Drupal 8 Development Cookbook

What this book will do for you...

- Extend Drupal through contributed or custom modules and themes
- Develop an international website with Drupal’s multilingual tools
- Integrate third-party frontend and backend libraries with Drupal
- Turn Drupal into a web services provider using REST
- Run SimpleTest and PHPUnit to test Drupal
- Understand the plugin system that powers many of Drupal 8’s new APIs to extend its functionality
- Learn the mechanics of the configuration management system to import and export site configuration

Inside the Cookbook...

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

Drupal 8 Development Cookbook

Over 60 hands-on recipes that get you acquainted with Drupal 8’s features and help you harness its power

Matt Glaman

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 3 'Displaying Content through Views'
- A synopsis of the book’s content
- More information on *Drupal 8 Development Cookbook*
About the Author

Matt Glaman is a developer at Commerce Guys. He is an open source developer who has been working with Drupal since 2013. He has also been developing web apps for many years prior to this. Since then, he has contributed to over 60 community projects, including being a co-maintainer of Drupal Commerce. While mostly focusing on Drupal and PHP development, he created https://contribkanban.com, an AngularJS application, to provide Kanban boards for the Drupal community to collaborate with.
Drupal is a content management system used to build websites for small businesses, e-commerce, enterprise systems, and many more. Created by over 4,500 contributors, Drupal 8 provides many new features for Drupal. Whether you are new to Drupal, or an experienced Drupalist, Drupal 8 Development Cookbook contains recipes that help you immerse yourself in what Drupal 8 has to offer.

What this book covers

Chapter 1, Up and Running with Drupal 8, covers the requirements to run Drupal 8, walks you through the installation process, and extends Drupal.

Chapter 2, The Content Authoring Experience, dives into the content management experience in Drupal, including working with the newly bundled CKEditor.

Chapter 3, Displaying Content through Views, explores how to use Views to create different ways to list and display your content in Drupal.

Chapter 4, Extending Drupal, explains how to work with Drupal's Form API to create custom forms to collect data.

Chapter 5, Frontend for the Win, teaches you how to create a theme, work with the new templating system, Twig, and harness Drupal's responsive design features.

Chapter 6, Creating Forms with the Form API, teaches you how to write a module for Drupal, the building blocks of functionalities in Drupal.

Chapter 7, Plug and Play with Plugins, introduces plugins, one of the newest components in Drupal. This chapter walks you through developing the plugin system to work with fields.

Chapter 8, Multilingual and Internationalization, introduces features provided by Drupal 8 to create an internationalized website that supports multiple languages for content and administration.
Chapter 9, Configuration Management – Deploying in Drupal 8, explains the configuration management system, new to Drupal 8, and how to import and export site configurations.

Chapter 10, The Entity API, dives into the Entity API in Drupal, allowing you to create custom configurations and content entities.

Chapter 11, Off the Drupalicon Island, explains how Drupal allows you to embrace the mantra of proudly built elsewhere and includes third-party libraries in your Drupal site.

Chapter 12, Web Services, shows you how to turn your Drupal 8 site into a web services API provider through a RESTful interface.

Chapter 13, The Drupal CLI, explores working with Drupal 8 through two command-line tools created by the Drupal community: Drush and Drupal Console.
This chapter will cover the Views module and how to use a variety of its major features:

- Listing content
- Editing the default admin interfaces
- Creating a block from a View
- Utilizing dynamic arguments
- Adding a relationship in a View
- Providing an Entity Reference result View

**Introduction**

For those who have used Drupal previously, Views is in core for Drupal 8! If you are new to Drupal, Views has been one of the most used contributed projects for Drupal 6 and Drupal 7.

To briefly describe Views, it is a visual query builder, allowing you to pull content from the database and render it in multiple formats. Select administrative areas and content listings provided out of the box by Drupal are all powered by Views. We'll dive into how to use Views to customize the administrative interface, customize ways to display your content, and interact with the entity reference field.
Listing content

Views does one thing, and it does it well: listing content. The power behind the Views module is the amount of configurable power it gives the end user to display content in various forms.

This recipe will cover creating a content listing and linking it in the main menu. We will use the Article content type provided by the standard installation and make an articles landing page.

Getting ready

The Views UI module must be enabled in order to manipulate Views from the user interface. By default this is enabled with the standard installation profile.

How to do it...

Let's list the Views listing content:

1. Visit Structure and then Views. This will bring you to the administrative overview of all the views that have been created:

<table>
<thead>
<tr>
<th>VIEW NAME</th>
<th>DESCRIPTION</th>
<th>TAG</th>
<th>PATH</th>
<th>OPERATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content</td>
<td>Displays: Page. Machine name: content. Find and manage content.</td>
<td>default</td>
<td>/admin/content</td>
<td>Edit</td>
</tr>
<tr>
<td>Files</td>
<td>Displays: Page. Machine name: files. Find and manage files.</td>
<td>default</td>
<td>/admin/content/files/admin/content/files/usage/%</td>
<td>Edit</td>
</tr>
</tbody>
</table>
2. Click on **Add new view** to begin creating a new view.

3. The first step is to provide the **View name** of **Articles**, which will serve as the administrative and (by default) displayed title.

4. Next, we modify the **VIEW SETTINGS**. We want to display **Content** of the type **Articles** and leave the **tagged with** empty. This will force the view to only show content of the **article** content type.

5. Choose to **Create a page**. The **Page title** and **Path** will be auto populated based on the view name and can be modified as desired. For now, leave the display and other settings at their default values.

6. Click on **Save and edit** to continue modifying your new view.

7. In the middle column, under the **Page settings** section we will change the **Menu item** settings. Click on **No menu** to change the default.
8. Select **Normal menu entry**. Provide a menu link title and optional description. Keep the **Parent** set to `<Main Navigation>`.

- **Type**
  - No menu entry
  - Normal menu entry
  - Menu tab
  - Default menu tab

- **Menu link title**
  
  - Articles

- **Description**
  - View all articles
  
  - Shown when hovering over the menu link.

- **Parent**
  - `<Main navigation>`
  
  - The maximum depth for a link and all its children is fixed. Some menu links may not be available as parents if selecting them would exceed this limit.

- **Weight**
  - 0
  
  - In the menu, the heavier links will sink and the lighter links will be positioned nearer the top.

  ![Apply Cancel]

9. Click on **Apply** at the bottom of the form.

10. Once the view is saved you will now see the link in your Drupal site’s main menu.

---

**How it works...**

The first step for creating a view involves selecting the type of data you will be displaying. This is referred to as the base table, which can be any type of entity or data specifically exposed to Views.

Nodes are labeled as Content in Views and you will find throughout Drupal this interchanged terminology.

When creating a Views page we are adding a menu path that can be accessed. It tells Drupal to invoke Views to render the page, which will load the view you create and render it.

There are display *style* and *row* plugins that format the data to be rendered. Our recipe used the **unformatted list** style to wrap each row in a simple div element. We could have changed this to a table for a formatted list. The row display controls how each row is output.
There's more...

Views has been one of the must-use modules since it first debuted, to the point that almost every Drupal 7 site used the module. In the following section we will dive further into Views.

Views in Drupal Core Initiative

Views has been a contributed module up until Drupal 8. In fact, it was one of the most used modules. Although the module is now part of Drupal core it still has many improvements that are needed and are being committed.

Some of these changes will be seen through minor Drupal releases, such as 8.1x and 8.2.x, as development progresses and probably not through patch releases (8.0.10).

Views and displays

When working with Views, you will see some different terminology. One of the key items to grasp is what a display is. A view can contain multiple displays. Each display is of a certain type. Views comes with the following display types:

- **attachment**: This is a display that becomes attached to another display in the same view
- **block**: This allows you to place the view as a block
- **embed**: The display is meant to be embedded programmatically
- **Entity Reference**: This allows Views to provide results for an entity reference field
- **feed**: This display returns an XML based feed and can be attached to another display to render a feed icon
- **page**: This allows you to display the view from a specific route

Each display can have its own configuration, too. However, each display will share the same base table (content, files, etc.). This allows you to take the same data and represent it in different ways.

Format style plugins: style and row

Within Views there are two types of style plugins that represent how your data is displayed – style and row.

- The **style** plugin represents the overall format
- The **row** plugin represents each result row's format

For example, the **grid** style will output multiple div elements with specified classes to create a responsive grid. At the same time, the **table** style creates a tabular output with labels used as table headings.
Row plugins define how to render the row. The default content will render the entity as defined by its selected display mode. If you choose **Fields** you manually choose which fields to include in your view.

Each format style plugin has a corresponding **Twig** file that the theme layer uses. You can define new plugins in custom modules or use contributed modules to access different options.

### Using the Embed display

Each of the available display types has a method to expose itself through the user interface, except for **Embed**. Often, contributed and custom modules use Views to render displays instead of manually writing queries and rendering the output. Drupal 8 provides a special display type to simplify this.

If we were to add an Embed display to the view created in the recipe, we could pass the following render array to output our view programmatically.

```php
$view_render = [
    '#type' => 'view',
    '#name' => 'articles',
    '#display_id' => 'embed_1',
];
```

When rendered, the **#type** key tells Drupal this is a view element. We then point it to our new display `embed_1`. In actuality, the Embed display type has no special functionality, in fact it is a simplistic display plugin. The benefit is that it does not have additional operations conducted for the sake of performance.

### See also

- VDC Initiative:  
  [https://www.drupal.org/community-initiatives/drupal-core/vdc](https://www.drupal.org/community-initiatives/drupal-core/vdc)
- Chapter 7, **Plug and Play with Plugins**, to learn more about plugins

### Editing the default admin interfaces

With the addition of Views in Drupal core, many of the administrative interfaces are powered by Views. This allows customization of default admin interfaces to enhance site management and content authoring experiences.

In Drupal 7 and 6 there was the administrative Views module, which provided a way to override administrative pages with Views. This module is no longer required, as the functionality comes with Drupal core out of the box!
In this recipe we will modify the default content overview form that is used to find and edit content. We will add the ability to filter content by the user who authored it.

How to do it...

1. Visit Structure and then Views. This will bring you to the administrative overview of all existing views.
2. From the Enabled section, select the Edit option for the Content view. This is the view displayed on /admin/content when managing content.
3. In order to filter by the content author, we must add a FILTER CRITERIA to our view, which we will expose the following for users to modify:

   ![FILTER CRITERIA]

   Content: Published status or admin user
   Content: Publishing status (grouped)
   Content: Type (exposed)
   Content: Title (exposed)
   Content: Translation language (exposed)

4. Click on Add to add a new filter. In the search text box type Authored by to search the available options. Select Content: Authored by and click Apply (all displays):

   ![Add filter criteria]

5. Check Expose this filter to visitors, to allow them to change it via checkbox. This will allow users to modify the data for the filter.
6. You may modify the Label and add a Description to improve the usability of the filter option for your use case.
Displaying Content through Views

7. Click on **Apply (all displays)** once more to finish configuring the filter. It will now show up in the list as filter criteria active. You will also see the new filter in the preview below the form.

8. Click on **Save** to commit all changes to the view.

9. View **/admin/content** and you will have your filter. Content editors will be able to search for content authored by a user through autocompleted username searches:

<table>
<thead>
<tr>
<th>Published status</th>
<th>Type</th>
<th>Title</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Any -</td>
<td>- Any -</td>
<td></td>
<td>- Any -</td>
</tr>
</tbody>
</table>

**Authored by**

admin (1)

Enter a comma separated list of user names.

[Filter]  [Reset]

**How it works...**

When a view is created that has a path matching an existing route, it will override it and present itself. That is how the **/admin/content** and other administrative pages are able to be powered by Views.

If you were to disable the Views module you can still manage content and users. The default forms are tables that do not provide filters or other extra features.

Drupal uses the overridden route and uses Views to render the page. From that point on the page is handled like any other Views page would be rendered.

**There's more...**

We will dive into additional features available through Views that can enhance the way you use Views and present them on your Drupal site.

**Exposed versus non-exposed**

Filters allow you to narrow the scope of the data displayed in a view. Filters can either be exposed or not; by default a filter is not exposed. An example would be using the **Content: Publishing status** set to **Yes (published)** to ensure a view always contains published content.

This is an item you would configure for displaying content to site visitors. However, if it were for an administrative display, you may want to expose that filter. This way content editors have the ability to view, easily, what content has not been published yet or has been unpublished.

All filter and sort criteria can be marked as exposed.
Filter identifiers
Exposed filters work by parsing query parameters in the URL. For instance, on the content management form, changing the Type filter will add type=Article amongst others to the current URL.

With this recipe the author filter would show up as uid in the URL. Exposed filters have a Filter identifier option that can change the URL component.

This could be changed to author or some other value to enhance the user experience behind the URL, or mask the Drupal-ness of it.

Overriding routes with Views
Views is able to replace administrative pages with enhanced versions due to the way the route and module system works in Drupal. Modules are executed in order of the module's weight or alphabetical order if weights are the same. Naturally, in the English alphabet, the letter V comes towards the end of the alphabet. That means any route that Views provides will be added towards the end of the route discovery cycle.

If a view is created and it provides a route path, it will override any that exist on that path. There is not a collision checking mechanism (and there was not in Views before merging into Drupal core) that prevents this.

This allows you to easily customize most existing routes. But, beware that you could easily have conflicting routes and Views will normally override the other.

Creating a block from a View
Previous recipes have shown how to create and manipulate a page created by a view. Views provides different display types that can be created, such as a block. In this recipe we will create a block powered by Views. The Views block will list all Tag taxonomy terms that have been added to the Article content type.

Getting ready
This recipe assumes you have installed the standard installation profile and have the default node content types available for use.
Displaying Content through Views

How to do it...

1. Visit Structure and then Views. This will bring you to the administrative overview of all the views that have been created.
2. Click on Add new view to begin creating a new view.
3. The first step is to provide the View name of Tags, which will serve as the administrative and (by default) displayed title.
4. Next, we modify the View settings. We want to display Taxonomy terms of the type Tags. This will make the view default to only displaying taxonomy terms created under the Tags vocabulary.
5. Check the Create a block checkbox.
6. Choose the HTML List option from the Display format choices. Leave the row style as Fields.

7. We want to display all of the available tags. To change this, click the current pager style link. Pick the Display all items radio and click Apply (all displays). On the next model, click Save to keep the offset at 0.
8. Next we will sort the view by tag name instead of order of creation. Click Add on the Sort criteria section. Add Taxonomy term: Name and click Apply (all displays) to use the default sort by ascending.
9. Press **Save** to save the view.

10. Visit **Structure** and **Block layout** to place the block on your Drupal site. Press **Place block** for the **Sidebar** region in the **Bartik** theme.

11. Filter the list by typing your view’s name. Press **Place block** to add your view’s block to the block layout.

12. Finally click on **block** to commit your changes!

## How it works...

In the Drupal 8 plugin system there is a concept called **Derivatives**. Plugins are small pieces of swappable functionality within Drupal 8. Plugins and plugin development are covered in **Chapter 7, Plug and Play with Plugins**. A derivative allows a module to present multiple variations of a plugin dynamically. In the case of Views, it allows the module to provide variations of a **ViewsBlock** plugin for each view that has a block display. Views implements the `\Drupal\views\Plugin\Block\ViewsBlock\ViewsBlock` class, providing the base for the dynamic availability of these blocks. Each derived block is an instance of this class.

When Drupal initiates the block, Views passes the proper configuration required. The view is then executed and the display is rendered whenever the block is displayed.
There's more...

We will explore some of the other ways in which Views interacts with blocks.

Exposed forms as blocks

Pages and feeds have the ability to provide blocks, however not for the actual content displayed. If your view utilizes exposed filters you have the option to place the exposed form in a block. With this option enabled you may place the block anywhere on the page, even pages not for your view!

To enable the exposed filters as a block, you must first expand the Advanced section on the right side of the Views edit form. Click on the Exposed form in block option from the Advanced section. In the options modal that opens, select the Yes radio button and click Apply. You then have the ability to place the block from the Block layout form.

An example for using an exposed form in a block is for a search result view. You would add an exposed filter for keywords that control the search results. With the exposed filters in a block you can easily place it in your site's header. When an exposed filters block is submitted, it will direct users to your view's display.

See also

- Chapter 7, Plug and Play with Plugins, to learn more about derivatives

Utilizing dynamic arguments

Views can be configured to accept contextual filters. Contextual filters allow you to provide a dynamic argument that modifies the view's output. The value is expected to be passed from the URL; however, if it is not present there are ways to provide a default value.

In this recipe we will create a new page called My Content, which will display a user's authored content on the route /user/%/content.
How to do it...

1. Visit Structure and then Views. This will bring you to the administrative overview of all the views created. Click on Add new view to begin creating a new view.

2. Set the View name to My Content.

3. Next, we modify the View settings. We want to display Content of the type All and leave the Tagged with empty. This will allow all content to be displayed.

4. Choose to Create a page. Keep the page title the same. We need to change the path to be user/%/content. Click Save and edit to move to the next screen and add the contextual filter.

   ![When building a views page, adding a percentage sign to the path identifies a route variable.](image)

5. Toggle the Advanced portion of the form on the right hand side of the page. Click on Add in the Contextual filters section.

6. Select Content: Authored by and then click Apply (all displays).

7. Change the default value When the filter is not in the URL to Display "Access Denied" to prevent all content from being displayed with a bad route value.

   ![WHEN THE FILTER VALUE IS NOT IN THE URL](image)

   - Default actions
     - Display all results for the specified field
     - Display a summary
     - Display contents of "No results found"
     - Display "Access Denied"

   - EXCEPTIONS

8. Click Apply (all displays) and save the form.

9. Visit /user/1/content and you will see content created by the root admin!

How it works...

Contextual filters mimic the route variables found in the Drupal routing system. Variables are represented by percentage signs as placeholders in the view's path. Views will match up each placeholder with contextual filters by order of their placement. This allows you to have multiple contextual filters; you just need to ensure they are ordered properly.
Views is aware of how to handle the placeholder because the type of data is selected when you add the filter. Once the contextual filter is added there are extra options available for handling the route variable.

**There's more...**

We will explore extra options available when using contextual filters.

**Previewing with contextual filters**

You are still able to preview a view from the edit form. You simply add the contextual filter values in to the text form concatenated by a forward slash (/). In this recipe you could replace visiting `/user/1/content` with simply inputting `1` into the preview form and updating the preview.

**Displaying as a tab on the user page**

Even though the view created in the recipe follows a route under `/user`, it will not show up as a local task tab until it has a menu entry defined. From the **Page settings** section you will need to change **No menu** from the **Menu** option. Clicking on that link will open the menu link settings dialog.

Select **Menu tab** and provide a **Menu link title**, such as **My Content**. Click on **Apply** and save your view.

**Altering the page title**

With contextual filters you have the ability to manipulate the current page's title. When adding or editing a contextual filter you can modify the page title. From the **When the filter value is present in the URL or a default is provided** section, you may check the **Override title** option. This text box allows you to enter in a new title that will be displayed. Additionally, you can use the information passed from the route context using the format of `%#` where the # is the argument order.

**Validation**

Contextual filters can have validation attached. Without specifying extra validation, Views will take the expected argument and try to make it *just work*. You can add validation to help limit this scope and filter out invalid route variables.

You can enable validation by checking **Specify validation criteria** from the **When the filter value is present in the URL or a default is provided** section. The default is set to – **Basic Validation** – which allows you to specify how the view should react if the data is invalid; based on our recipe, if the user was not found.
The list of Validator options is not filtered by the contextual filter item you selected, so some may not apply. For our recipe one might want User ID and select the Validate user has access to the User. This validator would make sure the current user is able to view the route's user's profile. Additionally, it can be restricted further based on role.

![Validate user has access to the User](image)

This gives you more granular control over how the view operates when using contextual filters for route arguments.

**Multiple and exclusion**

You may also configure the contextual filter to allow AND or OR operations along with exclusion. These options are under the More section when adding or editing a contextual filter.

The Allow multiple values option can be checked to enable AND or OR operations. If the contextual filter argument contains a series of values concatenated by plus (+) signs it acts as an OR operation. If the values are concatenated by commas (,) it acts as an AND operation.

When the Exclude option is checked the value will be excluded from the results rather than the view being limited by it.

**Adding a relationship in a View**

As stated at the beginning of the chapter, Views is a visual query builder. When you first create a view, a base table is specified to pull data from. Views automatically knows how to join tables for field data, such as body text or custom attached fields.
When using an entity reference field, you have the ability to display the value as the raw identifier, the referenced entity's label, or the entire rendered entity. However, if you add a Relationship based on a reference field you will have access to display any of that entity's available fields.

In this recipe, we will update the Files view, used for administering files, to display the username of the user who uploaded the file.

How to do it...

1. Visit **Structure** and then **Views**. This will bring you to the administrative overview of all the views that have been created.
2. Find the **Files** view and click **Edit**.
3. Click on **Advanced** to expand the section and then click **Add** next to **Relationships**.
4. Search for **user**. Select the **User who uploaded** relationship option and click **Apply (this display)**.

5. Next we will be presented with a configure form for the relationship. Click **Apply (this display)** to use the defaults.
6. Add a new field by clicking **Add** in the **Fields** section.
7. Search for **name** and select the **Name** field and click **Apply (this display)**.
8. This view uses aggregation, which presents a new configuration form when first adding a field. Click **Apply and continue** to use the defaults.

![Image](image.png)

We will discuss Views and aggregation in the *There's more...* section.

9. We will use the default field settings that will provide the label **Name** and format it as the username and link to the user’s profile. Click **Apply (all displays)**.

![Configure field: User: Name](image.png)

10. Click on **Save** to finish editing the view and commit your changes.

**How it works...**

Drupal stores data in a normalized format. Database normalization, in short, involves the organization of data in specific related tables. Each entity type has its own database table and all fields have their own database table. When you create a view and specify what kind of data will be shown, you are specifying a base table in the database that Views will query. Views will automatically associate fields that belong to the entity and the relationship to those tables for you.
Displaying Content through Views

When an entity has an Entity reference field you have the ability to add a relationship to the referenced entity type's table. This is an explicit definition, whereas fields are implicit. When the relationship is explicitly defined all of the referenced entity type's fields come into scope. The fields on the referenced entity type can then be displayed, filtered, and sorted by.

**There's more...**

Using relationships in Views allows you to create some powerful displays. We will discuss aggregation and additional information about relationships.

**Relationships provided by entity reference fields**

Views uses a series of hooks to retrieve data that it uses to represent ways to interact with the database. One of these is the `hook_field_views_data` hook, which processes a field storage configuration entity and registers its data with Views. The Views module implements this on behalf of Drupal core to add relationships, and reverse relationships, for Entity reference fields.

Since Entity reference fields have set schema information, Views can dynamically generate these relationships by knowing the field's table name, destination entity's table name, and the destination entity's identifier column.

**Relationships provided through custom code**

There are times where you would need to define a relation on your own with custom code. Typically, when working with custom data in Drupal, you would more than likely create a new entity type, covered in Chapter 9, Confiuration Management – Deploying in Drupal 8. This is not always the case, however, and you may just need a simple method of storing data. An example can be found in the Database Logging module. The Database Logging module defines schema for a database table and then uses `hook_views_data` to expose its database table to Views.

The `dblog_schema` hook implementation returns a `uid` column on the watchdog database table created by the module. That column is then exposed to Views with the following definition:

```php
$data['watchdog']['uid'] = array(
    'title' => t('UID'),
    'help' => t('The user ID of the user on which the log entry was written.'),
    'field' => array(
        'id' => 'numeric',
    ),
    'filter' => array(
        'id' => 'numeric',
    ),
    'argument' => array(
        'id' => 'numeric',
    ),
```
This array tells Views that the watchdog table has a column named uid. It is numeric in nature for its display, filtering capabilities and sorting capabilities. The relationship key is an array of information that instructs Views how to use this to provide a relationship (LEFT JOIN) on the users table. The User entity uses the users table and has the primary key of uid.

**Using Aggregation and views.**

There is a view setting under the Advanced section that allows you to enable aggregation. This feature allows you to enable the usage of SQL aggregate functions, such as MIN, MAX, SUM, AVG, and COUNT. In this recipe, the Files view uses aggregation to SUM the usage counts of each file in the Drupal site.

Aggregation settings are set for each field and when enabled have their own link to configure the settings.
Providing an Entity Reference result View

The Entity reference field, covered in Chapter 2, *The Content Authoring Experience*, can utilize a custom view for providing the available field values. The default entity reference field will display all available entities of the type it is allowed to reference. The only available filter is based on the entity bundle, such as only returning Article nodes. Using an entity reference view you can provide more filters, such as only content that user has authored.

In this recipe we will create an entity reference view that filters content by the author. We will add the field to the user account form, allowing users to select their favorite contributed content.

How to do it...

1. Visit Structure and then Views. This will bring you to the administrative overview of all the views that have been created. Click on Add new view to begin creating a new view.

2. Set the View name to My Content Reference View. Modify the View settings. We want to display Content of the type All and leave the Tagged with empty.

3. Do not choose to create a page or block. Click Save and edit to continue working on your view.

4. Click on the Add button to create a new display. Select the Entity Reference option to create the display.
5. The **Format** will be automatically set to **Entity Reference List**, which utilizes fields. Click on **Settings** next to it to modify the style format.

6. For **Search Fields**, check the **Content:Title** option then click **Apply**. This is what the field will autocomplete search on.

7. You will need to modify the **Content: Title** field to stop it from wrapping the result as a link. Click on the field label and uncheck **Link to the Content**. Click **Apply (all displays)** to save.

8. Click on **Save** to save the view.

9. Go to **Configuration** and then **Account settings** to be able to **Manage fields** on user accounts.

10. Add a new **Entity Reference** field that references **Content**, call it Highlighted contributions, and allow it to have unlimited values. Click the **Save field settings** button.
11. Change the Reference type method to use View: Filter by an entity reference view and select the view we have just created:

```
**REFERENCE TYPE**

Reference method *
Views: Filter by an entity reference view ▼

View used to select the entities *
my_content_reference_view - Entity Reference ▼

Choose the view and display that select the entities that can be referenced.
Only views with a display of type "Entity Reference" are eligible.

View arguments

Provide a comma separated list of arguments to pass to the view.
```

**How it works...**

The entity reference field definition provides selection plugins. Views provides an entity reference selection plugin. This allows entity reference to feed data into a view to receive available results.

The display type for Views requires you to select which fields will be used to search against when using the autocomplete widget. If not using the autocomplete widget and using the select list or checkboxes and radio buttons, then it will return the view's entire results.

**There's more...**

**View arguments**

Entity reference view displays can accept contextual filter arguments. These are not dynamic, but can be passed manually through the field's settings. The View arguments field allows you to add a comma separated list of arguments that are passed to the view. The order should match the order of the contextual filters as configured.

In this recipe we could have added a Content: type contextual filter that fell back to Display all results if the argument was missing. This allows the view to be reused in multiple references. Perhaps there is one view that should limit the available references to all Articles created by the current user. You would then add Article to the text field and pass the argument to the view.

**See also**

- Chapter 7, Plug and Play with Plugins, to learn more about plugins
Where to buy this book

You can buy Drupal 8 Development Cookbook from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers. Click here for ordering and shipping details.