22 Data Structures course	23 Intro to Programming course
24 Fundamentals of Teaching course	25	26

1 2 3 next › last »

How it works...

This **Views** display selects all published nodes, and shows the title and content type of each as a grid cell.

We used a replacement tag `[type]` as part of the class name for the cell, so that cells can be themed by their content type. We chose to have the **Type** field output as the content type machine name. This field's value is used in the class name for the grid cell. Had we not output the content type as a machine name, such as `real-estate-flier`, it would have been put in the HTML as real estate flier, which would result in invalid mark-up. In conjunction with the class name, we added entries to the CSS file that provide a different background color for cells of each content type.

We also created a local copy of the grid template file and edited it to insert a cell number at the top of each cell.

Theming a table

Using the **Style** settings in the Views UI, we can elect to have a view output as table data using an HTML table. Our options, however, as to how the table is structured are limited. We can overcome these limitations by theming the table output.

Getting ready

We're going to clone the view from *Theming a field* recipe and modify the display in it. We'll create a table that classifies nodes by content type.

If you have not tried that recipe yet, perform the *Getting ready* section and steps 1-12 from it at this point.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`), click the downarrow next to the **edit view name/description** button and select **Clone**, change the **View name** to `Themed table`, and click the **Continue** button.
2. Click the **Themed links** link next to **Title**, select **This page (override)**, change the title to `Themed table`, and click the **Apply (all displays)** button.
3. Click **/themed-links** next to **Path** in the **Page settings** pane, enter `themed-table` as the path, and click the **Apply** button.
4. Click the link for **Content: Post date (desc)** in the **Sort Criteria** box, then click the **Remove** button.
5. Click the link for **Content: Nid** in the **Fields** box, then click the **Remove** button.

6. Click the link for **Content: Path** in the **Fields** box, then click the **Remove** button.
7. Click the **Unformatted list** link next to **Format:** in the **Format** pane, click the **Table** radio button and the **Apply (all displays)** button.
8. In the **Table Page: Style options** pane, check both **Sortable** checkboxes and the **Default Sort** radio button for **Content: Title**, then click the **Apply (all displays)** button.
9. Click the **Advanced** link, then the **Information** link next to **Theme** in the **Other** section, copy the right-most file name from the **Style output** line, `views-view-table--themed-table--page.tpl.php`, and click the **OK** button.
10. Click the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view-table.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it the name you copied earlier, `views-view-table--themed-table--page.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 22 for:

```
<table <?php if ($classes) { print 'class="'. $classes . '"
'; } ?><?php print $attributes; ?>>
```

4. Immediately prior to this line add the following:

```
<?php
$header['title'] = 'Title';
$header['Article'] = 'Article';
$header['Country'] = 'Country';
$header['Course'] = 'Course';
$header['Employee'] = 'Employee';
$header['Extension'] = 'Extension';
$header['Other'] = 'Other';
$fields['Article'] = 'type';
$fields['Country'] = 'type';
$fields['Course'] = 'type';
$fields['Employee'] = 'type';
$fields['Extension'] = 'type';
$fields['Other'] = 'type';
```

```
foreach ($rows as $count => $row) {
    $rows[$count]['Article'] = ' ';
    $rows[$count]['Country'] = ' ';
    $rows[$count]['Course'] = ' ';
    $rows[$count]['Employee'] = ' ';
    $rows[$count]['Extension'] = ' ';
    $rows[$count]['Other'] = ' ';
    $type =
    (in_array($row['type'],array('Article','Country',
    'Course','Employee','Extension'))) ? $row['type'] :
    'Other';
    $rows[$count][$type] = ($type == 'Other') ?
    $rows[$count]['type'] : 'X';
    unset($rows[$count]['type']);
}
unset($header['type']);
?>
```

5. At or about line 56 find:

```
<th <?php if ($header_classes[$field]) { print 'class="'.
$header_classes[$field] . '" '; } ?>>
```

6. Change the line to read:

```
<th <?php if (isset($header_classes[$field])) { print
'class="'. $header_classes[$field] . '" '; } ?>>
```

7. At or about line 69 find:

```
<td <?php //if ($field_classes[$field][$row_count]) { print
'class="'. $field_classes[$field][$row_count] . '" '; }
?><?php //print
drupal_attributes($field_attributes[$field][$row_count]);
?>>
```

8. Change the line to read:

```
<td>
```

9. Save the file, and then navigate to `themed-table` to view something similar to the following image:

Themed table

Title	Article	Country	Course	Employee	Extension	Other
105					X	
711					X	
Argentina		X				
Australia		X				
Beginning Banjo			X			
Biology			X			
Biology Lab			X			
Brazil		X				
Business						Department
Business Management			X			
Canada		X				
Chagall's Windows						Gallery
China		X				
Chocolate is the New Black of Food						Blog entry
Computer Science						Department
Data Structures			X			

How it works...

The view selects all published nodes, and we created a new page display that formats two fields from each node, the title and content type, as a HTML table. We selected only those two fields because the ultimate table is only meant to display the node title, and some manner of showing the content type.

Inside the **Table** template, there were three portions of the table that we needed to address.

The `$header` array contains a key for each column that is to have a heading. We added a column to it for each of several content types, and one catch-all column, titled **Other**, to summarize content of other types. We also reset the column heading for `type` that was there due to it being one of the fields we selected within the view. Had we not removed this column heading, the column headings would have been misaligned, making the table data appear to be incorrect.

The `$fields` array contains an entry for each field that is to be displayed, with the name of the class to be used when displaying it. We added an entry for each of the columns we were adding, and ultimately removed the entry `$fields['type']`, since it would not be needed.

The most important of the three arrays that we manipulated was the `$rows` array. It contains a key for each column of data to be displayed. We added a column for each of the specific content types that would appear in the table, as well as one for **Other**. We then examined the content type for the row. If the content type matched one of the specific content types (**Article**, **Country**, **Course**, **Employee**, and **Extension**), we simply inserted an *X* into the applicable column for that row. If, however, the content type was other than one of those five, we put the name of the content type into the **Other** column for that row rather than an *X* to make it more helpful.

Theming a row

A record selected for output in a view is a row of data when it reaches the **Views** templates, and often there is a need to format the rows. We will do that with a simple view to make each stand out more than they would otherwise.

Getting ready

We're going to use the **Country** content type, the details of which are given in *Appendix B*, *Bundles*.

How to do it...

Create the view with the following steps:

1. Navigate to the views list (`admin/structure/views`), click the **+Add new view** link, enter `Country countdown` as the **View name**, check the **Description** checkbox, and enter `Themed country rows` into the textbox.
2. Select **Country** from the **of type** select box.

3. In the **Create a page** section, change **teasers** to **fields** in the **Display format** section and click the **Continue & edit** button.
4. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Area**, and click the **Apply (all displays)** button.
5. In the configuration window for **Content: Area**, uncheck the **Create a label** checkbox, change the setting for **Scale** to **0**, and click the **Apply (all displays)** button.
6. Click the **Add** button in the **Sort Criteria** section, check the checkbox for **Content: Area (field_country_area)**, and click the **Apply (all displays)** button.
7. In the subsequent configuration window, check the checkbox for **Sort Descending** and click the **Apply (all displays)** button.
8. Click the link in the **Sort Criteria** section for **Content: Post date**, and click the **Remove** button.
9. Click the **Information** link next to **Theme:** in the **Other** pane and copy the right-most file name from the **Row style output** line, `views-view-fields--country-countdown—page-1.tpl.php`.
10. Click the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view-fields.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it `views-view-fields--country-countdown—page-1.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 27 for:

```
<?php foreach ($fields as $id => $field): ?>
```

4. Immediately prior to this line add the following:

```
<?php global $base_url; ?>
```

5. Then, after the last line in the file, add the following line:

```
<div style="margin: 6px"><img src="<?php echo $base_url .
'/' . file_stream_wrapper_get_instance_by_scheme('public')-
>getDirectoryPath();?>/country.png" /></div>
```

6. Save the file, and navigate to `country-countdown` to see output similar to the following image:

How it works...

The view selects nodes of the content type `Country`, and uses the fields title (the country name) and `country_area`, the size of the country. We sorted the countries from largest to smallest.

Inside the **Fields** template, we added a graphical divider between each row.

There's more...

There is more that we'd like to do with this view, but those changes require that we be able to compare each row to the next. In this template, we receive one row at a time, and so the template has no knowledge of the rows in bulk. The next recipe will address this.

Theming rows

This recipe is attached to the previous one to show the difference between theming a row and theming rows, and because it's important to show how different templates can be used in conjunction with each other.

We're going to make changes to the presentation of the view display and target rows based on their relation to the other rows around them.

Getting ready

If you have not used the previous recipe, *Theming a row*, do so first.

How to do it...

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme`), copy the file `views-view-unformatted.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it `views-view-unformatted--country-countdown—page-1.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 10 for:

```
<?php if (!empty($title)): ?>
```

4. Immediately prior to this line add the following:

```
<style type="text/css">
#cc-container {
  width: 180px;
}
.cc-odd, .cc-even {
  padding: 6px;
  border: 4px solid black;
  width: 120px;
  position: relative;
  text-align: center;
}
.cc-odd {
  left: 0;
  background-color: #aaa;
}
.cc-even {
  left: 60px;
  background-color: #eee;
}
.cc-value {
  font-size: 36px;
}
</style>
<?php $ctr = sizeof($rows) + 1; ?>
```

5. Find `<?php foreach ($rows as $id => $row): ?>` around line 37, and insert the following before it:

```
  <div id="cc-container">
```

6. Find `<div<?php if ($classes_array[$id]) { print ' class="' . $classes_array[$id] .'"'; } ?>>` around line 39, and insert the following lines before it:

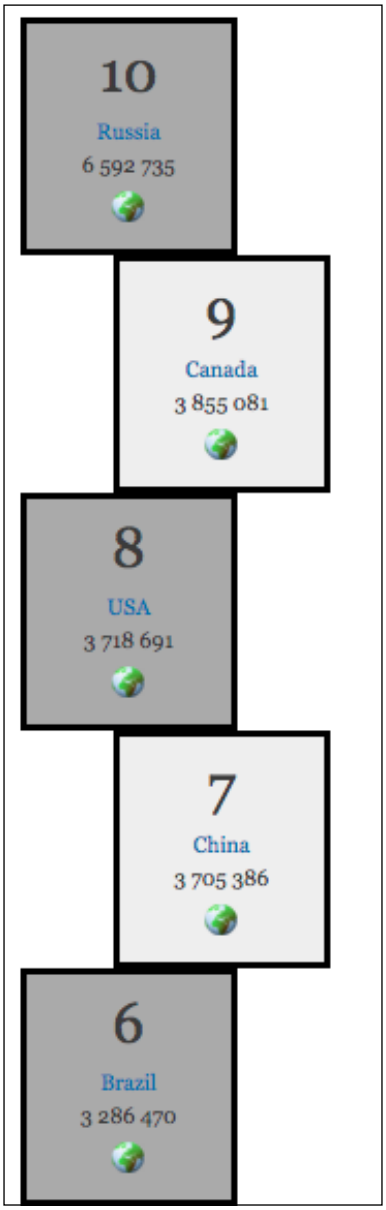
```
  <div class="cc-<?php echo ($ctr % 2) ? 'odd' : 'even';
  ?>">
      <?php $ctr--; ?>
      <div class="cc-value"><?php echo $ctr; ?></div>
```

7. Around line 45, find `<?php endforeach; ?>` and add `</div>` to both the line before it and the line after it.

8. The resulting code is as follows:

```
<style type="text/css">
#cc-container {
  width: 180px;
}
.cc-odd, .cc-even {
  padding: 6px;
  border: 4px solid black;
  width: 120px;
  position: relative;
  text-align: center;
}
.cc-odd {
  left: 0;
  background-color: #aaa;
}
.cc-even {
  left: 60px;
  background-color: #eee;
}
.cc-value {
  font-size: 36px;
}
</style>
<?php $ctr = sizeof($rows) + 1; ?>
<div id="cc-container">
<?php if (!empty($title)): ?>
  <h3><?php print $title; ?></h3>
<?php endif; ?>
<?php foreach ($rows as $id => $row): ?>
  <div class="cc-<?php echo ($ctr % 2) ? 'odd' : 'even';
?>">
      <?php $ctr--; ?>
      <div class="cc-value"><?php echo $ctr; ?></div>
  <div<?php if ($classes_array[$id]) { print ' class="' .
  $classes_array[$id] .'"';  } ?>>
    <?php print $row; ?>
  </div>
  </div>
  <?php endforeach; ?>
</div>
```

9. Save the file, and navigate to `country-countdown` to see output similar to the following image:



How it works...

We took an existing feed in which each row is themed, and added a template file to theme at the rows level, also known as the **unformatted level**. At the row (singular) level, a collection of fields is available for the row currently being processed. At the rows (plural) level, the collection of all rows is available.

We added code to the template file that counts down from the number of rows to 1, and styled the output so that the countdown number would display in large text. We also classified each row, based on the counter number, as odd or even by converting the countdown value to binary and calling it even if 0, odd if 1. We formatted based on whether the row was odd or even, one being to the right of the other, and using a different background color for each.

There's more...

There is so much that can be done at this level. We could *pluck* some of the rows, summarize them, and display them in a callout, for example.

We chose to embed a CSS style sheet within the template file rather than add it to a CSS file. Some would say that all CSS should be in the CSS file, if for no other reason than it is easier for the next person downstream to find. However, the specification *does* allow embedded styles, and if the style is not going to be used anywhere else, keeping it with the code for which it is a unique style also makes sense.

Theming an RSS feed

RSS feeds are used to provide information to other sites about the available content on your site. The **Views** options are less for feed displays than for others, because there is an expectation of the format on the receiving end. Still, some formatting can be done, and we will do a little here by adding a logo to the feed.

Getting ready

We're going to use the **Article** content type, which is included with Drupal 7.

How to do it...

On the view list page (`admin/structure/views`):

1. Click the **+ Add new view** link, enter `Articles` as the **View name**, check the box for **Description**, and enter `Articles feed` into the textbox.
2. In the **of type** select box in the **Show** pane, select **Article**.

3. Remove the check from the box for **Create a page**.
4. Click the **Continue & Edit** button.

Create the feed display with the following steps:

1. Click the **+ Add** button and select **Feed** to create a new display.
2. Click **Use site default RSS settings** next to **Show: Content** in the **Format** section, change the selection to **Teaser**, and click the **Apply** button.
3. Click **No path is set** next to **Path** in the **Feed settings** section, enter `articles/feed` as the path, and click the **Apply** button.
4. Click the **Advanced** link, click the **Information** link next to **Theme** in the **Other** panel, scroll down and copy the right-most file name from the **Style output** line, `views-view -rss--articles--feed-1.tpl.php`, and click the **Ok** button.
5. Click the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view- rss .tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it `views-view -rss--articles--feed-1.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 10 for:

```
<?php print "<?xml"; ?> version="1.0" encoding="utf-8"
<?php print "?>"; ?>
```

4. Immediately prior to this line add the following:

```
<?php global $base_url; ?>
```

5. Then, around line 14 look for:

```
<title><?php print $title; ?></title>
```

6. Insert the following immediately before it:

```
<image>
    <url><?php print $base_url . '/' .
    file_stream_wrapper_get_instance_by_
    scheme('public')->getDirectoryPath() . '/logo.png';
    ?></url>
    <title>Ayen Designs logo</title>
    <link><?php print $base_url; ?></link>
</image>
```

7. Save the file, and navigate to `articles/feed`. If you have an RSS reader, you will see something similar to this:

How it works...

We created a simple view that selects article content, and an RSS feed display for it. We then copied the RSS feed template from the Views theme directory, named it so that it would theme our particular view and display, and added XML to the template so our RSS feed page would include a logo.

There's more...

There are few changes that one can make to RSS feed templates, whether the one we worked on or the one that formats the row items, because the RSS specification is XML and only certain elements are defined, such as title and description.

More creative changes can be made to a RSS feed, but to do so requires making changes to the namespace, with which the XML elements are defined. This is outside the scope of this book, but certainly worth investigating if you would like to include, for example, images with your feed elements.

Theming a block

Blocks are ubiquitous in Drupal. With some, the content is inserted manually from the administration panel; in some it is generated within a module, and with others, the content comes from a view.

We can theme blocks just like any other display. We will clone an existing block display that lists recent content and make some changes to it via a template so that it offers a Facebook **Like** button for each listed content item.

Getting ready

We will be cloning the **Content topics** view from the final recipe in *Chapter 2*, *Basic Custom Views*, for this recipe.

How to do it...

On the view list page:

1. Navigate to the views list (`admin/structure/views`).
2. Click the downarrow next to **Edit** for the **Content topics** view and click the **Clone** link button.
3. Enter `Content topics Facebook` in the textbox and click the **Continue** button.
4. Click the **edit view name/description** link, enter `Bulleted list of topics with Facebook buttons` as the **View description**, and click the **Apply** button.
5. Click **Contents bullet** next to **Block name** in the **Block settings** section, enter `Contents bullet list with Facebook buttons` in the textbox, and click the **Apply** button.
6. Click the **Add** button for `Filter Criteria`, check the checkbox for **Content: Type**, and click the **Apply (all displays)** button.
7. Check the **Article** checkbox and click the **Apply (all displays)** button.
8. Click the **Add** button in the **Fields** section, check the checkboxes for **Content: Body** and for **Content: Path**, click the **Add (all displays)** button.
9. In the **Content: Body** configuration window, uncheck the **Create a label** checkbox, select **Trimmed** in the **Formatter** select box, set the **Trim length** to **100**, and click the **Apply (all displays)** button.
10. In the **Content: Path** configuration window, uncheck the **Create a label** checkbox, and click the **Apply (all displays)** button.

11. Click the **Advanced** link and then the **Information** link next to **Theme** in the **Other** section, scroll down and copy the right-most file name from the **Row style output** line, `views-view-fields--content-topics-facebook--block.tpl.php`.
12. Click the **Ok** button and then the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view- fields .tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it `views-view-fields--content-topics-facebook--block.tpl.php` like you copied earlier.
2. Edit this new file and look around line 27 for:

```
<?php foreach ($fields as $id => $field): ?>
```

3. Prior to this line insert the following:

```
<?php
$path = $fields['path']->content;
unset($fields['path']);
?>
```

4. Insert the following as the final line in the file:

```
<iframe src="<?=$path?>" scrolling="no"
frameborder="0"allowTransparency="true" style="border:none;
overflow:hidden; width:240px; height:px">
```

5. Save the file.
6. Click the **Information** link again and copy the right-most name in the **Display output** row, `views-view—content-topics-facebook—block.tpl.php` and click the **Ok** button.
7. Return to the **Views** module templates and copy the `views-view.tpl.php` template to your theme's `template` directory, and save it as the name you copied earlier.
8. Look for `<div class=<?php print $classes; ?>` at around line 30, and insert the following just before it:

```
<div id="fb-root"></div>
<script>(function(d, s, id) {
  var js, fjs = d.getElementsByTagName(s)[0];
  if (d.getElementById(id)) return;
  js = d.createElement(s); js.id = id;
  js.src = "//connect.facebook.net/en_US/sdk.js#xfbml=
  1&version=v2.4";
  fjs.parentNode.insertBefore(js, fjs);
}(document, 'script', 'facebook-jssdk'));</script>
```

9. Save the file.
10. Clear your caches (`admin/config/development/performance`).
11. Navigate to the **Blocks** admin page (`admin/content/block`).
12. Scroll down to the **Disabled** section and set **Contents bullet list with Facebook buttons** to **Sidebar first** and click the **Save blocks** button.
13. Navigate to home to view the block:

How it works...

We cloned an existing view that produces a bullet list of content titles. We added a field to the selection to capture the node path, and specified that only articles were to be selected.

We copied the `fields-level` template from the `views theme` folder to our theme folder and renamed it so that it would override the formatting of the block from our new view. Inside our template we set a variable equal to the contents of the node path field, and then removed that field from the `$fields` collection so that it would not be printed in the block. We pasted a line of code that requests a Like button from Facebook in the form of an iframe, and within that code snippet printed the contents of the field containing the node path, so that each button refers to the proper node.

The appearance of the iframe is controlled by Facebook, so it could look different than in the image earlier, and will likely look different to logged-in users than those not logged into Facebook.

Theming a view page

Sometimes, we will want to make changes to the entire view page display layout to give a page a unique look. The highest level of Views template allows us to do that. We'll create a view that displays items using a uniquely themed structure.

Getting ready

We're going to use the **Articles** view from the *Theming an RSS feed* recipe.

How to do it...

On the view list page (`admin/structure/views`):

1. Click the downarrow next to the **Edit** link for the **Articles** view and select **Clone**.
2. Enter `Articles with themed page` in the textbox and click the **Continue** button.

We could have simply added a page display to the existing view, but having a separate view makes it easier to make it available as a standalone feature that can be imported from this book's resource page.

Create the page display with the following steps:

1. Click the **+Add** button and select **Page**.
2. Click the **Fields** link next to **Show** in the **Format** section, select **This page (override)** from the select box, click the radio button for **Content**, and then click the **Apply (this display)** button.
3. Select **Teaser** for the **View mode** in the subsequent configuration window, then click the **Apply (this display)** button.
4. Click the **Paged, 10 items** link in the **Pager** section, select **This page (override)** from the select box, change the **10** to **3** in the **Items per page** textbox, put **1** into the **Number of pages** textbox, and click the **Apply (this display)** button.

5. Click **No path is set** in the **Page settings** section, enter `articles-list` as the path, and click the **Apply** button.
6. Click the **Advanced** link, then the **Information** link next to **Theme** in the **Other** section, scroll down and copy the right-most file name from the **Style output** line, `views-view-unformatted--articles-with-themed-page--page-1.tpl.php`.
7. Click the **Ok** button, and then the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view-unformatted.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, using the name that you copied earlier, `views-view-unformatted--articles-with-themed-page--page-1.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 10 for:

```
<?php if (!empty($title)): ?>
```

4. Insert the following before it:

```
<style type="text/css">
.article-1 {
    background-color: #ffaaaa;
    font-size: 14pt;
    border: 4px solid black;
}
.article-2 {
    background-color: #aaffaa;
    font-size: 12pt;
    border: 4px solid black;
}
.article-3 {
    background-color: #aaaaff;
    font-size: 10pt;
    border: 4px solid black;
}
</style>
```

5. Around line 30 find:

```
<?php //foreach ($rows as $id => $row): ?>
  <div class="<?php //print $classes_array[$id]; ?>">
    <?php //print $row; ?>
  </div>
<?php //endforeach; ?>
```

6. Replace those lines with the following:

```
<table id="articles-table">
    <tr>
        <td rowspan="2" class="article-1"><?php print
        $rows[0];?></td>
        <td class="article-2"><?php print $rows[1];?></td>
    </tr>
    <tr>
        <td class="article-3"><?php print $rows[2];?></td>
    </tr>
</table>
```

7. Save the file, and navigate to `articles-list` to see an image similar to the following one:

How it works...

We created a page display that selects three Article nodes. We didn't bother with verifying the selection within the template for this exercise, being comfortably assured that there will be at least three articles in our project site to draw from. We then modified the **Views** template that receives the selection rows as a collection and instead of looping through them and printing each the same, we printed each individually with specific formatting in mind.

We used a style sheet internal to the template rather than the CSS file. We also used a table to format the three records simply because it's an easier way to accomplish the goal within the scope of this book.

There's more...

This recipe showed how to manipulate a collection of rows within one display, but we can do more than that. We can have multiple displays in one view, all displayed simultaneously, and manipulate the data for them. We'll do just that in the next recipe.

Theming multiple displays

Having multiple displays displayed simultaneously allows us to use parts of the viewing area for distinct but related content, and, if desired, interactive functionality, much like a desktop application. This enables an enhanced user experience and increased site value.

In this recipe we will have a display that lists available courses; another that lists departments, which drives the course list; and a third display that provides course details. This view will also make use of contextual filters. Information about the **Course** and **Department** content types can be found in *Appendix B*, *Bundles*.

How to do it...

On the view list page (`admin/structure/views`):

1. Click the **+ Add new view** link.
2. Enter `Departments and courses` as the **View name**.
3. Check the box for **Description** and enter `Interactive course and department listing` in the textbox.
4. In the **Show** select boxes, change **sorted by** to **Title**.
5. On the **Display format** line of the **Create a page** section, change **teasers** to full **posts**.

6. Change **Items to display** to **1**, and click the **Continue & edit** button.
7. Click the **+ Add** button and select **Attachment**.
8. Click the **Attachment** link next to **Display name**, change the name to **Department list**, and click the **Apply** button.
9. Click the **+ Add** button and select **Attachment**.
10. Click the **Attachment 2** link next to **Display name:**, change the name to **Course list** and click the **Apply** button.

Create the **Course** list attachment with the following steps:

1. Click the **Content** link next to **Show** in the **Format** section, select **This attachment (override)** from the select box, click the **Fields** radio button, and click the **Apply (this display)** button.
2. Click the **Apply (this display)** button in the subsequent **Row style options** window.
3. Click the **Add** button in the **Fields** section, select **This attachment (override)** from the select box, check the checkbox for **Content: Nid**, and click **Apply (this display)** button.
4. Check the box for **Exclude from display** and click the **Apply (this display)** button.
5. Click the downarrow next to the **Add** button in the **Fields** section, select **Rearrange**, drag the **Nid** field above the **Title** field, and click the **Apply (this display)** button.
6. Click the **Content: Title** link in the **Fields** section and uncheck the checkbox for **Link this field to the original piece of content**.
7. Click the **Rewrite Results** link to reveal its settings, check **Output this field as a link**, enter `departments-and-courses/!1/[nid]` in the **Link path** textbox, and click the **Apply (this display)** button.
8. Click the **Add** button in the **Filter Criteria** section.
9. Select **This attachment (override)** from the select box, check the checkbox for **Content: Type** and click the button for **Apply (this display)**.
10. Check the checkbox for **Course** and click the **Apply (this display)** button.
11. In the **Attachment settings** section, click the **Not defined** link next to **Attach to**, check the checkbox for **Page**, and click the **Apply** button.
12. Click the link for **Before** next to **Attachment position:**, change the setting to **After**, and click the **Apply** button.

Configure the **Department** list attachment with the following steps:

1. Click the button for the **Department** list display.
2. Click the **Content** link next to **Show** in the **Format** section, change **All displays** to **This attachment (override)**, click the **Fields** radio button, and click the **Apply (this display)** button.
3. Click the **Apply (this display)** button in the subsequent window.
4. Click the **Add** button in the **Fields** section, select **This attachment (override)** from the select box, check the checkbox for **Content: Nid**, and click the **Apply (this display)** button.
5. Check the box for **Exclude from display** and click the **Apply (this display)** button.
6. Click the downarrow next to the **Add** button in the **Fields** section, select **Rearrange**, drag the **Nid** field above the **Title** field, and click the **Apply (this display)** button.
7. Click the **Content: Title** link in the **Fields** section, select **This attachment (override)** from the select box, and remove the check from **Link this field to the original piece of content**, click the **Rewrite Results** link to reveal its settings, and check **Output this field as a link** checkbox.
8. Click the **Add** button in the **Filter Criteria** section, select **This page (override)** from the select box, check the checkbox for **Content: Type**, and click the **Apply (this display)** button.
9. Check the **Department** checkbox, then click the **Apply (this display)** button
10. In the **Link path** textbox, enter `departments-and-courses/[nid]` and click the **Apply (this display)** button.
11. In the **Attachment settings** section, click the **Not defined** link next to **Attach to**, check the checkbox for **Page**, and click the **Apply** button.
12. Ensure that the **Attachment position** link reads **Before**.
13. Click the **Apply (this display)** button.

Complete the page display with the following steps:

1. Click the **Page display** button.
2. Click the **Add** button in the **Filter Criteria** section, select **This page (override)** from the select box, check the checkbox for **Content: Type**, and click the **Apply (this display)** button.
3. Check the **Course** checkbox, then click the **Apply (this display)** button.
4. Click the **Advanced** link.

5. Click the **Add** button in the **Contextual filters** section, select **This page (override)** from the select box, check the checkbox for **Global: Null** and click the **Apply (this display)** button.
6. Beneath **When the filter value is NOT in the URL**, click the radio button for **Display contents of "No results found"** and click the **Apply (this display)** button.
7. Click the **Add** button in the **Contextual filters** pane, check the box for **Content: Nid**, and click the **Apply and configure contextual filters** button.
8. Select **This page (override)**, and beneath **When the filter value is NOT in the URL** click the radio button for **Display contents of "No results found**" and click the **Apply** button.
9. Also in the **Advanced** section, click the **Add** button for **No Results Behavior**, select **This page (override)** from the select box, select **Global: Text area**, and click the **Apply (this display)** button.
10. In the subsequent configuration pane, select **This page (override)**, enter **Select a course** as the **Label**, enter `<h2>Please select a department and then a course</h2>` in the large textbox, and click the **Apply (this display)** button.
11. Click the **Information** link next to **Theme** in the **Other** section, scroll down and copy the right-most file name from the **Display output** line, `views-view--departments-and-courses--page.tpl.php`.
12. Click the **Ok** button and then the **Save** button.

Create the template with the following steps:

1. From the **Views** module `theme` directory (probably `sites/all/modules/views/theme` or `sites/all/modules/contrib/views/theme`), copy the file `views-view.tpl.php` and save it into the directory of the theme you're using (`sites/all/themes/your_theme`) that contains template files, naming it `views-view--departments-and-courses--page.tpl.php`.
2. Clear your caches (`admin/config/development/performance`).
3. Edit this new file and look around line 48 for:

```
<?php if ($attachment_before): ?>
  <div class="attachment attachment-before">
    <?php print $attachment_before; ?>
  </div>
<?php endif; ?>
```

4. Select that block of code, cut it, and paste it around line 62 immediately before:

```
<?php if ($attachment_after): ?>
```

5. Around line 74 look for:

```
<?php if ($more): ?>
```

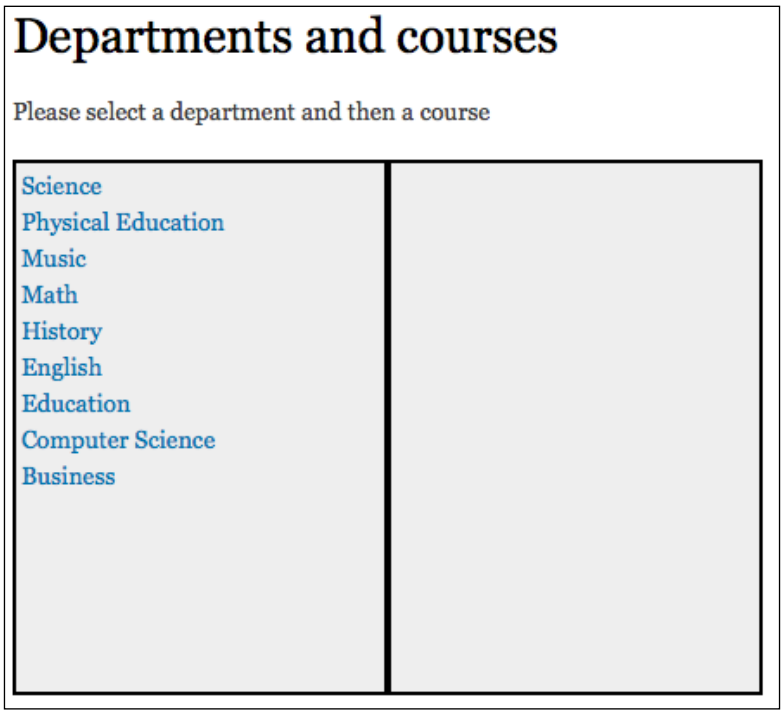
6. Immediately before that line enter:

```
<div style="clear:both"></div>
```

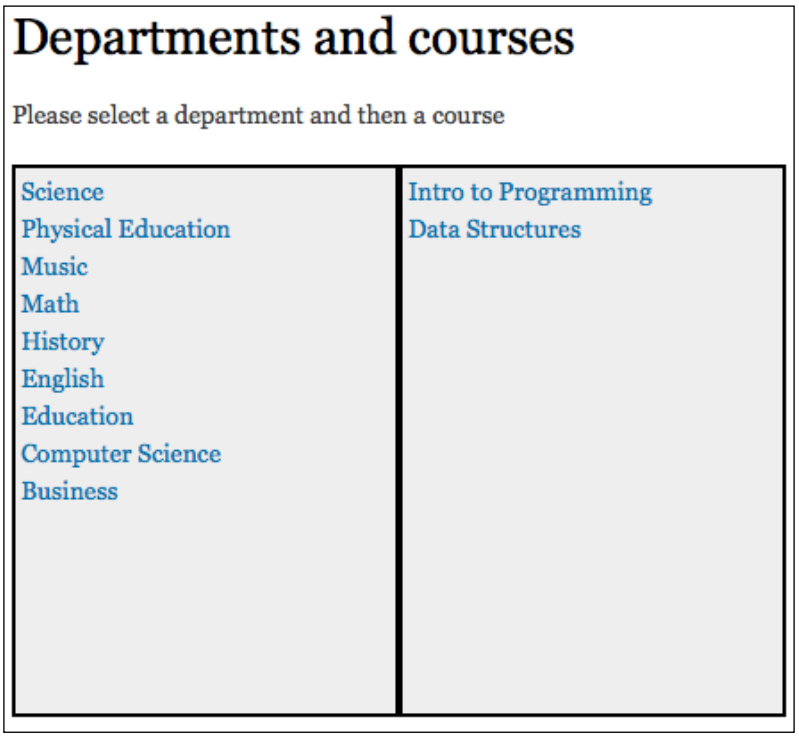
7. Edit your theme's CSS file (in my case it's `style.css`) and add the following at the bottom of the file:

```
.page-departments-and-courses .attachment-before,
.page-departments-and-courses .attachment-after {
  border: 2px solid black;
  width: 46%;
  min-height: 300px;
  background-color: #eee;
  float: left;
  padding: 3px;
}
```

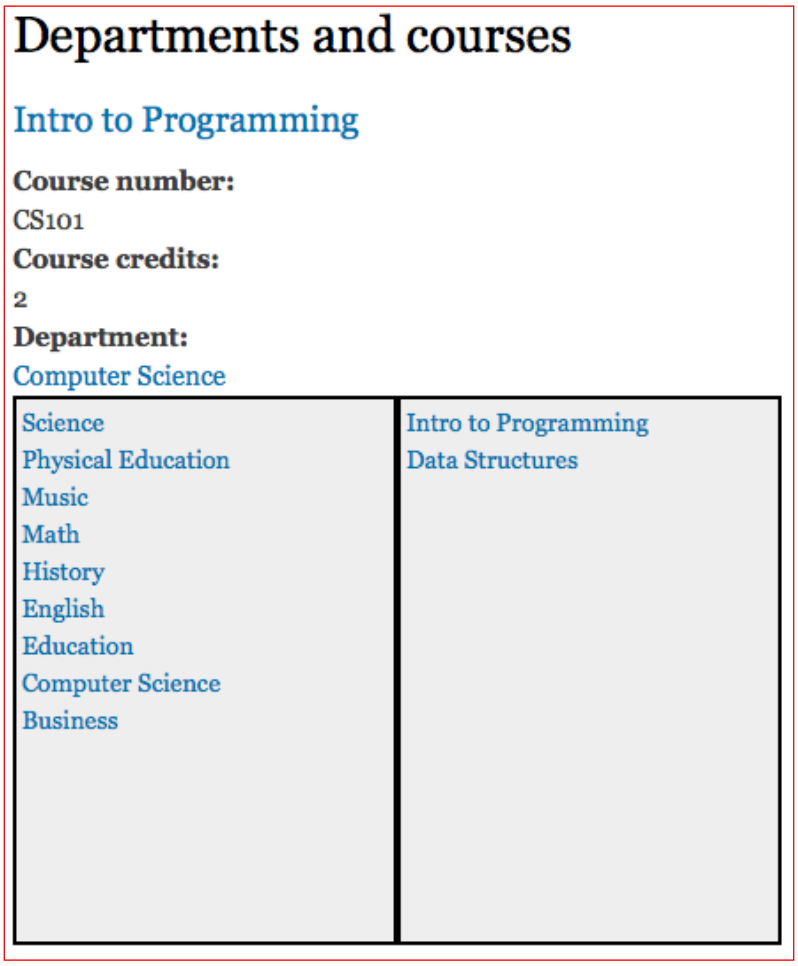
8. Save the file, clear your caches, and navigate to `departments-and-courses`:



9. Click on a department name:



10. Click a course for that department:



How it works...

The secret of this recipe is the use of attachment displays to augment the normal content of the page display, and contextual filters moving data between the steps.

The first step is to display a list of departments in the first attachment display, and to rewrite the department titles as links back to the *same* page, `departments-and-courses`, but containing an argument that is the department node ID (`nid`), such as `departments-and-courses/32`. We set the page to display **Please select a department and then a course** until a course is selected.

The second attachment display is set up to display a list of courses, *but only courses that belong to the chosen department*. We did this with a contextual filter that takes the `nid` passed in the URL and matches it to the department `nid` that is present in course records, so every course that points to that department will be selected and used to populate the course list. We again rewrite the title as a link, in this case the course title, but this time we pass a second argument in the URL, the `nid` of the *course*.

The page display provides the course details and uses the second URL argument to identify which course to display based on the course `nid`.

We edited the views template to move the *before* attachment to appear after the content rather than in its normal position, and then edited the theme CSS file to format the two attachments and have them appear side by side.

There's more...

For more information on using multiple displays, see Packt Publishing's book, *Drupal 6 Attachment Views* at `https://www.packtpub.com/drupal-6-attachment-views/book`. The concepts in it apply to Drupal 7 as well.

6
Programmatic Views

In this chapter, we will cover:

- Coding a view
- Handling a view field
- Styling a view field
- Fine-tuning the query

Introduction

In this chapter, we will switch from the admin user interface to code, creating a view within a module, and other examples of using code in conjunction with the Drupal and views architectures to manipulate content. These methods should only be considered if you are comfortable with PHP and the Drupal architecture and API. The benefits of using code include more granular control as well as the ability to achieve behaviors otherwise unavailable. The drawbacks are that the views environment can be very complex, and one can easily break the environment. I highly advise using a development environment for coding experimentation, and note that the typical first symptom of a syntax error in the code is a **white screen of death** (**WSOD**)...a blank screen instead of the page you expected to see.

The differences between using the Views UI to create a view and doing it in a module is that the UI "does the coding" for you and makes it more convenient to make changes to the view afterwards. That said, the UI does not make it easy to distribute a ready-made view, nor does it make it facilitate tying such a view to other code.

Coding a view

Creating a view in a module is a convenient way to have a predefined deployable view available with Drupal. As long as the module is installed and enabled, the view will be there to be used. If you have never created a module in Drupal, or even never have written a line of Drupal code, you will still be able to create a simple view using this recipe, which creates a `list_all_nodes` view.

Getting ready

Creating a module involves the creation of two files at a minimum:

- An `.info` file that gives Drupal the information needed to add the module
- A `.module` file that contains the PHP script

More complex modules will consist of more files, but these two are all we will need. Feel free to download the code used in this recipe from the author's website rather than typing it: `http://accidentalcoder.com/content/drupal-7-views-cookbook-revised`.

How to do it...

1. If it isn't present, already, create a new directory, `custom`, inside your modules directory (probably `sites/all/modules`).
2. Create a subdirectory inside that directory; we will call it `d7vc` (*Drupal 7 Views Cookbook*).
3. In the newly created `d7vc` directory, open a new file with your editor and add the following lines:

```
; $Id:
name = Programmatic Views
description = Provides supplementary resources such as
programmatic views
package = D7 Views Recipes

version = "7.x-1.0"
core = "7.x"
```

4. Save the file as `d7vc.info`.
5. Open another new file and add the following lines:

Don't panic when looking at the amount of the following code...in *Chapter 7, But Wait – There's More*, you learn how you can have it produced for you.

```
<?php
/**
 * Implements hook_views_api().
 */
function d7vc_views_api() {
  return array(
    'api' => 3,
    'path' => drupal_get_path('module', 'd7vc'),
  );
}
/**
 * Implements hook_views_default_views().
 */
function d7vc_views_default_views() {
    return d7vc_list_all_articles();
}
/**
 * Begin view
 */
function d7vc_list_all_articles() {
  $view = new view();
  $view->name = 'list_all_articles';
  $view->description = 'Provide a list of titles, creation
  dates, owner and status';
  $view->tag = '';
  $view->base_table = 'node';
  $view->human_name = 'List all articles';
  $view->core = 0;
  $view->api_version = '3.0';
  $view->disabled = FALSE; /* Edit this to true to make a
  default view disabled initially */

  /* Display: Master */
  $handler = $view->new_display('default', 'Master',
  'default');
  $handler->display->display_options['use_more_always'] =
  FALSE;
  $handler->display->display_options['access']['type'] =
  'none';
```

```
$handler->display->display_options['cache']['type'] =
'none';
$handler->display->display_options['query']['type'] =
'views_query';
$handler->display->display_options['exposed_form']
['type'] = 'basic';
$handler->display->display_options['pager']['type'] =
'full';
$handler->display->display_options['style_plugin'] =
'table';
$handler->display->display_options['style_options']
['columns'] = array(
  'title' => 'title',
  'type' => 'type',
  'uid' => 'uid',
  'created' => 'created',
  'status' => 'status',
);
$handler->display->display_options['style_options']
['default'] = 'title';
$handler->display->display_options['style_options']
['info'] = array(
  'title' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => '',
    'separator' => '',
    'empty_column' => 0,
  ),
  'type' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => '',
    'separator' => '',
    'empty_column' => 0,
  ),
  'uid' => array(
    'align' => '',
    'separator' => '',
    'empty_column' => 0,
  ),
  'created' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => '',
```

```
      'separator' => '',
      'empty_column' => 0,
    ),
    'status' => array(
      'sortable' => 1,
      'default_sort_order' => 'asc',
      'align' => '',
      'separator' => '',
      'empty_column' => 0,
    ),
  );
  /* Relationship: Content: Author */
  $handler->display->display_options['relationships']
  ['uid']['id'] = 'uid';
  $handler->display->display_options['relationships']
  ['uid']['table'] = 'node';
  $handler->display->display_options['relationships']
  ['uid']['field'] = 'uid';
  /* Field: Content: Title */
  $handler->display->display_options['fields']['title']
  ['id'] = 'title';
  $handler->display->display_options['fields']['title']
  ['table'] = 'node';
  $handler->display->display_options['fields']['title']
  ['field'] = 'title';
  /* Field: Content: Type */
  $handler->display->display_options['fields']['type']
  ['id'] = 'type';
  $handler->display->display_options['fields']['type']
  ['table'] = 'node';
  $handler->display->display_options['fields']['type']
  ['field'] = 'type';
  /* Field: Content: Post date */
  $handler->display->display_options['fields']['created']
  ['id'] = 'created';
  $handler->display->display_options['fields']['created']
  ['table'] = 'node';
  $handler->display->display_options['fields']['created']
  ['field'] = 'created';
  $handler->display->display_options['fields']['created']
  ['date_format'] = 'custom';
  $handler->display->display_options['fields']['created']
  ['custom_date_format'] = 'Y-m-d';
  $handler->display->display_options['fields']['created']
  ['second_date_format'] = 'long';
  /* Field: User: Name */
```

```
$handler->display->display_options['fields']['name']
['id'] = 'name';
$handler->display->display_options['fields']['name']
['table'] = 'users';
$handler->display->display_options['fields']['name']
['field'] = 'name';
$handler->display->display_options['fields']['name']
['relationship'] = 'uid';
$handler->display->display_options['fields']['name']
['label'] = 'Author';
/* Field: Content: Published */
$handler->display->display_options['fields']['status']
['id'] = 'status';
$handler->display->display_options['fields']['status']
['table'] = 'node';
$handler->display->display_options['fields']['status']
['field'] = 'status';
$handler->display->display_options['fields']['status']
['not'] = 0;
/* Filter criterion: Content: Type */
$handler->display->display_options['filters']['type']
['id'] = 'type';
$handler->display->display_options['filters']['type']
['table'] = 'node';
$handler->display->display_options['filters']['type']
['field'] = 'type';
$handler->display->display_options['filters']['type']
['value'] = array(
  'article' => 'article',
);

/* Display: Defaults */
$handler = $view->new_display('default', 'Defaults',
'de-fault');
$handler->display->display_options['title'] = 'List All
Articles';
$handler->display->display_options['use_more_always'] =
FALSE;
$handler->display->display_options['access']['type'] =
'role';
$handler->display->display_options['access']['role'] =
array(
  3 => '3',
);
$handler->display->display_options['cache']['type'] =
'none';
```

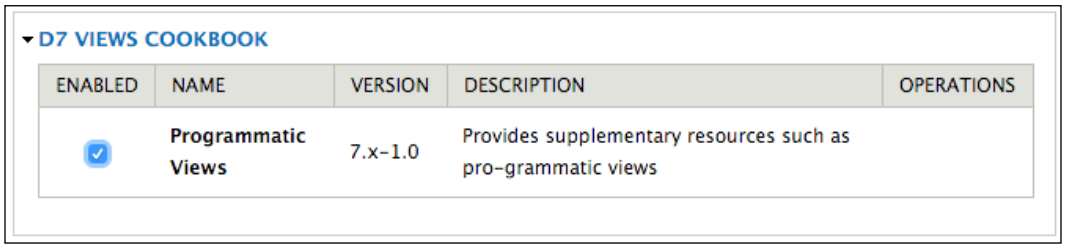
```
$handler->display->display_options['query']['type'] =
'views_query';
$handler->display-
>display_options['exposed_form']['type'] = 'basic';
$handler->display->display_options['pager']['type'] =
'full';
$handler->display->display_options['pager']
['options']['items_per_page'] = '15';
$handler->display->display_options['pager']
['options']['offset'] = '0';
$handler->display->display_options['pager']
['options']['id'] = '0';
$handler->display->display_options['style_plugin'] =
'table';
$handler->display->display_options['style_options']
['columns'] = array(
  'title' => 'title',
  'type' => 'type',
  'created' => 'created',
  'name' => 'name',
  'status' => 'status',
);
$handler->display->display_options['style_options']
['default'] = 'title';
$handler->display->display_options['style_options']
['info'] = array(
  'title' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => 'views-align-left',
    'separator' => '',
    'empty_column' => 0,
  ),
  'type' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => 'views-align-left',
    'separator' => '',
    'empty_column' => 0,
  ),
  'created' => array(
    'sortable' => 1,
    'default_sort_order' => 'asc',
    'align' => 'views-align-left',
    'separator' => '',
```

```
      'empty_column' => 0,
    ),
    'name' => array(
      'sortable' => 1,
      'default_sort_order' => 'asc',
      'align' => 'views-align-left',
      'separator' => '',
      'empty_column' => 0,
    ),
    'status' => array(
      'sortable' => 1,
      'default_sort_order' => 'asc',
      'align' => 'views-align-left',
      'separator' => '',
      'empty_column' => 0,
    ),
  );
  /* Header: Global: Text area */
  $handler->display->display_options['header']['area']
  ['id'] = 'area';
  $handler->display->display_options['header']['area']
  ['table'] = 'views';
  $handler->display->display_options['header']['area']
  ['field'] = 'area';
  $handler->display->display_options['header']['area']
  ['empty'] = TRUE;
  $handler->display->display_options['header']['area']
  ['content'] = '<h2>Following is a list of all
  articles.</h2>';
  $handler->display->display_options['header']['area']
  ['format'] = '3';
  /* Footer: Global: Text area */
  $handler->display->display_options['footer']['area']
  ['id'] = 'area';
  $handler->display->display_options['footer']['area']
  ['table'] = 'views';
  $handler->display->display_options['footer']['area']
  ['field'] = 'area';
  $handler->display->display_options['footer']['area']
  ['empty'] = TRUE;
  $handler->display->display_options['footer']['area']
  ['content'] = '<small>This view is brought to you
  courtesy of the D7 Views Cookbook module</small>';
  $handler->display->display_options['footer']['area']
  ['format'] = '3';
  /* Relationship: Content: Author */
```

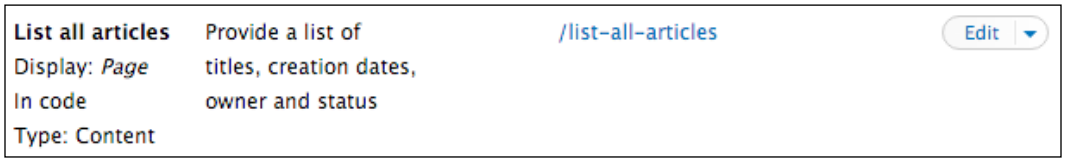
```
$handler->display->display_options['relationships']
['uid']['id'] = 'uid';
$handler->display->display_options['relationships']
['uid']['table'] = 'node';
$handler->display->display_options['relationships']
['uid']['field'] = 'uid';
/* Field: Content: Title */
$handler->display->display_options['fields']['title']
['id'] = 'title';
$handler->display->display_options['fields']['title']
['table'] = 'node';
$handler->display->display_options['fields']['title']
['field'] = 'title';
$handler->display->display_options['fields']['title']
['link_to_node'] = FALSE;
/* Field: Content: Type */
$handler->display->display_options['fields']['type']
['id'] = 'type';
$handler->display->display_options['fields']['type']
['table'] = 'node';
$handler->display->display_options['fields']['type']
['field'] = 'type';
/* Field: Content: Post date */
$handler->display->display_options['fields']['created']
['id'] = 'created';
$handler->display->display_options['fields']['created']
['table'] = 'node';
$handler->display->display_options['fields']['created']
['field'] = 'created';
$handler->display->display_options['fields']['created']
['date_format'] = 'custom';
$handler->display->display_options['fields']['created']
['custom_date_format'] = 'Y-m-d';
/* Field: User: Name */
$handler->display->display_options['fields']['name']
['id'] = 'name';
$handler->display->display_options['fields']['name']
['table'] = 'users';
$handler->display->display_options['fields']['name']
['field'] = 'name';
$handler->display->display_options['fields']['name']
['relationship'] = 'uid';
$handler->display->display_options['fields']['name']
['label'] = 'Author';
$handler->display->display_options['fields']['name']
['link_to_user'] = FALSE;
/* Field: Content: Published */
```

```
    $handler->display->display_options['fields']['status']
    ['id'] = 'status';
    $handler->display->display_options['fields']['status']
    ['table'] = 'node';
    $handler->display->display_options['fields']['status']
    ['field'] = 'status';
    $handler->display->display_options['fields']['status']
    ['type'] = 'true-false';
    $handler->display->display_options['fields']['status']
    ['not'] = 0;
    /* Filter criterion: Content: Type */
    $handler->display->display_options['filters']['type']
    ['id'] = 'type';
    $handler->display->display_options['filters']['type']
    ['table'] = 'node';
    $handler->display->display_options['filters']['type']
    ['field'] = 'type';
    $handler->display->display_options['filters']['type']
    ['value'] = array(
      'article' => 'article',
    );

    /* Display: Page */
    $handler = $view->new_display('page', 'Page', 'page_1');
    $handler->display->display_options['path'] = 'list-all-
    articles';

    $views[$view->name] = $view;
    return $views;
}
```

6. Save the file as `d7vc.module`.
7. Navigate to the modules admin page at `admin/modules`.
8. Scroll down to the new module and activate it—make sure to click the **Save Configuration** button at the bottom of the page:



9. Navigate to the views admin page (`admin/structure/views`) to verify that the view appears in the list. Note that is shows the view as **Page in code** rather than the normal **Page in database**:



List all articles Display: *Page* In code Type: Content	Provide a list of titles, creation dates, owner and status	/list-all-articles	Edit

10. Clear the cache (`admin/config/development/performance`) to update the menu router so it will recognize the new path.
11. Finally, navigate to `list-all-articles` to see the view:

Title	Type	Post date	Author	Published
Efficient Sightseeing	Article	2015-08-23	ayen	Yes
Joining the Group	Article	2015-08-23	jag	Yes
Lodging	Article	2015-08-23	ayen	Yes
Test article 1	Article	2015-06-24	ayen	Yes
The Tourist Life	Article	2015-08-23	ayen	Yes
This title doesn't use the 'f' word	Article	2015-08-19	ayen	Yes
Unpublished article	Article	2015-08-04	ayen	No
Unpublished content 1	Article	2015-07-10	ayen	No
Where to Eat?	Article	2015-08-23	ayen	Yes

How it works...

The module we have just created could have many other features associated with it beyond simply a view, and enabling the module will make those features and the view available; disabling it will hide those same features and view.

When compiling the list of installed modules, Drupal looks first in its own modules directory for `.info` files, then in the sites modules directories. As can be deduced from the fact that we put our `.info` file in a second-level directory of `sites/all/modules` and it was found there, Drupal will traverse the modules directory tree looking for `.info` files.

We created a `.info` file that provided Drupal with the name and description of our module, its version, the version of Drupal it is meant to work with, and a list of files used by the module, which in our case is just one.

We saved the `.info` file as `d7vc.info` (Drupal 7 Views recipes programmatic view); the name of the directory in which the module files appear (`d7vr`) has no bearing on the module itself.

The module file contains the code that will be executed, at least initially. Drupal doesn't actively `call` module code in an active way; instead, there are events that occur during Drupal's creation of a page, and modules can elect to register with Drupal to be notified of such events when they occur so that the module can provide code to be executed at that time, sort of like you registering with a business to receive an e-mail in the event of a sale. Just like you are free to act or not, but the sales goes on regardless, Drupal continues whether or not the module decides to do something or not when given the chance.

Our module `hooks` the `views_api` and `views_default_views` events in order to establish the fact that we do have a view to offer. The latter hook tells the **Views** module which function in our code executes our view: `d7vc_list_all_nodes()`. The first thing it does is create a view object by calling a function provided by the **Views** module. Having instantiated the new object, we then proceed to provide the information it needs, such as the name of the view, its description, and all the information that we would have selected via the Views UI had we used it. Since we're specifying the view options in code, we need to provide the information that is needed by each handler of view functionality.

The net effect of the code is that when we have cleared the cache and enabled our module, Drupal then includes it in its list of modules to poll during events. When we navigate to the Views admin page, an event occurs in which any module wishing to include a view in the list on the admin screen does so, including ours. One of the things our module does is define a path for the page display of our view, which is then used to establish a callback. When that path, `list-all-nodes`, is requested, it results in the function in our module being invoked, which in turn provides all the information necessary for our view to be rendered and presented.

There's more...

The details of the code provided to each handler are outside the scope of this book, but you don't really need to understand it all in order to use it. Until you do, you can create a view using the admin Views UI, choose to export it (see *Chapter 7*, *But Wait – There's More*), copy the importable code, and paste it in your module in the appropriate function.

Handling a view field

As you may have noticed in the previous code, that you typed or pasted, views makes tremendous use of handlers. What is a handler? It's simply a script that performs a special task on one or more elements. Think of a house being built. The person who comes in to tape, mud, and sand the drywall is a handler.

In views, one type of handler is the field handler, which handles any number of field-related operations, from providing settings options in the field configuration dialog to facilitating the field being retrieved from the database, if it's not part of the primary record, to rendering the data. We're going to create a field handler in this recipe that will add a string to the display of a ZIP code that shows how many nodes have the same ZIP code, and we will add some formatting options to it in the next recipe.

Getting ready

A handler lives inside a module, so we will create one:

1. Create a directory in your custom modules path (probably `sites/all/modules/custom`) for this module. I'll create `sites/all/modules/custom/d7vc_zip`.
2. Open a new text file in your editor and paste the following:

```
; $Id:
name = Zip Code Handler
description = Provides a view handler to format a field as
a zip code
package = D7 Views Cookbook
; Handler
files[] = d7vc_zip_handler_field_zip_code.inc
files[] = d7vc_zip_views.inc

version = "7.x-1.0"
core = "7.x"
```

3. Save the file as `d7vc_zip.info`.
4. Create another text file and paste the following:

```
<?php
/**
 * Implements hook_views_data_alter()
 */
function d7vc_zip_field_views_data_alter(&$data, $field) {
        if (array_key_exists('field_data_field_zip_code',
        $data)) {
```

```
            $data['field_data_field_zip_code']
            ['field_zip_code']['field']['handler'] =
            'd7vc_zip_handler_field_zip_code';
          }
}
```

5. Save the file as `d7vc_zip.views.inc`.
6. Create another text file and paste the following:

```
<?php
/**
 * Implements hook_views_api().
 */
function d7vc_zip_views_api() {
  return array(
    'api' => 3,
    'path' => drupal_get_path('module', 'd7vc_zip'),  );
}
```

7. Save the file as `d7vc_zip.module`.
8. Create another text file and paste the following:

```
<?php
// $Id: $

/**
 * Field handler to format a zip code.
 *
 * @ingroup views_field_handlers
 */
class d7vc_zip_handler_field_zip_code extends
views_handler_field_field {
  function option_definition() {
    $options = parent::option_definition();

    $options['display_zip_totals'] = array(
      'contains' => array(
        'display_zip_totals' => array('default' => FALSE),
      )
    );

    return $options;
  }

  /**
```

```
 * Provide link to the page being visited.
 */
function options_form(&$form, &$form_state) {
  parent::options_form($form, $form_state);
  $form['display_zip_totals'] = array(
    '#title' => t('Display Zip total'),
    '#description' => t('Appends in parenthesis the
    number of nodes containing the same zip code'),
    '#type' => 'checkbox',
    '#default_value' => !empty($this-
    >options['display_zip_totals']),
  );
}

function pre_render(&$values) {
  if (isset($this->view->build_info['summary']) ||
  empty($values)) {
    return parent::pre_render($values);
  }
  if (!empty($values)) {
    foreach ($values as $value) {
  if ($this->view->field['field_zip_code']-
  >options['display_zip_totals'])
  $result = db_query("SELECT count(*) AS recs FROM
  {field_data_field_zip_code} WHERE field_zip_code_value
  = :zip AND entity_type = 'node'", array(':zip' =>
  $value->field_field_zip_code
  [0]['raw']['safe_value']));
      $count = $result->fetchField();
      $value->field_field_zip_code[0]['rendered']
      ['#markup'] .= ' (' . $count . ')';
    }
  }
 }
}
```

9. Save the file as `d7vc_zip_handler_field_zip_code.inc`.
10. Navigate to `admin/build/modules` and enable the new module, which shows as **Zip Code Handler** in the **Drupal 7 Views Cookbook** section.
11. We will test the handler in a quick view. Navigate to `admin/structure/views`.
12. Click the **+ Add new view** link and enter `test zip code` as the **View name**.
13. Check the box for **Description** and enter `Zip code handler test`.
14. Clear the **Create a page** check box and click the **Continue & edit** button.

15. On the **Views** edit page, click the **Add** button in the **Filter Criteria** section, check the checkbox for **Content: Type**, and click the **Add and configure filter criteria** button.
16. In the **Configure filter criterion** configuration window, in the **Content types** select list, select **Home** then click the **Apply** button.
17. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Zip code**, then click the **Add and configure fields** button.
18. Check the box at the bottom of the **Configure field: Content: Zip code** configuration window titled **Display Zip total** and click the **Apply** button.
19. Click the **Save** button and see the result of our custom handler in the live preview:

How it works...

The views field handler is simply a set of functions that provide support for populating and formatting a field for views, much in the way a printer driver does for the operating system. We created a module in which our handler resides, and whenever that field is requested within a view, our handler will be invoked. We also added a display option to the configuration options for our field, which when selected, takes each zip code value to be displayed, determines how many nodes have the same zip code, and appends the parenthesized total to the output.

The three functions, two in the `views.inc` file and one in the module file, are very important. Their result is that our custom handler file will be used for `field_zip_code` instead of the default handler used for entity text fields.

Documentation for views field handlers can be found at `https://api.drupal.org/api/views/handlers%21views_handler_field.inc/group/views_field_handlers/7`.

There's more...

In the next recipe, we will add zip code formatting options to our custom handler.

Styling a view field

In the preceding recipe, we created a module for a custom field handler for a ZIP code and a small test view to see the result. In this recipe, we will add styling options to the handler to offer the user a choice of output styles.

Getting ready

This recipe continues from what was created in the *Handling a view field* recipe. If you have not yet tried that recipe, please do, so that you will have the module and view necessary for this recipe.

Edit one of the **Home** content types (or add a few if you have none). At least two of the entries should have the same ZIP code, and at least one should have a (nine-digit) *ZIP + 4* code with no hyphen (for example, 123456789).

How to do it...

1. Edit the `d7vc_zip_handler_field_zip_code.inc` file and insert the following code in boldface in the `options_form()` function:

```
function options_form(&$form, &$form_state) {
  parent::options_form($form, $form_state);
  $form['display_zip_totals'] = array(
    '#title' => t('Display Zip total'),
    '#description' => t('Appends in parenthesis the
    number of nodes containing the same zip code'),
    '#type' => 'checkbox',
    '#default_value' => !empty($this-
    >options['display_zip_totals']),
  );
  $form['type'] = array(
    '#type' => 'select',
```

```
        '#title' => t('Formatter'),
        '#options' => array(
            t('Zip+4 or Zip'),
            t('Zip'),
            t('Alphanumeric')
        ),
        '#default_value' => $this->options['type'],
      );
    }
```

2. In the function `pre_render`, look around line 50 for the following line:

```
foreach ($values as $value) {
```

3. And insert the following line after it:

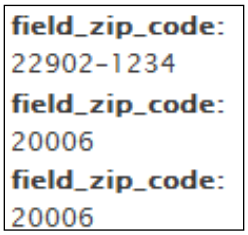
```
$value->field_field_zip_code[0]['rendered']['#markup'] =
$this->_make_zip($value->field_field_zip_code[0]['raw']
['safe_value'], $this->options['type']);
```

4. In the same file, just prior to the final } (around line 57), insert the following code:

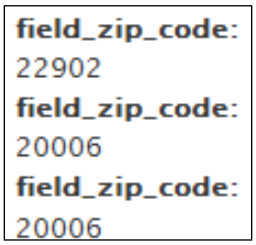
```
function _make_zip($value, $zip_type = 2) {
// remove the hyphen if present
$zip = explode('-', $value);
switch ($zip_type) {
   case 1: // zip+4 or zip depending on size
     if (is_numeric($zip[0])) {
        $value = $zip[0];
        if (sizeof($zip) > 1) {
         if (is_numeric($zip[1])) {
          $value .= '-' . $zip[1];
         }
        }
        else {
         if (strlen($zip[0]) > 5) {
          $value = substr($zip[0],0,5);
          if (strlen($zip[0]) == 9) {
           $value .= '-' . substr($zip[0],5,4);
          }
         }
        }
     }
     break;
   case 2: // zip (trim to 5)
     if (is_numeric($zip[0]) && strlen($zip[0] >= 5)) {
        $value = substr($zip[0],0,5);
     }
```

```
      break;
    case 3: // just change case
      $value = strtoupper($value);
      break;
  }
  return $value;
}
```

5. Save the file.
6. Navigate to `admin/structure/views` and edit the test view.
7. Click the link in the **Fields** box for **Fields: field_zip_code**, and at the bottom of the configuration box, select **Zip+4 or Zip** from **Formatter**, clear the **Display Zip total** check box, and click the **Apply** button, resulting in the output shown in the following image:



8. Click the same field link once again, and this time select **Zip** from **Formatter** and click the **Apply** button, resulting in the output shown in the following image:



How it works...

The formatter select box in the field configuration screen is merely a form field that passes along the selected value. The code we put in place created three options that are then fulfilled by a formatting function.

The options for the formatting use an associated array with key values 1 to 3. Having no key values would result in the first option having a key of 0, and this would result in that option being ignored, when selected.

The first option displays the ZIP code as either ZIP + 4 (12345-6789) or ZIP (12345), depending on what ZIP codes the reader used when creating content. If the ZIP code is numeric and either of the format *xxxxxxxxx* or *xxxxx-xxxx*, it will be displayed as ZIP + 4, or if 5 digits, as a regular zip code.

The second option is to always display in ZIP format, so that a ZIP + 4 code will be truncated to five digits.

The third option forces the value to uppercase, which would be good for alphanumeric post codes.

Fine-tuning the query

The Views UI is a powerful query builder tool, in addition to its other functionality, but sometimes the SQL query generated by it is not precisely what you want it to be. In this recipe, we will make a change to the underlying SQL of a Views query.

Getting ready

This recipe continues from the test view and code created in the *Handling a view field* section. If you have not yet tried that recipe, please do, so that you will have the module and view necessary for this recipe.

How to do it...

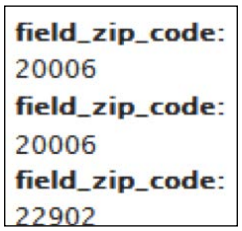
1. Edit the module file `d7vc_zip.module` and add the following code:

```
/**
 * Implements hook_views_query_alter
 */
function d7vc_zip_views_query_alter(&$view, &$query) {
  if ($view->name == 'test_zip_code') {
    // Add the sort field to the query
    $query->fields ['field_data_field_zip_code_
    field_zip_code_value'] = array(
      'table' => 'field_data_field_zip_code',
      'field' => 'field_zip_code_value',
      'alias' => 'field_data_field_zip_code_
      field_zip_code_value');
```

```
    // Tell Views how to link the node table to the table
    // containing the zip code value
    $join = new views_join;
    $join->table = 'field_data_field_zip_code';
    $join->left_table = 'node';
    $join->left_field = 'nid';
    $join->field = 'entity_id';
    $join->extra = array(
      0 => array(
        'field' => 'entity_type',
        'value' => 'node'
      ),
      1 => array(
        'field' => 'deleted',
        'value' => 0,
        'numeric' => true),
      );
    $join->type = "LEFT";
    $join->extra_type = 'AND';
    $join->adjusted = 'TRUE';
    // Tell the view about the additional table
    $view->query->table_queue['field_data_field_zip_code']
    = array(
      'table' => 'field_data_field_zip_code',
      'num' => 1,
      'alias' => 'field_data_field_zip_code',
      'join' => $join,
      'relationship' => 'node',
    );
    // Add the table to the view's table list
    $view->query->tables['node']
     ['field_data_field_zip_code'] = array(
      'count' => 1,
      'alias' => 'field_data_field_zip_code',
    );
    // Now that the field can be referenced, define the new
    // sort, overwriting the original
    $query->orderby[0]['field'] =
    'field_data_field_zip_code_field_zip_code_value';
    $query->orderby[0]['direction'] = 'ASC';
  }
}
```

2. Save the file.

3. Edit the test view to see the preview:



How it works...

The underlying query in a view is made available as a data structure at various points in the view rendering process. We made use of a hook into the process, and altered the query structure to change the sort field.

In order to do this, we had to take a series of steps to make the query and the view aware of the field we used. The reason for this is that in Drupal 7, the data for each custom field is kept in a database table unique to that field. When the field is selected on the screen in the Views UI, the code necessary to join the underlying node table to the field's table is automatically added, but when hooking the query in code and asking it to add a field, we also have to tell it how to do that.

The steps we took in the code (see the comments) were:

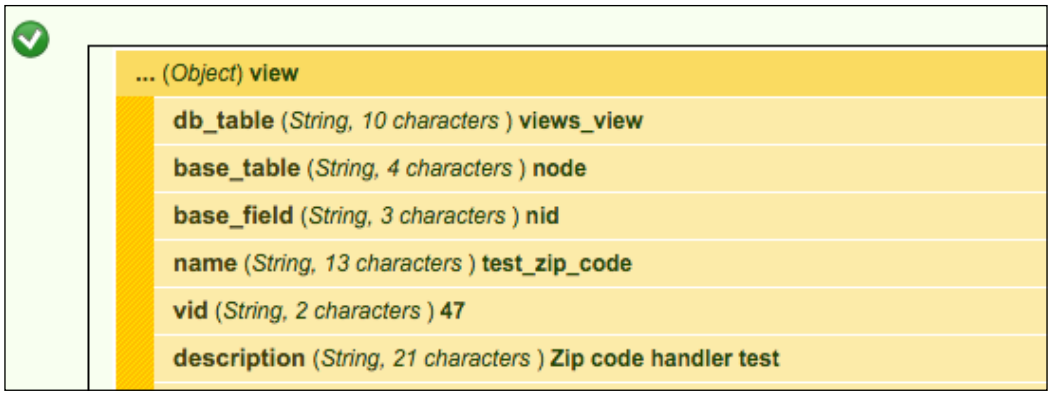
- Add the sort field to the query
- Tell views how to link the node table to the table containing the ZIP code value
- Tell the view about the additional table
- Add the table to the view's table list

None of this is intuitive. And, honestly, finding any straightforward documentation on the Views API, Views object, and query object is problematic, though pieces exist in various places. So, even finding the correct fields can take hours, at times. A good place to start is exploring the Views and query objects, using either the Devel module or native PHP.

If you use the Devel module, you can add a line to your code, such as the following:

```
dpm($view);
```

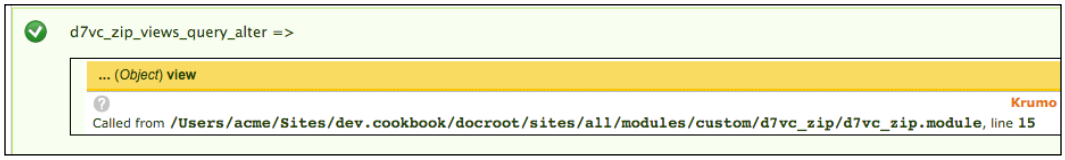
Which adds an expandable navigation bar to your display (you must be logged in with the proper permissions), as shown in the following image:



You can also use the following:

```
dpm($view, __function__);
```

Which will also produce the expandable navigation bar, but will also provide the information about from which function and line number the call was made.



If, for some reason, you are unable to use the Devel module (in some cases it can wreak havoc on the display for some pages), you can do it the PHP way. The following snippet will produce a structured listing of the object, just keep in mind that it is very large:

```
print '<pre>' . print_r($view, 1) . '</pre>';
```

Why would you want to do this instead of simply using the Views UI? The Views UI is good for users to manipulate a view, given the applicable permissions, but those changes would be persisted until the view was changed again. Also, you may want aspects of the underlying query to be handled dynamically, determined based on the data being queried, the user doing the query, or other factors.

7
But Wait – There's More

In this chapter, we will cover some basic administration as well as settings that can be very useful:

- Exporting a view
- Importing a view
- Bulk exporting views
- Cloning a view
- "Featurizing" a view
- Using image styles
- Making a view perkier
- Revealing a recipe's ingredients

Introduction

When working with multiple views and/or sites, it can become quite tedious creating every view from scratch. There are, however, ways to avoid this predicament. We will cover how to copy views, whether to another view or another site.

We will also look at how to take the view created in the UI and make it resident in code, so that it can be under version control.

Keeping in mind that the lifeblood of a view is the underlying SQL, when considering page and site performance, that SQL could be a critical factor. There are ways to make a view perkier and better performing, and we will look at some simple things to try as a start.

Exporting a view

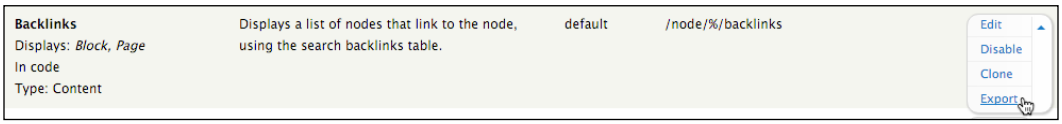
Exporting a single view can be done easily, but might not be something you've tried. Let's export one, now, that came bundled with the **Views** module.

Getting ready

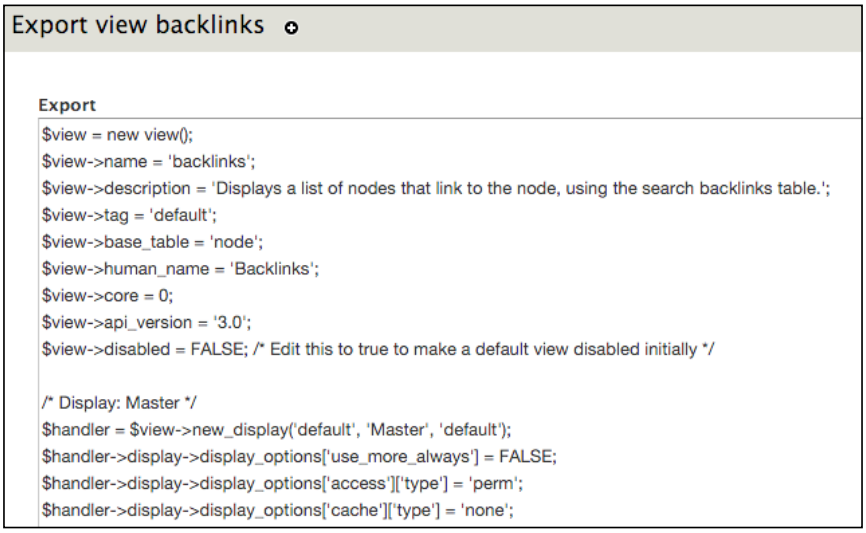
Navigate to the views list (`admin/structure/views`) and enable the **Backlinks** view, if not already enabled.

How to do it...

1. Navigate to the views list (`admin/structure/views`).
2. Click the down arrow next to the **Edit** link for the **Backlinks** view and click the **Export** link:



3. A text area box populated with the PHP code necessary to recreate the view will appear. The first part of that code is shown in the following image. Simply copy all the text in the textbox, paste it into a file, and save it. The file name and extension are of no importance... just be sure to remember what you name it and where you put it:



How it works...

The export tool generates the code necessary to recreate the view, when imported.

Importing a view

The view import will typically be used when copying a view from one site to another. When working within a site, the clone function will typically be more useful (see the following recipe). Importing is typically quick and painless. Let's try it.

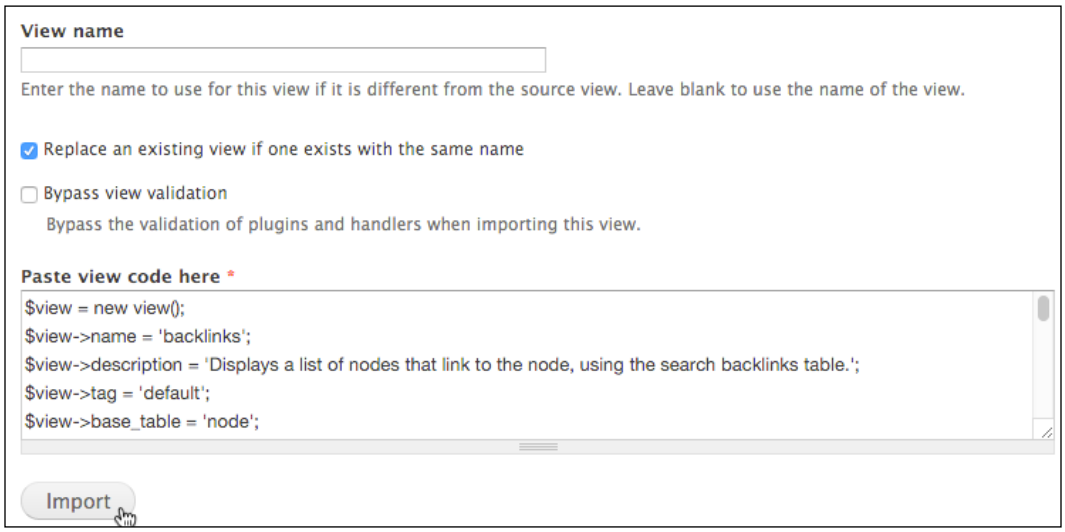
Getting ready

We'll be using the exported view from the previous recipe, so take a minute and follow that recipe now, if you haven't already.

How to do it...

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+ Import** link at the top of the list. If the link is not present, either the PHP filter is not enabled or you do not have the permission setting to use it.
3. Will you be replacing the existing view or creating a new one? If the view does not yet exist on the site where you are importing it to, or if it already exists, but you want to create a newer copy of it and leave the original untouched, enter a name for the view in the **View name** textbox in the import dialog, making sure the name is different than the existing view. Skip to step 6.
4. If you are porting a newer version of an existing view, you might want to overwrite the older existing view rather than creating a new one. If so, in the import dialog leave the **View name** textbox blank.
5. Check the checkbox labeled **Replace an existing view if one exists with the same name**.

6. Paste the code to be imported into the **Paste view code here** textbox and click the **Import** button.



7. In the **Views** UI, remember to save your view!

How it works...

The import function takes the code that was exported and executes the PHP instructions, which is equivalent to having created a new view via the UI and manually selecting each of the view settings.

You might be recreating a view that uses add-on handlers that are not yet present in the environment in which you are doing the import. If the import process complains about this and you know that the handlers will be available when needed, you can check the box labeled **Bypass view validation** before importing.

There's more...

Though simple, exporting and importing individual views is not the most efficient method when you have many in one site that need to be migrated. In that situation, it is much better to perform a bulk export, and that's the next recipe.

Bulk exporting views

Moving views from one site to another can seem a problem, since the views are kept in several database tables. You can export each view using the previous recipe and then import each at the other site, but if you have many views that can quickly become tedious. Being able to export several views at one time would be much better.

Thankfully, the Chaos Tools Suite, which should be present since it is required by views, comes with a **Bulk Export** module.

Getting ready

The **Bulk Export** module needs to be enabled in order to use this recipe. Enable it either by using the admin page at `admin/modules` and checking the checkbox in the **CHAOS TOOLS SUITE** section for **Bulk Export**, then clicking the **Save configuration** button, or by using Drush and issuing this command:

```
drush en bulk_export -y
```

How to do it...

1. Navigate to the **Bulk Exporter** page (`admin/structure/bulk-export`).
2. Select the view(s) to be exported.

Checking the **VIEWS_VIEW** checkbox will select all the views.

VIEWS	TAG	DESCRIPTION
blog_posts		Subsets of blog posts
chameleon		Content type displays
comments_recent	default	Contains a block and a page to list recent comments; the block will automatically link to the page, which displays the comment body as well as a link to the node.
content_topics		Bulleted list of topics
content_topics_facebook		Bulleted list of topics with Facebook buttons
country_countdown		View title and Themed country rows
course_list		A list of available courses
custom_node_links		An index of site content, with each entry being a custom link
department_course_list		Department information and available courses
employee_extensions		Employee information and extensions
frontpage	default	Emulates the default Drupal front page; you may set the default home page path to this view to make it your front page.
glossary	default	A list of all content, by letter.

3. Scroll down to below the file list and enter a name for the module in which you will use the exported code, and then click the **Export** button.

4. The **Bulk Export** module will give you three windows of code, as shown in the following screenshots. The first you will save as `my_module.info`, the second as `my_module.module`, and the third as `my_module.views_default.inc` (replacing each occurrence of `my_module` with the name you provide for the module in step 4). The three files should be saved in a folder, typically with the same name as the module, and that folder should be located in the same directory used for custom modules on your site, probably `sites/all/modules` or `sites/all/modules/custom`.

Place this in test.info

```
name = test export module
description = Export objects from CTools
dependencies[] = views
package = Chaos tool suite
core = 7.x
```

Place this in test.module

```
/**
 * @file
 * Bulk export of objects generated by Bulk export module.
 */

/**
 * Implements hook_views_api().
 */
function test_views_api($module, $api) {
  if ($module == 'views' && $api == 'views_default') {
    return array('version' => 2);
  }
}
```

Place this in test.views_default.inc

```
<?php

/**
 * @file
 * Bulk export of views_default objects generated by Bulk export module.
 */

/**
 * Implements hook_views_default_views().
 */
function test_views_default_views() {
  $views = array();
```

5. After saving the files, clear the cache (`admin/config/development/performance`).
6. Enable the new module, and your views will be found in the **Views** UI list.

How it works...

There are two files needed to create a Drupal 7 module, the `.info` file, which describes the module, and the `.module` file, which typically contains the hooks implemented by the module, but can be empty. The **Views Exporter** module creates the content for these files, as well as content meant for a third file, the code for creating a programmatic view.

The reason you need to provide a module name is that this name is used in function declarations and calls within the three files, so the export tool needs to know the name in order to generate these instructions.

There's more...

You will notice when you enable your new module in an environment that did not previously have its views, they will not have a **Delete** action in the views list (`admin/structure/views`), but a **Disable** action instead. Since the view is in code and not in the database, it cannot be deleted—views cannot delete code but the view can be disabled, even when the module itself is enabled.

There could come a time when you want the view to be resident in the database again instead of in code. The easiest way to do this is to clone the view, which, once saved will be a database-resident copy.

Cloning a view

Sometimes you will need a view that is similar to an existing one. Creating a new view and making each setting within it step by step would be tedious. Luckily, you can simply clone the original view instead, and then make whatever changes you need to it.

How to do it...

1. Navigate to the views list (`admin/structure/views`).
2. Select the view that you wish to clone and click its **Clone** link by clicking the down-arrow on the **Operations** dropdown.
3. Choose a new name for the view and click the **Next** button.
4. When the UI appears, be sure to save the view... it is not permanent until you do so.

"Featurizing" a view

You've seen throughout the book how views are created by using the **Views** user interface. The downside of creating them that way is that they then live in the database without a way to version them. There is a contributed module available for Drupal that facilitates saving database-resident structures to code so that they can be versioned. The module is called **Features**, and one of the structures that can be saved to code using it is a view. Let's do that.

Getting ready

Navigate to `admin/structure` and see if **Features** is in the list of structures. If not, download and enable it either from the Drupal site (`http://drupal.org/project/features`), or with Drush:

```
drush dl features
drush en features -y
```

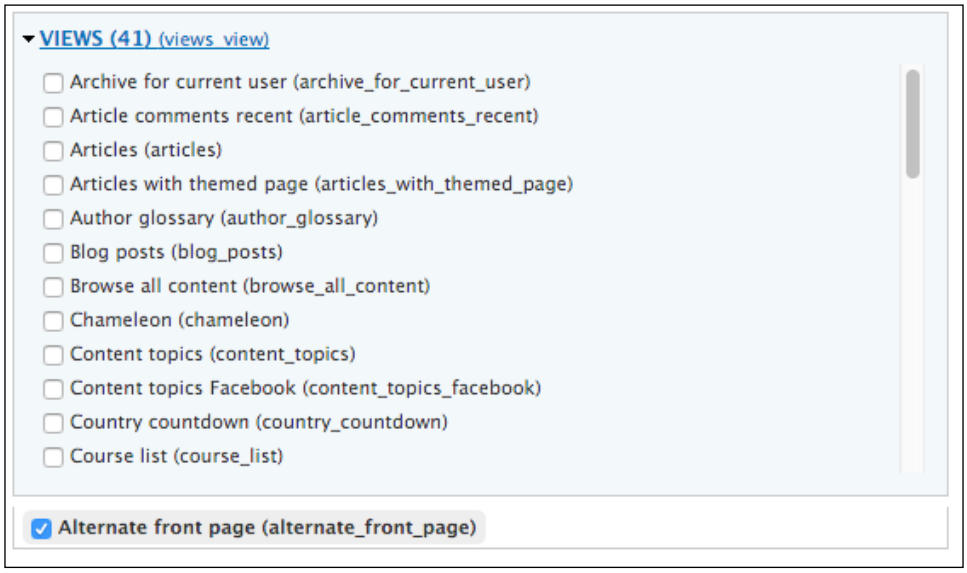
How to do it...

1. Navigate to the **Features** create page, `admin/structure/features/create`.

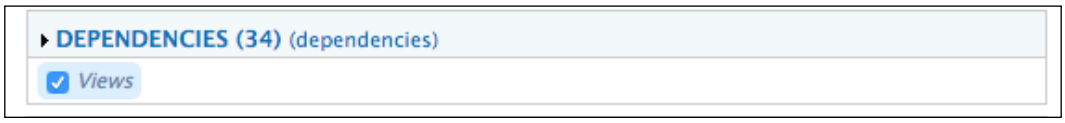
There is a case to be made for each feature being a single view as well as for having one feature containing all views. Here, we will be creating a feature that contains a single view.

2. Click the link for **VIEWS** and note the name of a view to save as a feature. I'll be using **Alternate front page**:

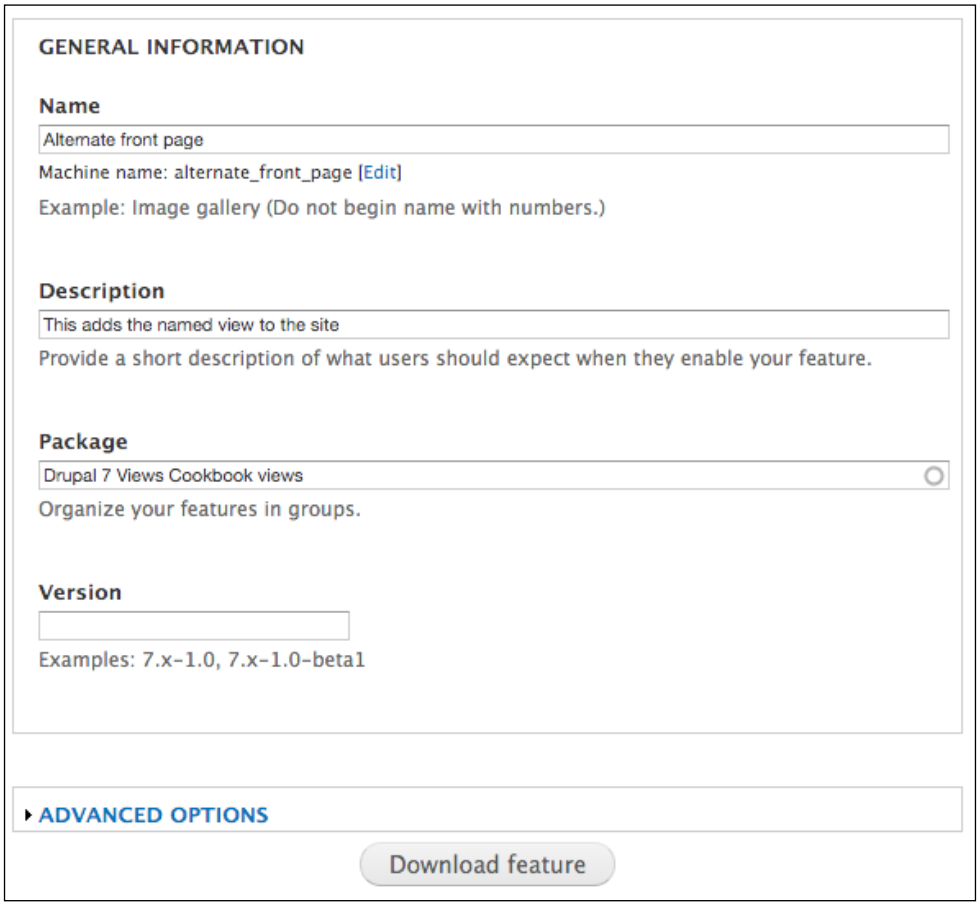
3. When you check the checkbox for the view, the selected view will be removed from the list of views and will appear below the list as an item in the feature.



4. Also, the **Views** module is added to the feature as a dependency, since the view requires it in order to be processed.



5. Enter a name for the feature in the **Name** textbox. Since I'm creating a feature consisting of a single view, I'll give it the same name as the view.
6. In the **Description** textbox, we'll describe what the feature is meant to provide: `This adds the named view to the site`.
7. When we later navigate to the **Features** list, it will be more organized if features are grouped together in a meaningful way, so in the **Package** textbox let's enter `Drupal 7 Views Cookbook views`.



8. At this point, we'll click the **Download feature** button. There are a couple of things I need to point out. The reason the button is not labeled **Save** is because when we click it, the feature will not be saved to the database like a view would be. Instead, the feature will be converted into a module, tar'd, and we will receive a download dialog asking where to save it.
9. At this point, you have saved your view as a feature and are ready to move it to another site (typically to its `sites/all/modules/contrib/features` folder), decompress it, and enable it, or install it on your current site and commit it to version control. Using the **Features** module is described in detail at `http://drupal.org/node/580026`.

Using image styles

There are many different venues in a website for images displayed from the same source, but with a different appearance for each. In Drupal 6, this was achieved via a contributed module, **ImageCache**. That module was brought into the core in Drupal 7, and we will use that functionality to display images in three different sizes.

Getting ready

1. We're going to use the **Gallery** content type. If you have not already added it, you will find the details in *Appendix B*, *Bundles*.
2. You will need four pieces of **Gallery** content.
3. There will be three image styles used, **Exhibit**, **Exhibit teaser**, and **Exhibit block**, the creation details of which are in *Appendix B*, *Bundles*.

How to do it...

1. Navigate to the views list (`admin/structure/views`).
2. Click the **+ Add new view** link, enter **Image styles** as the **View name**, check the **Description** box and enter `Use image styles with view displays` as the **View description**, then change **All** to **Gallery** in **Show content of type**.
3. In the **Create a page** section, change **teasers** to **fields** for the **Display format** and change **Items to display** to **1**.
4. Check the box to **Create a block**, change **titles (linked)** to **fields** for **Display format**, change the **Items per page** to **1**, and click the **Continue & edit** button.

Edit the page display:

1. Click the **Add** button in the **Filters Criteria** section, check the checkbox next to **Content: Exhibit image (field_exhibit_image:fid)**, and click the **Apply (all displays)** button.
2. In the configuration window, for **Content: Exhibit image (field_exhibit_image:fid)** select **Is not empty (NOT NULL)** from the select box and click the **Apply (all displays)** button.
3. Click the **Add** button in the **Fields** section, change **All displays** to **This page (override)**, check the checkbox for **Content: Body**, then click the **Apply (this display)** button.
4. In the **Content: Body** configuration window, uncheck the **Create a label** checkbox and click the **Apply (all displays)** button.
5. Click the **Add** button in the **Fields** section, check the checkbox for **Content: Exhibit image**, then click the **Apply (this display)** button.
6. In the **Content: Exhibit image** configuration box, change **All displays** to **This page (override)**, uncheck the **Create a label** textbox, select **Exhibit** from the **Image style** select box, and click the **Apply (this display)** button.
7. Click the **Full** link next to **Use pager:** in the **Pager** section, select **This page (override)**, change the selection to **Display a specified number of items**, and click the **Apply (this display)** button.
8. Click the **Apply (this display)** button.

Edit the block display:

1. Click the **Block** button in the displays list at the top of the page.
2. Click the **Add** button in the **Fields** section, check the checkbox next to **Content: Exhibit image** and click the **Add (all displays)** button.
3. In the **Content: Exhibit image** configuration window, select **This block (override)** from the select box, uncheck the **Create a label** checkbox, select **Exhibit block** from the **Image style** select box, select **Content** from the **Link image to** select box, and click the **Apply (this display)** button.
4. Click **1 item** in the **Pager** pane, select **This block (override)** from the select box, change **Offset** to **1**, then click the **Apply (this display)** button.

Create the attachment display:

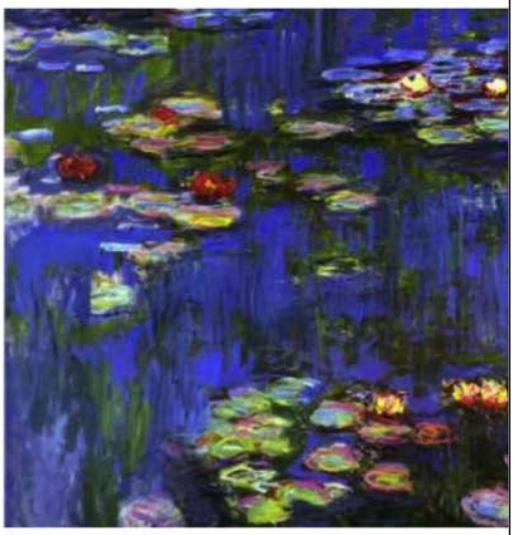
1. Click the **+Add** button in the **Displays** section at the top of the page and click the **Attachment** link.
2. In the **Attachment Settings** section, click **Before** next to **Attachment position**, click the radio button for **After**, and click the **Apply** button.
3. Also in the **Attachment Settings** section, click **Not defined** next to **Attach to**, check the checkbox for **Page**, and click the **Apply** button.
4. Click the **Content: Exhibit image** link in the **Fields** section, select **This attachment (override)** from the select box, uncheck the **Create a label** textbox, select **Exhibit teaser** from the **Image style** select box, and click the **Apply (this display)** button.
5. Click **10 items** in the **Pager** pane, select **This attachment (override)** from the select box, change **Items to display** to **2** and **Offset** to **2**, then click the **Apply (this display)** button.
6. Click the **Content: Title** link in the **Fields** section, ensure **This attachment (override)** is selected in the select box, and click the **Remove** button.
7. Click the **Save** button.
8. Navigate to `admin/structure/block`, find **View: Image styles** in the **Disabled** list, select a sidebar from the **Region** select box, and click the **Save blocks** button.
9. Scroll up to the same entry, which is now in the section for the sidebar you selected, click its **configure** link, under **Show block on specific pages** select **Only the listed pages** and enter `image-styles` in the textbox, then click the **Save block** button.
10. Navigate to `image-styles` to see the view as shown in the following screenshot:

Image styles

Image styles

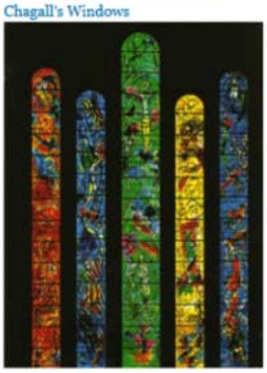
Water Lilies

Oil on canvas by French Impressionist Claude Monet.



Mona Lisa

Chagall's Windows



How it works...

When an image style is created, that style is made available for any image, including those used in views. We created three image styles of different sizes, and used each style in a different display in the same view.

We also set offsets in the paging section of each display. The page display chose the newest image, the block had an offset of 1 so chose the second newest image, and the attachment display with its offset of two ended up choosing the third and fourth newest images.

With the image style feature, there is no need to upload multiple images simply to have different sizes. Just make sure to upload the largest size you will use, because scaling down in size is fine, but scaling up in size will yield unsatisfactory results, because it cannot generate additional image pixels that don't exist in order to form a larger image.

Making a view perkier

A view is an incredible tool for presenting content, which is the main idea with a content management system. Whatever the site, Drupal or not, it had better deliver content to the visitor in under 2 seconds, or that visitor might *bounce* to another site before the page finishes loading. Also, whatever the site, there are three resources the system draws on in accomplishing all the work necessary to compile, render, and deliver the web page: memory, storage (disk), and horsepower (the processor).

Certain activities tax these resources more than others, and one that can quickly put on the brakes is SQL, which is what views depends upon. There are many ways to optimize a site for better performance, and even a couple that are easy to enable and can help make your views perkier.

How to do it...

1. Navigate to `admin/structure/views`, select a view, and click the **Edit** link.
2. Click the **Advanced** link to expand it.
3. In the **Other** section, click **No next** to **Use AJAX**, select **Yes** and click **Apply (all displays)**.
4. Click **None** next to **Caching**, select **Time-based**, click **Apply (all displays)**, select a time in which you would not expect the content to change and use it for both fields, then click **Apply (all displays)**.

How it works...

The usual default behavior for a web page is that when you click a pager link to receive more information, the page reloads with that information. If there are items on the page besides the content that need to change as result of requesting more information, then this default behavior might be the way to go, but if not, loading the subsequent information using AJAX could be. It can be more efficient, and it certainly leaves the rest of the page visible while the new content is being retrieved, which can make it seem faster for the visitor, whether it is or not. Turning this on will cause Drupal to process pager requests using AJAX instead of page loads.

Caching is taking something that had to be constructed, whether an intermediate object such as the results of a query or a final one such as a web page, and storing it so that subsequent requests for the same thing can be fulfilled from what was stored rather than constructed all over again. Views, both the underlying query and the rendered output, can be cached. Having enabled caching for your view, it will not be reconstructed every time it is requested.

There are contributed modules that provide more fine-tuned caching mechanisms for views, such as the views content cache (`https://www.drupal.org/project/views_content_cache`), which will expire the cache whenever the underlying data has changed.

Revealing a recipe's ingredients

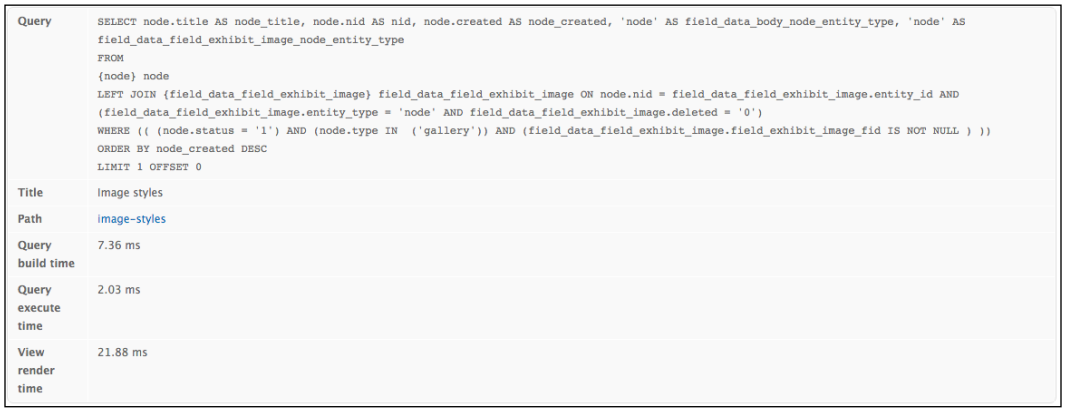
Sometimes when working on a view you might think, "Gee, I wish I knew what it was doing." It could be that the data isn't being retrieved the way you expect, or that the view is running slowly, or both. Wouldn't it be helpful for you to know what the underlying queries look like and how long they are taking to run? Well, you can! Let's do it.

The following settings should only be used in a development environment.

How to do it...

1. Navigate to `admin/structure/views`.
2. Click the **Settings** tab.
3. Check the checkboxes in the **Live Preview Settings** box for **Show the SQL query** and for **Show performance statistics**.

4. Click the **Save configuration** button.
5. Navigate to `admin/structure/views`, select a view (I'll use the **Image styles** view) and click its **Edit** link.
6. The preview should be prefaced with something similar to the following image:



Query	`SELECT node.title AS node_title, node.nid AS nid, node.created AS node_created, 'node' AS field_data_body_node_entity_type, 'node' AS field_data_field_exhibit_image_node_entity_type FROM {node} node LEFT JOIN {field_data_field_exhibit_image} field_data_field_exhibit_image ON node.nid = field_data_field_exhibit_image.entity_id AND (field_data_field_exhibit_image.entity_type = 'node' AND field_data_field_exhibit_image.deleted = '0') WHERE (((node.status = '1') AND (node.type IN ('gallery')) AND (field_data_field_exhibit_image.field_exhibit_image_fid IS NOT NULL))) ORDER BY node_created DESC LIMIT 1 OFFSET 0`
Title	Image styles
Path	image-styles
Query build time	7.36 ms
Query execute time	2.03 ms
View render time	21.88 ms

There is another setting, on the **Advanced** tab (the ones we enabled were on the **Basic** tab) that enables displaying query and performance data when using the **Devel** module. This is worth enabling, and the **Devel** module is well worth using `https://www.drupal.org/project/devel`.

A
Installing Views

In this appendix, we will cover the installation of the **Views** module.

With Drupal 7, there are three options for installing modules:

- Manually, using FTP, as with Drupal 6
- Using the Drupal installer
- Using the `drush` utility

You will find instructions in this appendix, for each of the three. After installation, you should be able to access the admin UI at `admin/structure/views`.

Drupal installer

The easiest way is to simply provide the Drupal installer with a URL for the module, and let the installer do all the work.

Many people continue to have trouble getting the installer to function as designed, including me. If it seems to simply *sit there* without downloading the module(s), switch to one of the other methods.

So, where do you obtain the URL? At the Drupal.org project site. Every contributed add-on module for Drupal has a project page. There you will find a description of the module, and links to its support page, download package, and (sometimes) documentation. These pages are found at `http://drupal.org/project/module_name` where `module_name` is the desired module. In our case, we will navigate to `http://drupal.org/project/Views`.

You will often find that there is more than one version of the module available. You will want the URL of the download link for the recommended version for the appropriate Drupal release; 7.x for our use.

Take care using versions that have a suffix of alpha, beta or dev. Alpha means that the module is raw and just starting community testing, and that at this point any of the features of it are subject to change. Beta is further along, and while the features are probably *frozen*, there can still be many bugs. Dev is a development snapshot, and can go from working to not working on any or all features from snapshot to snapshot. None of these are recommended for use in a production environment. It is best to use a standard release, such as 7x-3.13, with no suffix.

Perform the following steps to download and install **Views**:

1. Navigate to the **Views** project page at `http://drupal.org/project/views` and scroll down to the **Downloads** section:

Version	Download	Date
7.x-3.13	tar.gz (1.57 MB) | zip (1.81 MB)	2015-Nov-06
Other releases		
Version	**Download**	**Date**
6.x-3.2	tar.gz (1.1 MB) | zip (1.27 MB)	2015-Feb-11
6.x-2.18	tar.gz (1.21 MB) | zip (1.36 MB)	2015-Feb-11
Development releases		
Version	**Download**	**Date**
7.x-3.x-dev	tar.gz (1.58 MB) | zip (1.82 MB)	2016-Apr-28
6.x-3.x-dev	tar.gz (1.1 MB) | zip (1.27 MB)	2015-Oct-23

2. We can see that at this time, the recommended release of **Views** for Drupal 7.x is 7.x-3.13. We will install it. On a PC, right-click on the **Download** link beside that release and copy the link location. Choose the version you want, `.zip` or `tar.gz`. On a Mac, control-click over the link and choose the appropriate Finder command to copy it.
3. Navigate to the modules install page (`admin/modules/install`) and paste the URL you copied into the **Install from a URL** text box, and click the **Install** button.
4. When the installation has completed, click the link to activate the new module. On the module page, scroll down to the **Views** module (probably at or near the bottom of the page) and check the boxes for **Views**, **Views exporter**, and **Views UI**, then click the **Save configuration** button.

Drush

Drush is a command-line utility that is very useful for performing Drupal administrative functions quickly. Typically, only system administrators have command-line access to the server and to this utility, because it has total access to alter the site:

1. If you do not have command-line access to the server, or if `drush` is not installed (enter `drush` from the command line to check), move on to the final install option.
2. Enter the following commands, one at a time, at the command line:

```
drush dl views
drush en views
drush cc all
```

Manual installation

Perform the following steps to install **Views** manually:

1. In the row that reflects the version that you want, click the link for either the `tar.gz` or `.zip` file and download the package to your computer.
2. Decompress the file.
3. Use a FTP client to upload the resulting **Views** folder to the directory you are using for contributed modules, which is usually `sites/all/modules` or `sites/all/modules/contributed`.
4. Navigate to the modules admin page, `admin/modules`, and enable **Views** and **Views UI**.

B
Bundles

The following are the details of the entity types and fields that are used within this book, and the instructions for creating them.

Bundle: Country

This is used in the *Teaming two content lists* recipe of *Chapter 4*, *Creating Advanced Views*.

Details

Name	Comments	Author information
Country	None	None

Field	Type	Format
Area (`field_country_area`)	Integer	Text field

Creating bundle: Country...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Country` into the **Name** textbox:
    1. Enter `Country information` in the **Description** textbox.
    2. Click the **Display settings** vertical tab.

3. Uncheck the **Display author and date information** checkbox.
4. Click the **Comment settings** vertical tab.
5. Select **Hidden** from the **Default comment setting for new content** select box.
6. Click the **Save and add fields** button.

4. Click the `delete` link for the **Body** field and confirm the deletion.
5. Under **Add new field**, enter `Area` into the **Label** textbox:

    1. Click the **Edit** link next to `field_area` in the same row and enter `country_area` into the **Field name** textbox.
    2. Select **Decimal** from the **Type of data to store** select box.
    3. Click the **Save** button.
    4. Click the **Save field settings** button.
    5. Click the **Save settings** button.

    Here is some sample data to enter as content, if you want to use the actual top ten data used in the recipe:

Country	Area
Russia	6592735
Canada	3855081
USA	3718691
China	3705386
Brazil	3286470
Australia	2967893
India	1269338
Argentina	1068296
Kazakhstan	1049150
Sudan	967493

Bundle: Course

This is used in the *Understanding relationships* recipe of *Chapter 3*, *Intermediate Custom Views*.

Create this content type *after* the **Department** content type, and *after* installing the references contributed module (*Appendix A*, *Installing Views*).

Details

Name	Comments	Author information
Course	None	None

Field	Type	Format
Course number (`field_course_number`)	Text	Text field

Field	Type	Min	Max
Course credits (`field_course_credits`)	Integer	1	12

Field	Type	Format
Department (`field_department_ref`)	Node reference	Checkboxes

Creating bundle: Course...

1. Navigate to `admin/structure/type`.
2. Click **+Add content type**.
3. Enter `Course` into the **Name** textbox:
    1. Enter `College course` in the **Description** textbox.
    2. Click the **Display settings** vertical tab.
    3. Uncheck the **Display author and date information** checkbox.
    4. Click the **Comment settings** vertical tab.
    5. Select **Hidden** from the **Default comment setting for new content** select box.
    6. Click the **Save and add fields** button.

4. Under **Add new field**, enter `Course number` into the **Label** textbox:
    1. Select **Text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. Enter `36` as the **Maximum** length.
    4. Click the **Save field settings** button.
    5. In the **Course settings** box, enter `12` into the **Size of textfield** textbox.
    6. Click the **Save field settings** button.
5. Under **Add new field**, enter **Course credits** into the **Label** textbox:
    1. Select **Integer** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. Click the **Save field settings** button.
    4. In the **Course settings** box, enter `1` in the **Minimum** textbox.
    5. Enter `6` in the **Maximum** textbox.
    6. Click the **Save settings** button.
6. Under **Add new field**, enter **Department** into the **Label** textbox:
    1. Select **Node reference** from the **Type of data to store** select box.
    2. Select **Checkboxes/radio buttons** from the **Form element to edit the data** select box.
    3. Click the **Save** button.
    4. Check the checkbox for **Department** content type.
    5. Click the **Save field settings** button.
    6. Click the **Save settings** button.
    7. Click the **Save** button.

Bundle: Department

This is used in the *Understanding relationships* recipe of *Chapter 3*, *Intermediate Custom Views*.

Details

Name	Comments	Author information
Department	None	None

Field	Type	Format
Chairman (`field_department_chairman`)	Text	Text field

Field	Type	Format
Phone (`field_department_phone`)	Text	Text field

Field	Type	Format
Degrees (`field_department_degrees`)	Long text	Text area

Creating bundle: Department...

1. Navigate to `admin/structure/type`.
2. Click **+Add content type**.
3. Enter `Department` into the **Name** textbox:
    1. Enter `University departments` in the **Description** textbox.
    2. Enter `Department` in the **Title field label** textbox.
    3. Click the **Display settings** vertical tab.
    4. Uncheck the **Display author and date information** checkbox.
    5. Click the **Comment settings** vertical tab.
    6. Select **Hidden** from the **Default comment setting for new content** select box.
    7. Click the **Save and add fields** button.
4. Under **Add new field**, enter `Chairman` into the **Label** textbox:
    1. Select **Text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. Click the **Save field settings** button.
    4. Click the **Save settings** button.

5. Under **Add new field**, enter `Phone` into the **Label** textbox:
    1. Select **Text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. Click the **Save field settings** button.
    4. Click the **Save settings** button.
6. Under **Add new field**, enter `Degrees` into the **Label** textbox:
    1. Select **Long text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. Click the **Save field settings** button.
    4. Click the **Save settings** button.
    5. Click the **Save** button.

Bundle: Employee

This is used in the *Page, block, and attachment* recipe of *Chapter 4*, *Creating Advanced Views*.

Details

Name	Comments	Author information
Employee	None	None

Field	Type	Format
Department (`field_employee_dept`)	Text	Text field

Field	Type	Format
Position (`field_employee_position`)	Text	Text field

Field	Type	Format
Employee ID (`field_employee_id`)	Text	Text field

Creating bundle: Employee...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Employee` into the **Name** textbox:
    1. Enter `Company employees` in the **Description** textbox.
    2. Enter `Employee name` in the **Title field label** textbox.
    3. Click the **Display settings** vertical tab.
    4. Uncheck the **Display author and date information** checkbox.
    5. Click the **Comment settings** vertical tab.
    6. Select **Hidden** from the **Default comment setting for new content** select box.
    7. Click the **Save and add fields** button.
4. Under **Add existing field**, select **Node reference: field_department** into the **Field to share** textbox:
    1. Select **Check boxes/radio buttons** from the **Form element to edit the data** select box.
    2. Click the **Save** button.
    3. Click the **Save settings** button.
5. Under **Add new field**, enter `Position` into the **Label** textbox:
    1. Click the **Edit** link next to `field_position` and enter `employee_position` into the textbox.
    2. Select **Text** from the **Type of data to store** select box.
    3. Click the **Save** button.
    4. Click the **Save field settings** button.
    5. Click the **Save settings** button.
6. Under **Add new field**, enter `ID` into the **Label** textbox:
    1. Click the **Edit** link next to **field_id** and enter `employee_id` into the textbox.
    2. Select **Text** from the **Type of data to store** select box.
    3. Click the **Save** button.
    4. Click the **Save field settings** button.
    5. Click the **Save settings** button.
7. Click the **delete** link for the **Body** field and confirm the deletion.
8. Click the **Save** button.

Bundle: Extension

This is used in the *Page, block, and attachment* recipe of *Chapter 4*, *Creating Advanced Views*.

Details

Name	Comments	Author information
Extension	None	None

Field	Type	Format
Employee ID (`field_employee_id`)	Text	Text area

Creating bundle: Extension...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Extension` into the **Name** textbox:
    1. Enter `Employee extensions` in the **Description** textbox.
    2. Enter `Extension` in the **Title field label** textbox.
    3. Click the **Display settings** vertical tab.
    4. Uncheck the **Display author and date information** checkbox.
    5. Click the **Comment settings** vertical tab.
    6. Select **Hidden** from the **Default comment setting for new content** select box.
    7. Click the **Save and add fields** button.
4. Under **Add new field**, enter `Employee` into the **Label** textbox:
    1. Select **Node reference** from the **Type of data to store** select box.
    2. Choose **Select list** from the **Form element to edit the data** select box.
    3. Click the **Save** button.
    4. Check the checkbox for **Employee** and click the **Save field settings** button.
    5. Click the **Save settings** button.
5. Click the **delete** link for the **Body** field and confirm the deletion.
6. Click the **Save** button.

Image style: Exhibit

Details

Effect	Width	Height
Scale and crop	480 px	640 px

Creating image style: Exhibit...

1. Navigate to `admin/config/media/image-styles`.
2. Click **+Add style**.
3. Enter `Exhibit` into the **Style name** textbox.
4. Click the **Create new style** button.
5. Select **Scale and crop** in the **EFFECT** select box.
6. Click the **Add** button.
7. Enter `480` into the **Width** textbox.
8. Enter `640` into the **Height** textbox.
9. Click the **Add effect** button.
10. Click the **Update style** button.

Image style: Exhibit_teaser

Details

Effect	Width	Height
Scale and crop	120 px	160 px

Creating image style: Exhibit_teaser...

1. Navigate to `admin/config/media/image-styles`.
2. Click **+Add style**.
3. Enter `Exhibit teaser` into the **Style name** textbox.
4. Click the **Create new style** button.
5. Select **Scale and crop** in the **EFFECT** select box.
6. Click the **Add** button.

7. Enter `240` into the **Width** textbox.
8. Enter `320` into the **Height** textbox.
9. Click the **Add effect** button.
10. Click the **Update style** button.

Image style: Exhibit_block

Details

Effect	Width	Height
Scale and crop	240 px	320 px

Creating image style: Exhibit_block...

1. Navigate to `admin/config/media/image-styles`.
2. Click **+Add style**.
3. Enter `Exhibit block` into the **Style name** textbox.
4. Click the **Create new style** button.
5. Select **Scale and crop** in the **EFFECT** select box.
6. Click the **Add** button.
7. Enter `120` into the **Width** textbox.
8. Enter `160` into the **Height** textbox.
9. Click the **Add effect** button.
10. Click the **Update style** button.

Bundle: Gallery

This is used in the *Changing the front page view* and *Using image styles recipes* of *Chapter 1*, *Modifying Default Views*, and *Chapter 7*, *But Wait – There's More*, respectively.

Details

Name	Comments	Author information
Gallery	None	None

Field	Type	Format
Image exhibit (`field_image_exhibit`)	Image	Image exhibit

Creating bundle: Gallery...

1. From the admin **Structure** menu (`admin/structure`), click **Content types.**
2. Click **+Add content type**.
3. Enter `Gallery` into the **Name** textbox.
4. Enter `Gallery exhibits for use on the alternate front page.` in the **Description** textbox.
5. Click **Display settings**.
6. Clear the **Display author and date information** checkbox.
7. Click **Comment settings**.
8. Select **Hidden** from the **Default comment setting for new content** select box.
9. Click **Menu settings**.
10. Clear the **Main menu** checkbox.
11. Click the **Save and add fields** button.
12. Under **Add new field**, enter `Exhibit image` into the **Label** textbox.
13. Enter `exhibit_image` into the **Field name** textbox.
14. Select **Image** from the **Type of data to store** select box.
15. Click the **Save** button.
16. Click the **Manage display** tab.
17. Click the **CUSTOM DISPLAY SETTINGS** link.
18. Check the **Full content** checkbox.
19. Click the **Save** button.
20. Click the **Full content** button at the top right.
21. Select **<Hidden>** from the **LABEL** (column) select box for the **Exhibit image** field (row).
22. Select **Image** from the **FORMAT** (column) select box for **Exhibit image** (row).
23. Click the **Settings** icon on the far right.
24. Select **Exhibit** from the **Image style** select box.
25. Click the **Save** button.

26. Click the **Teaser** button at the top right.
27. Select **<hidden>** from the **LABEL** (column) select box for the **Exhibit image** field (row).
28. Select **Image** from the **FORMAT** (column) select box for **Exhibit image** (row).
29. Click the **Settings** icon on the far right.
30. Select **Exhibit teaser** from the **Image style** select box.
31. Click the **Save** button.

Bundle: Home

This is used in the following recipes:

- *Using a view content filter* of *Chapter 2*, *Basic Custom Views*
- *Winning that argument* of *Chapter 2*, *Basic Custom Views*
- *Content within content* of *Chapter 3*, *Intermediate Custom Views*
- *A view with multiple personalities* of *Chapter 4*, *Creating Advanced Views*

Details

Name	Comments	Author information
Home	None	None

Field	Type	Format
Zip code (`field_zip_code`)	Text	Text field

Field	Type	Format
Image exhibit (`field_image_exhibit`)	Image	Image exhibit

Creating bundle: Home...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Home` into the **Name** textbox.
4. Enter `Homes for sale` in the **Description** textbox.
5. Enter `MLS ID` in the **Title field label** textbox.

6. Click the **Display settings** vertical tab.
7. Uncheck the **Display author and date information** checkbox.
8. Click the **Comment settings** vertical tab.
9. Select **Hidden** from the **Default comment setting for new content** select box.
10. Click the **Menu settings** vertical tab.
11. Uncheck the **Main menu** checkbox.
12. Click the **Save and add fields** button.
13. Under **Add new field**, enter `Zip code` into the **Label** textbox.
14. Select **Text** from the **Type of data to store** select box.
15. Click the **Save** button.
16. Enter `10` into the **Maximum length** textbox.
17. Click the **Save field settings** button.
18. Click the **Save settings** button.
19. Under **Add existing field**, select **Image: field_image (Image)** from the **Field to share** select box.
20. Enter `Home image` into the **Label** textbox.
21. Drag the field and drop it below the **Zip code** field.
22. Click the **Save** button.
23. Click the **Save settings** button.
24. Under **Add existing field**, select **Decimal: field_product_price (Product price)** from the **Field to share** select box.
25. Enter `House price` into the **Label** textbox.
26. Drag the field and drop it below the **Home image** field.
27. Click the **Save** button.
28. Click the **Save settings** button.
29. Click the **Manage display** tab.
30. Select **Inline** from the **LABEL** (column) select box for the **Zip code** field (row).
31. Select **<Hidden>** from the **LABEL** (column) select box for the **Home image** field (row).
32. Click the settings gear icon in the same row.
33. Select **Medium (220x220)** from the **Image style** select box.
34. Select **<Hidden>** from the **LABEL** (column) select box for the **House price** field (row).

35. Click the **CUSTOM DISPLAY SETTINGS** link.
36. Check the **Full content** checkbox.
37. Click the **Save** button.
38. Click the **Teaser** button at the top of the screen.
39. Select **Inline** from the **LABEL** (column) select box for the **Zip code** field (row).
40. Click **Default** from the **FORMAT** select box.
41. Select **<Hidden>** from the **LABEL** (column) select box for the **House price** field (row).
42. Click **Default** from the **FORMAT** select box.
43. Select **<Hidden>** from the **LABEL** (column) select box for the **Home image** field (row).
44. Select **Image** from the **FORMAT** select box.
45. Click the settings gear icon in the same row.
46. Select **Medium (220x220)** from the **Image style** (column) select box for **Home image** (row).
47. Click the **Save** button.

Bundle: Ingredient

This is used in the *Displaying a table of entity fields* recipe of *Chapter 3*, *Intermediate Custom Views*.

Details

Name	Comments	Author information
Ingredient	None	None

Field	Type	Format
Quantity (`field_ingredient_quantity`)	Decimal	Text field

Field	Type	Format
Measure (`field_ingredient_measure`)	Text	Text field

Creating bundle: Ingredient...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Ingredient` into the **Name** textbox:
    1. Enter `Bulk ingredient for shopping list` in the **Description** textbox.
    2. Click the **Publishing options** vertical tab.
    3. Enter `Name` into the **Title field label** textbox.
    4. Click the **Display settings** vertical tab.
    5. Uncheck the **Display author and date information** checkbox.
    6. Click the **Comment settings** vertical tab.
    7. Select **Hidden** from the **Default comment setting for new content** select box.
    8. Click the **Save and add fields** button.
4. Under **Add new field**, enter `Quantity` into the **Label** textbox:
    1. Select **Text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. In the **Quantity** field settings, click the **Save field settings** button, and then in the subsequent window, the **Save settings** button.
5. Under **Add new field**, enter `Measure` into the **Label** textbox:
    1. Select **Text** from the **Type of data to store** select box.
    2. Click the **Save** button.
    3. In the **Field settings** dialog box, click the **Save field setttings** button, and then the **Save settings** button.
6. Click the **Save** button.

Bundle: Product

This is used in the following recipes:

- *Sortable table with a header and footer* of *Chapter 3*, *Intermediate Custom Views*
- *Creating a random ad block* of *Chapter 2*, *Basic Custom Views*
- *A marketing bundle* of *Chapter 4*, *Creating Advanced Views*

Details

Name	Comments	Author information
Product	None	None

Field	Type	Format
Product image (`field_product_image`)	Image	Image exhibit

Field	Type
Product price (`field_product_price`)	Decimal

Creating bundle: Product...

1. Navigate to `admin/structure/types` and click **+Add content type**.
2. Enter `Product` into the **Name** textbox.
3. Enter `Products for sale` in the **Description** textbox.
4. Click the **Display settings** vertical tab.
5. Uncheck the **Display author and date information** checkbox.
6. Click the **Comment settings** vertical tab.
7. Select **Hidden** from the **Default comment setting for new content** select box.
8. Click the **Save and add fields** button.
9. Under **Add new field**, enter `Product image` into the **Label** textbox.
10. Select **Image** from the **Type of data to store** select box.
11. Click the **Save** button.
12. Click the **Save field settings** button.
13. Click the **Save settings** button.
14. Under **Add new field**, enter `Product price` into the **Label** textbox.
15. Select **Decimal** from the **Type of data to store** select box.
16. Click the **Save** button.
17. Click the **Save field settings** button.
18. Enter `$` in the **Prefix** textbox, or another currency identifier in the **Prefix** or **Suffix** textboxes.

19. Click the **Save settings** button.
20. Click the **Manage display** tab.
21. Select `<hidden>` from the **LABEL** (column) select box for the **Product image** field (row).
22. Click the settings gear icon in the same row.
23. Select **Medium (220x220)** from the **Image style** (column) select box.
24. Select **<Hidden>** from the **LABEL** (column) select box for the **Product price** field (row).
25. Select **Default** from the **FORMAT** (column) select box for **Product price** (row).
26. Click the **CUSTOM DISPLAY SETTINGS** link.
27. Check the **Full content** checkbox.
28. Click the **Teaser** tab at the top of the window.
29. Select **<Hidden>** from the **LABEL** (column) select box for the **Product image** field (row).
30. Select **Image** from the **FORMAT** (column) select box for the same row.
31. Click the settings gear icon in the same row.
32. Select **Medium (220x220)** from the **Image style** (column) select box.
33. Select **<hidden>** from the **LABEL** (column) select box for the **Product price** field (row).
34. Select **Default** from the **FORMAT** (column) select box for **Product price** (row).
35. Rearrange the fields by dragging so that the order is **Product image**, **Product price**, and **Body**.
36. Click the **Save** button.

Bundle: Real Estate flier

This is used in the *Content within content* recipe of *Chapter 3*, *Intermediate Custom Views*.

Details

Name	Comments	Author information
Real Estate flier	None	None

Field	Type	Format
Property (`field_property`)	Node reference	Select list

Creating bundle: Real Estate flier...

Create this content type *after* the home content type, and *after* installing the references contributed module (*Appendix A*, *Installing Views*)

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Real Estate flier` into the **Name** textbox:
    1. Enter `Body text and node references for the week's hot properties` in the textbox.
    2. Click the **Display settings** vertical tab.
    3. Uncheck the **Display author and date information** checkbox.
    4. Click the **Comment settings** vertical tab.
    5. Select **Hidden** from the **Default comment setting for new content** select box.
    6. Click the **Save and add fields** button.
4. Under **Add new field**, enter `Property` into the **Label** textbox:
    1. Select **Node reference** from the **Type of data to store** select box.
    2. Choose **Select** list from the **Form element to edit the data** select box.
    3. Click the **Save** button.
    4. Check the checkbox next to **Home**.
    5. Click the **Save field settings** button.
    6. Select **Unlimited** in the **Number of values** select box.
    7. Click the **Save settings** button.
    8. Click the **Save** button.

Bundle: Sponsor

This is used in the following recipes:

- *Creating a paged block display* of *Chapter 2*, *Basic Custom Views*
- *Creating a dynamic links display block* of *Chapter 2*, *Basic Custom Views*

Details

Name	Comments	Author information
Sponsor	None	None

Field	Type	Format
Sponsor logo (`field_sponsor_logo`)	Image	Thumbnail

Creating bundle: Sponsor...

1. Navigate to `admin/structure/types`.
2. Click **+Add content type**.
3. Enter `Sponsor` into the **Name** textbox.
4. Enter `Event sponsor` in the **Description** textbox.
5. Enter `Sponsor name` in the **Title** textbox.
6. Click the **Display settings** vertical tab.
7. Uncheck the **Display author and date information** checkbox.
8. Click the **Comment settings** vertical tab.
9. Select **Hidden** from the **Default comment setting for new content** select box.
10. Click the **Menu settings** vertical tab.
11. Uncheck the **Main menu** checkbox.
12. Click the **Save and add fields** button.
13. Under **Add new field**, enter `Sponsor logo` in the **Label** textbox.
14. Select **Image** from the **Type of data to store** select box.
15. Click the **Save** button.
16. Click the **Save field settings** button.
17. Click the **Save settings** button.
18. Click the **Manage Display** tab.
19. Click the **Teaser** button.
20. Select **<Hidden>** from the **LABEL** (column) select box for the **Sponsor logo** field (row).
21. Select **Image** from the **FORMAT** (column) select box for the same field (row).

22. Click the settings gear icon for the **Sponsor logo** field (row).
23. Select **Thumbnail (100x100)** from the **Image style** select box.
24. Click the **Update** button.
25. Select **<hidden>** from the **LABEL** (column) select box for the **Body** field (row).
26. Click the **Save** button.

Creating taxonomy tags

Used in the *Related content – adding depth to a term ID* recipe of *Chapter 4*, *Creating Advanced Views*.

Creating taxonomy tags...

1. Navigate to `admin/structure/taxonomy/tags/add`.
2. Enter `Tourist Info` in the **Name** field, click the **Relations** link, ensure that **<root>** is selected in the **Parent terms list** box, and click **Save**.
3. Enter `Dining` in the **Name** field, click the **Relations** link, select **Tourist Info** in the **Parent terms** list box, and click **Save**.
4. Enter `Sightseeing` in the **Name** field, click the **Relations** link, select **Tourist Info** in the **Parent terms** list box, and click **Save**.
5. Enter `Lodging` in the **Name** field, click the **Relations** link, select **Tourist Info** in the **Parent terms** list box, and click **Save**.
6. Enter `Group Tours` in the **Name** field, click the **Relations** link, select **Sightseeing** in the **Parent terms** list box, and click **Save**.

C
Resources

At the FTP site `downloads.theaccidentalcoder.com/d7vc2`, you will find downloadable resources that are available to you should you not want to create the examples in the book step by step.

Available resources

The following resources are available to you as TAR files:

- **Modules**: The `d7vc` and `d7vc_zip` modules
- **Themes**: The `recipes` theme
- **Features**: Features, that when enabled, will load needed entities into your site database:
  - **Field bases**: The field definitions of the custom fields used in the content types
  - **Content types**: The custom content types listed in *Appendix B*, *Bundles*
  - **Views**: The views from Chapters 1 to 5

The file path structure

Your file path structure will very probably differ from mine. It isn't required for the two to be the same. For example, where I might have custom modules in the path `sites/all/modules/custom`, your path might be `sites/all/modules`. Either way, Drupal will find them, and likewise with themes, as long as they are located somewhere below the `modules` and `themes` folders, respectively, within the appropriate site folder.

Should you want to arrange your files in the same way as I have, following is how my paths are structured:

```
sites
    all
        modules
            custom
                d7vc
                d7vc_zip
                d7vc_features
                    content_types
                    fields
                    views
                        ch1
                        ch2
                        ch3
                        ch4
                        ch5
        themes
            recipes
```

Themes

This book uses a subtheme of the Bartik theme, which is included with Drupal. The subtheme is called `recipes` and can be downloaded and installed using the following instructions:

1. Navigate to `downloads.theaccidentalcoder.com/d7vc2/theme/recipes.tar`.
2. Save the file to your `theme` folder, decompress the TAR file:

   ```
   tar -cf recipes.tar
   ```

3. Navigate to `admin/appearance`, enable the theme and make it the default.

Modules

The TAR files for either or both the custom modules used in the book can be downloaded from `downloads.theaccidentalcoder.com/d7vc2/modules`, decompressed and enabled via the admin modules page or with `drush`.

Features

Features are modules that are database-based Drupal structures saved to code. When you place them in the correct place and enable the module, you will have the structure available. For example, saving a feature containing views and enabling it will cause those views to be listed on the **Views** listing page. You can download and install as a feature any of the view recipes from Chapters 1-5, field bases, image styles, and content types used in this book simply by decompressing the TAR file and enabling the **Feature** module the way you would any module, via the admin UI, or the `drush` command.

The content types features are available in TAR files as the individual content types (for example, `d7vc_ct_country-7.x-1.0.tar`) as well as a TAR file that contains all of the content types, `content_types.tar`. The content types are available as individual tar files, organized on the ftp site by chapter (for example, `d7vc_v_ch1_alternate_front_page-7.x-1.0.tar`), TAR files containing all **Views** features for each chapter (for example, `views_ch1.tar`), and a TAR file containing all of the `Views` features, `views.tar`.

Note the whether you create your views step by step in the UI or by enabling a feature, address the following beforehand:

1. If you will create the custom content types manually, do so prior to creating a view that uses it, whether you create the view manually or by enabling a feature.
2. Whether you create the image styles manually or by enabling their feature, do so before creating a view that uses them.
3. If you will create the custom content types using their features, enable the field bases feature first.

Index

U

V

W