**Comparing Roles, Profiles, and Permission Sets**

Although it's easy to confuse profiles and permission sets with roles, they actually control two very different things.

As we learned earlier in this chapter, profiles and permission sets control a user's object- and field-level access permissions. Indeed, a user can't be defined without being assigned to a particular profile, since the profile specifies the most basic access for users.

Roles, on the other hand, primarily control a user's record-level access permissions through role hierarchy and sharing rules. Although a role assignment isn't exactly required when we define a user, it would be foolish of us *not* to assign a role since it makes it so much easier to define our record-level permissions. Indeed, trying to define record-level permissions without assigning a role to a user would be a lot like trying to travel from New York to San Francisco by car when there's an airplane available—there's just a much more efficient way of doing it!

# Role Hierarchies in Our Recruiting App

Given the Universal Containers role hierarchy that's pictured in the Universal Containers Role Hierarchy image, let's think about how implementing this hierarchy will open up certain kinds of record-level permissions to various users of our Recruiting app. Remember, since defining our org-wide defaults, our hiring managers are currently allowed to only view all position, job posting, and employment website records, and to view and update other recruiting records that they own. That doesn't make our app all that useful. However, once we implement our role hierarchy, we'll automatically grant several kinds of record-level permissions to various users. For example:

* The CEO, Cynthia Capobianco, will be able to view and update every record that anyone else in the organization can view and update.
* The VP of Development, Andrew Goldberg, will be able to view and update any record that his managers or his managers' employees can view or update.
* The VP of Human Resources, Megan Smith, will be able to view and update any record that Phil Katz, her recruiting manager, or Mario Ruiz, Phil's recruiter, can view and update.
* The Recruiting Manager, Phil Katz, will be able to view and update any record that is owned by Mario Ruiz, his recruiter.
* The Software Development manager, Ben Stuart, will be able to view and update any record that is owned by Melissa Lee, Tom Zales, or Craig Kingman, his software engineers.
* The director of QA, Clark Kentman, will be able to view and update any record that is owned by Flash Stevenson or Harry Potterham, his QA Engineers.
* The director of Product Management, Frank Linstrom, will be able to view and update any record that is owned by Amy Lojack or Andy Macrola, his product managers.

As we can see, the role hierarchy is very powerful in opening up data for people high up in the role hierarchy tree! However, let's look at some of the gaps that we still have in our record-level permissions:

* Megan Smith (and her whole recruiting team) won't be able to view any reviews that are owned by members of Andrew Goldberg's Development team because she doesn't have a direct line down to any Development roles in the role hierarchy.
* Ben Stuart, the software development manager, also won't be able to see any reviews that were written by members of the QA or Product Management groups, even if QA engineers or product managers interviewed candidates for a software engineering position in his group.
* Melissa Lee, a software engineer, won't be able to see the records for candidates that she's supposed to interview.

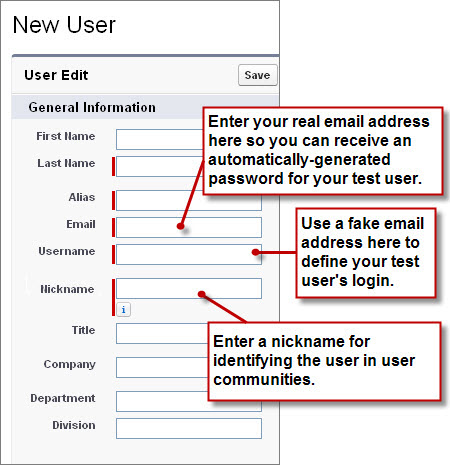
Clearly we'll need to use other record-level sharing methods to open up data between peers in the same group, and also between groups that appear in different branches of the role hierarchy (we'll get to those later in this chapter). However, the role hierarchy does give us a good start toward opening up record access, so let's take a look now at how to define it.

**Try It Out: Create a User**

Before we set up the role hierarchy, we’re going to need some users to assign the roles to. First let’s create the CEO of Universal Containers: Cynthia Capobianco.

1. From Setup, enter Users in the Quick Find box, then select **Users**.
2. Click **New User**.
3. In First Name, enter Cynthia.
4. In Last Name, enter Capobianco.
5. Enter your email address in the Email field, and a fake email in Username.

Each user must have a unique username across all Salesforce organizations. The value in Email must be a valid email address, though. We can use the “fake” value in the Username field to log in to the app as Cynthia, but we'll get her automatically generated password at the real email account specified in Email. Without that password, we'd never be able to log in!

New User Edit Page 

When creating a new user, you’re also required to create the user's nickname. The nickname is used to identify the user in the Ideas app, which is a community of users who post, vote for, and comment on ideas. Consider it an online suggestion box that includes discussions and popularity rankings for any subject. The nickname can contain up to 40 alphanumeric characters. For more information, see the Salesforce Help.

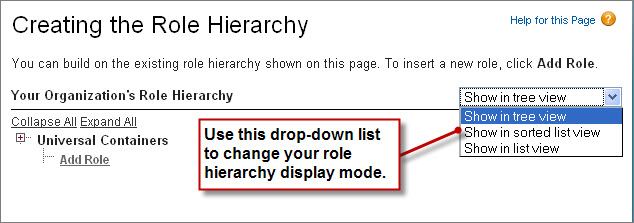
1. Leave Role unspecified. We’ll assign this when we create our role hierarchy.
2. In User License, select Salesforce.
3. In Profile, select Standard Employee.
4. Fill in any remaining fields that you want to.
5. Click **Save**.

**ry It Out: Define a Role Hierarchy**

Implementing a role hierarchy in the platform is easy once you have an idea of what the hierarchy should look like. It's best to start with your company's org chart and then consolidate different job titles into single roles wherever possible. For example, if Ben Stuart's software development group has a staff software engineer and a junior software engineer, these positions can be consolidated into a single Software Engineer role in the hierarchy.

Once that's squared away, we can get started defining the role hierarchy itself. For our exercise, we'll use the role hierarchy we talked about previously.

1. From Setup, enter Roles in the Quick Find box, then select **Roles**. If you see an introductory splash page called Understanding Roles, click **Set Up Roles** at the bottom of the page to skip to the actual tool.

Empty Role Hierarchy Page in Tree View Mode 

The default view for this page is the tree view, as indicated in the drop-down list on the far right side of the Role Hierarchy title bar. When creating a role hierarchy, it's probably easiest to stick with this or the list view, because they both make it easy to see how the roles all fit together in the hierarchy. The sorted list view is best if you know the name of a role that you want to find but aren't sure where it fits in the hierarchy, or if you don't want to click open all the tree nodes. For our purposes, we'll stick with the tree view.

When you first start defining a role hierarchy, the tree view displays a single placeholder node with the name of your organization. From this point, we need to add the name of the role that is highest up in the hierarchy—in our case, the CEO.

If you're building your Recruiting app with a free Developer Edition organization, you may have a role hierarchy predefined as a sample. That's alright. You can still follow along and create some more roles.

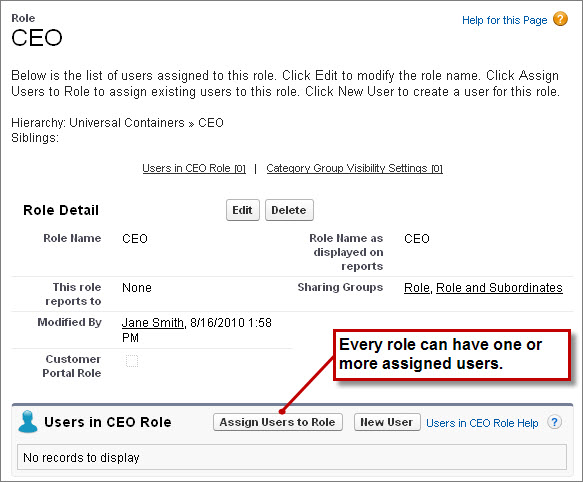
1. Just under the name of your organization—in this case, Universal Containers—click **Add Role**.

If the CEO role already exists, click **Edit**.

1. In the Label text box, enter CEO. The Role Name text box autopopulates with CEO.
2. In the This role reports to text box, click the lookup icon Magnifying glass iconand click **Select** next to the name of your organization.

By choosing the name of the organization in the This role reports to text box, we're indicating that the CEO role is a top-level position in our role hierarchy and doesn't report to anyone.

1. In the Role Name as displayed on reports text box, enter CEO. This text is used in reports to indicate the name of a role. Since you may not want a long role name, like Vice President of Product Development, taking up too much space in your report columns, it's advisable to use a shortened, yet easily identifiable, abbreviation.
2. Leave any other options, such as Opportunity Access, set to their defaults. These access options don't have anything to do with our Recruiting app, and only appear if you have the org-wide defaults for a standard object set to a level more restrictive than Public Read/Write.
3. Click **Save**.

CEO Role Detail Page 

Now that we've created our first role, we can assign the appropriate user to it.

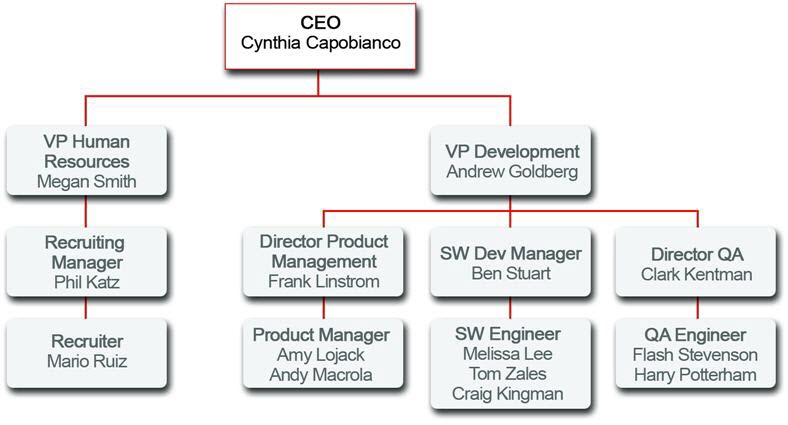
1. Click **CEO** to open the CEO role detail page.
2. In the CEO role detail page, click **Assign Users to Role**.
3. In the Available Users drop-down list, select All Unassigned.
4. Choose a user from the list (in our case, Cynthia Capobianco), and click **Add** to move her to the Selected Users for CEO list.
5. Click **Save**.

If we return to the main Roles page from Setup by entering Roles in the Quick Find box, then selecting **Roles**, we can now see our new CEO role in the hierarchy.

If you see the Sample Role Hierarchy image, click **Set Up Roles**.

1. Define the rest of the roles according to the Universal Containers Role Hierarchy diagram.

There's no need to assign users to every role at this point—we'll do that later when we create the rest of our users and test out our app.

Universal Containers Role Hierarchy 

To speed up the process of adding a new role, click **Add Role** directly under the name of the role to which the new role should report. When you do this, the This role reports to text box is automatically filled in with the name of the appropriate role.

Not too hard, right? With org-wide defaults and a role hierarchy in place, we're actually pretty close to finishing up our record-level access permissions. All we have left to do is share recruiting-related records between groups that appear in separate branches of the role hierarchy, and between peers in a single group. Fortunately, we can accomplish both of those tasks with a combination of sharing rules and manual sharing. We just need to figure out what's left that needs to be shared, and with whom.

**What's Left to be Shared?**

So what *is* left to be shared? After reviewing our table of required permissions, it turns out it's just a few more things (remember, since users always have access to the records that they own, we need to worry only about the read and update permissions for our record-level access settings):

* Recruiters need read and update access on every position, candidate, job application, and review record that exists in the app.
* Hiring managers need:
  + Read and update access on position and job posting records on which they're the hiring manager
  + Read access on candidate records for which they're the hiring manager
  + Read and update access on every job application and review record
* Interviewers need read access on the candidate and job application records for people they're interviewing, and the ability to update their reviews.

# Introducing Sharing Rules

First let's see what we can do with sharing rules. Sharing rules let us make automatic exceptions to organization-wide defaults for particular groups of users. We've already defined several specific groups with the roles that we created in the previous section, but we can also make up other groups as needed.

The thing to remember with sharing rules is that, like role hierarchies, we can use them only to open up record access to more users. Sharing rules and role hierarchies can never be stricter than our org-wide default settings.

# Sharing Rules in Our Recruiting App

Sharing rules work best when they're defined for a particular group of users that we can determine or predict in advance, rather than a set of users that is frequently changing. For example, in our Recruiting app, we need to share every position, candidate, job application, and review with every recruiter. Since recruiters all belong to either the Recruiting Manager or Recruiter roles in the role hierarchy, we can easily use a sharing rule to share those objects with the Recruiting Manager role and its subordinates.

Alternatively, consider another use case from our Recruiting app: interviewers need read access on the candidates and job applications for people they're interviewing. In this case, the set of interviewers is a lot harder to predict in advance—hiring managers might use different sets of interviewers depending on the position for which they're hiring, and the interviewers might come from different groups in the role hierarchy. As a result, this use case probably shouldn't be handled with sharing rules—the team of interviewers for any given manager is just too hard to predict.

Let's go through the set of required permissions we still need to implement and pick out the ones that would work best with sharing rules:

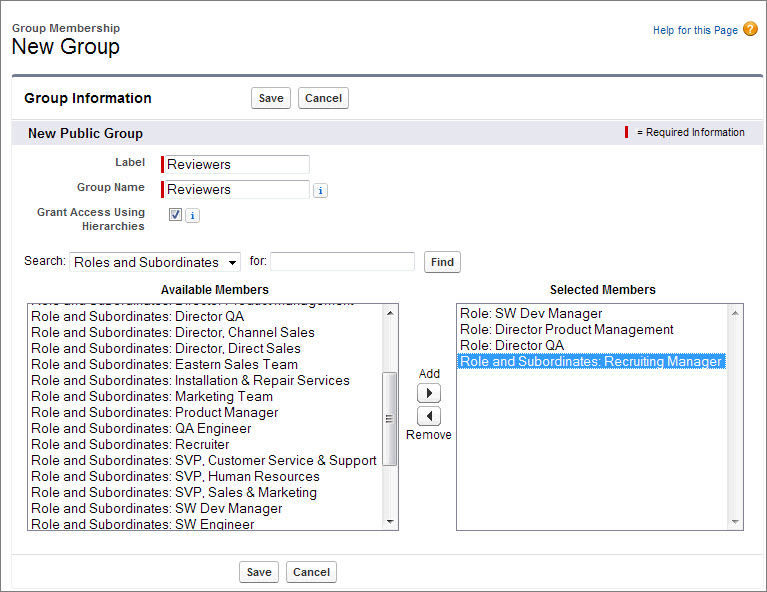
| **Use Case** | **Should we use a sharing rule?** |
| --- | --- |
| Recruiters need read and update access on every position, candidate, job application, and review record that exists in the app. | **Yes**. As we discussed previously, it's easy to pick out the group of recruiters in our role hierarchy. |
| Hiring managers need read and update access on position and job posting records on which they're the hiring manager. | **No**. It's too hard to predict which positions will be assigned to which hiring manager. We'll need to handle this use case some other way. |
| Hiring managers need read access on candidate records on which they're the hiring manager. | **No**. Again, it's too hard to predict which positions will be assigned to which hiring manager. |
| Hiring managers need read and update access on every job application and review record. | **Yes**. Since we're not restricting which job applications and reviews a hiring manager gets to read and update, we can easily pick out all of the hiring managers from our role hierarchy and define a sharing rule for them. |
| Interviewers need read access on the candidate and job application records for people they're interviewing. | **No**. As we discussed previously, it's hard to predict who will be a member of an interview team for a particular position. |

**Try It Out: Define a Public Group**

Before we dive head first into creating our sharing rules, we need to make sure that we have the appropriate public groups set up. A *public group* is a collection of individual users, other groups, individual roles, and/or roles with their subordinates that all have a function in common. For example, users with the Recruiter profile as well as users in the SW Dev Manager role both review job applications. Using a public group when defining a sharing rule makes the rule easier to create and, more important, easier to understand later, especially if it's one of many sharing rules that you're trying to maintain in a large organization. You'll need to create a public group if you ever want to define a sharing rule that encompasses more than one or two groups or roles, or any individual.

Looking at the required permissions that we want to implement, there are just two objects that need a public group for their sharing rules: Job Application and Review. The good news is that we can cover these objects in a single group because the Review object is on the detail side of a master-detail relationship, so it inherits the sharing settings we apply to the Job Application object. Since both recruiters and hiring managers need read and update access to job applications and reviews, let's go ahead and make a public group called Reviewers that encompasses recruiters and hiring managers.

1. From Setup, enter Public Groups in the Quick Find box, then select **Public Groups**.
2. Click **New**.

New Public Group Page 

The New Public Group page allows you to choose other public groups, individual roles, individual roles including the roles' subordinates, or individual users.

1. In the Label text box, enter Reviewers. Click in the Group Name text box to populate it automatically. Group Name refers to the unique name used by the API and managed packages.
2. In the Search drop-down list, choose Roles.
3. In the Available Members list, select SW Dev Manager, Director Product Management, and Director QA, then click **Add**.
4. Go back up to the Search drop-down list, and this time choose Role and Subordinates.
5. In the Available Members list, select Recruiting Manager, and click **Add**.
6. Click **Save**.

**Try It Out: Define Sharing Rules**

Since we just defined our Reviewers public group, let's use it to define our sharing rule for review records.

1. From Setup, enter Sharing Settings in the Quick Find box, then select **Sharing Settings**.

Remember this page? We were last here when we defined our org-wide defaults.

1. In the Manage sharing settings for drop-down list, choose Job Application.

Choosing an object in this drop-down list allows us to focus in on the org-wide defaults and sharing rules for a single object at a time rather than looking at all of them in a long page—a really useful thing if you've got a large organization with several custom objects.

If you had chosen Review instead of Job Application, you wouldn’t have the option of creating sharing rules, since you can’t create sharing rules for a detail record in a master-detail relationship. However, since you chose Job Application, a Sharing Rules related list appears. We'll use that to create the sharing rules that will apply to both the Job Application and the Review objects.

1. In the Job Application Sharing Rules area, click **New**.
2. In the Label text box, enter Review Records.
3. Click the Rule Name text box to populate it automatically.
4. For the rule type, make sure Based on record owner is selected.
5. In the Job Application: owned by members of drop-down list, select Public Groups.
6. Next to that drop-down list, choose All Internal Users.

Just as we talked about already, you can define a sharing rule only for a single public group, role, or role with all of its subordinates. By default, the platform includes a default public group that encompasses every user in your organization.

1. In the Share with drop-down list, select Public Groups.
2. Next to that drop-down list choose Reviewers.
3. In the Access Level drop-down list, select Read/Write.
4. Click **Save**.
5. Click **OK** in the dialog box that says this operation could take significant time.

And that's it! We just created a rule that shares reviews written and owned by any member of the organization with all recruiters and hiring managers. Since reviewers and hiring managers all need the power to read and update reviews, we handled everyone with a single sharing rule and a public group.

To finish up, create the following owner-based sharing rules:

| Table 1. Additional Sharing Rules | | | | |
| --- | --- | --- | --- | --- |
| **Object** | **Rule Label** | **Owned by...** | **Should be shared with...** | **Access Level** |
| Candidate | Edit Candidates | All Internal Users | The role and subordinates of the Recruiting Manager | Read/Write |
| Employment Website | Edit Employment Websites | The role and subordinates of the Recruiting Manager | Reviewers | Read/Write |
| Position | Edit Positions | The role and subordinates of the Recruiting Manager | The role and subordinates of the Recruiting Manager | Read/Write |

The sharing rule for the Employment Website object is necessary to let hiring managers post jobs, even though they will never update employment website records directly (the org-wide defaults prevent that). Without the rule, hiring managers can see employment website records but can’t create job postings. This is because the Job Posting object is a junction object (as you may recall from the last chapter), and the Employment Website object is one of the Job Posting object's two master-detail relationships. Sharing access to a junction object record is determined by a user's sharing access to both associated master records (in this case, the associated position and employment website records) and the Sharing Setting option on the relationship field. For example, if the sharing setting on both parents is Read/Write, then the user must have Read/Write access to both parents in order to have Read/Write access to the junction object.

In the sharing rule for the Employment Website object, we opted to use the existing Reviewers public group. Doing this saved us a few clicks without granting access to any users who shouldn't be looking at employment website records.

**Introducing Manual Sharing**

Now let's talk about what we have left to do to finish defining our sharing model. After implementing our sharing rules, the following required permissions remain:

* *Hiring managers need read and update access on position records on which they're the hiring manager.*
* *Hiring managers need read access on candidate records on which they're the hiring manager.*
* *Interviewers need read access on the candidate and job application records for people they're interviewing.*

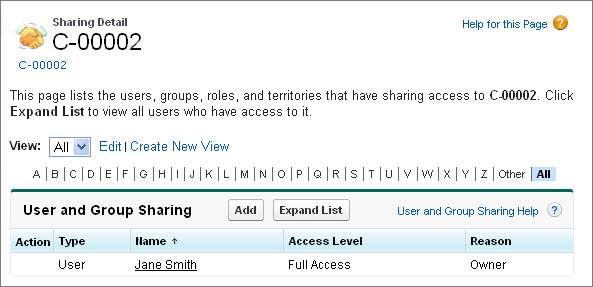
We didn't implement those required permissions with sharing rules because it was too hard for us to come up with a consistent group of users who would need access to a particular set of records. Really, this is where the job of the recruiter comes into play. A recruiter like Mario Ruiz owns the position, candidate, and job application records for jobs that he's trying to fill, and he also knows the hiring manager and interviewers who should be assigned to them.

Fortunately, we have one final type of record-access setting that allows Mario to share specific records with other specific users: manual sharing. With manual sharing, Mario can grant read or read/write access on records that he owns to any other user, role, or public group. Although it isn't automated like organization-wide defaults, role hierarchies, or sharing rules, manual sharing gives Mario the flexibility to share particular records with the ever-changing groups of interviewers and hiring managers with whom he has to deal every day.

**Try It Out: Define a Manual Sharing Rule**

Let's pretend that we're a recruiter like Mario and we need to share a particular candidate record that we own with another role, group, or user:

1. On the detail page for the candidate, click **Sharing**.

Sharing Detail Page 

Since we own this candidate record, we get to see details about who else can see the record and why. If we didn't own this record, there would be a message about not having sufficient privileges.

If we wanted to view the names of each user who has access to the record rather than just the names of the roles and public groups, we could click **Expand List** in this page. Although the operation can take some time depending on the number of users in our organization, it's helpful to determine whether we need to define a manual sharing rule for a particular user or if he or she already has access.

1. Click **Add**.
2. In the Search drop-down list, choose whether we want to manually share the record with a user, public group, role, or role and subordinates.
3. In the Available list, select the user, public group, or role that should have access to the record, and click **Add**.
4. In the Access Level drop-down list, specify whether the user, public group, or role should have read or read/write access to the record.
5. Click **Save**.