Chapter 5

Enhancing the Simple App with Advanced Fields and Page Layouts

In this chapter ...

- Adding Advanced Fields
- Introducing Validation Rules
- Introducing Page Layouts

In the last chapter, we got our Recruiting app off to a quick start by defining the Position custom object, tab, and several simple fields. This simple version of our app had the same look and feel as any other page in the Force Platform, and we were able to whip it together in a matter of minutes.

In this chapter, we're going to enhance the Positions tab: first by defining a few more advanced fields, then by defining a validation rule to make sure our data stays clean, and finally by moving our fields around within a page layout. These additions will help change the detail page of our Positions tab from a somewhat flat and inelegant user interface to something that users find powerful and intuitive to use. Let's get started!

Adding Advanced Fields

In this section, let's revisit the custom field wizard to help us create fields with more sophisticated functionality: picklists, dependent picklists, and fields that leverage custom formulas. We'll see how the platform's user interface helps guide us through the setup of these more complicated fields.

Introducing Picklists

When viewing the preview of what we wanted our Positions page to ultimately look like, there were several fields that were specified with drop-down lists. In Force Platform terms, these fields are called *picklists*, and they consist of several predefined options from which a user can select.

Picklists come in two flavors: a standard picklist, in which a user can select only one option, and a multi-select picklist, in which a user can select multiple options at a time. For the purposes of our Position object, we need to define standard picklists for a position's location, status, type of job, functional area, and job level.

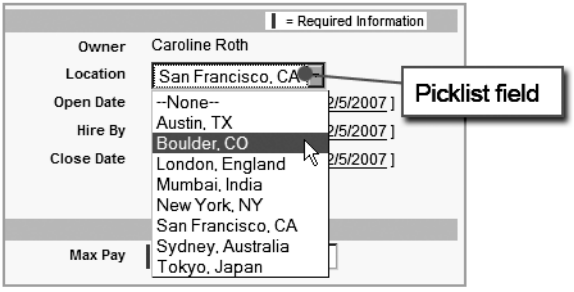

Figure 22: Location Picklist Field

Try It Out: Adding Picklists

Let's walk through the creation of the `Location` picklist field. Then, as in the previous chapter, we'll give you the information that you need to create the others on your own.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Picklist` data type and click **Next**.
5. In the `Field Label` text box, enter `Location`.

6. In the large text area box just below, enter the following picklist values, each on its own line:
   - San Francisco, CA
   - Austin, TX
   - Boulder, CO
   - London, England
   - New York, NY
   - Mumbai, India
   - Sydney, Australia
   - Tokyo, Japan
7. Select the `Use first value as default value` checkbox.

This option allows us to populate the field with a default value. If you leave it deselected, the field defaults to None on all new position records. Otherwise the field defaults to the first value that you specify in the list of possible picklist values. Because most positions at Universal Containers are based at its headquarters in San Francisco, CA, this should be the default.

8. Accept all other default settings for field-level security and page layouts.
9. Click **Save & New**.

Easy! Now specify the remaining picklists according to the table below:

Table 6: Status, Type, Functional Area, and Job Level Picklist Values

Data Type	Field Label	Picklist Values	Sort Alphabetically?	Use First Value as Default?
Picklist	Status	• New Position • Pending Approval • Open - Approved • Closed - Filled • Closed - Not Approved • Closed - Canceled	No	Yes
Picklist	Type	• Full Time	No	No

Data Type	Field Label	Picklist Values	Sort Alphabetically?	Use First Value as Default?
		• Part Time • Internship • Contractor		
Picklist	Functional Area	• Finance • Human Resources • Information Technology • Retail Operations • Warehousing • Miscellaneous	Yes	No
Picklist	Job Level	• FN-100 • FN-200 • FN-300 • FN-400 • HR-100 • HR-200 • HR-300 • HR-400 • IT-100 • IT-200 • IT-300 • IT-400 • RO-100 • RO-200 • RO-300 • RO-400 • WH-100 • WH-200 • WH-300 • WH-400 • MC-100	Yes	No

Data Type	Field Label	Picklist Values	Sort Alphabetically?	Use First Value as Default?
		• MC-200 • MC-300 • MC-400		

Introducing Field Dependencies

Now that we've made all those picklists, answer this question: How many times have you clicked on a drop-down list and found far too many values to choose from? For example, maybe you were selecting Uruguay from a list of countries, and every country in the world was on the list. That meant that you had to scroll all the way down to the countries that started with the letter U. What a pain!

Fortunately, the folks who built the Force Platform have encountered that situation a few times themselves, and as a result, they've given us a tool to help us avoid this problem with our own picklist fields: *field dependencies*.

Field dependencies are filters that allow us to change the contents of a picklist based on the value of another field. For example, rather than displaying every value for Country in a single picklist, we can limit the values that are displayed based on a value for another field, like Continent. That way our users can find the appropriate country more quickly and easily.

Picklist fields can be either *controlling* or *dependent* fields. A controlling field controls the available values in one or more corresponding dependent fields. A dependent field displays values based on the value selected in its corresponding controlling field. In the previous example, the Continent picklist is the controlling field, while the Country picklist is the dependent field.

Try It Out: Creating a Dependent Picklist

Looking at the picklists that we've created, it's quickly obvious that our users might get frustrated with the length of our `Job Level` picklist. Let's make our users happy by turning `Job Level` into a dependent field of the `Functional Area` picklist. Doing this will allow users to see only the four relevant job level values when a department is selected in the `Functional Area` picklist:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.

3. In the Custom Fields & Relationships related list, click **Field Dependencies**.
4. Click **New**.
5. For the `Controlling Field` drop-down list, choose Functional Area.
6. For the `Dependent Field` drop-down list, choose Job Level.
7. Click **Continue**.

A field dependency matrix displays with all the values in the controlling field across the top header row and the dependent field values listed in the columns below. For each possible value of the controlling field, we need to include the values that should be displayed in the dependent picklist when that controlling value is selected. In the field dependency matrix, yellow highlighting shows which dependent field values are included in the picklist for a particular controlling field value.

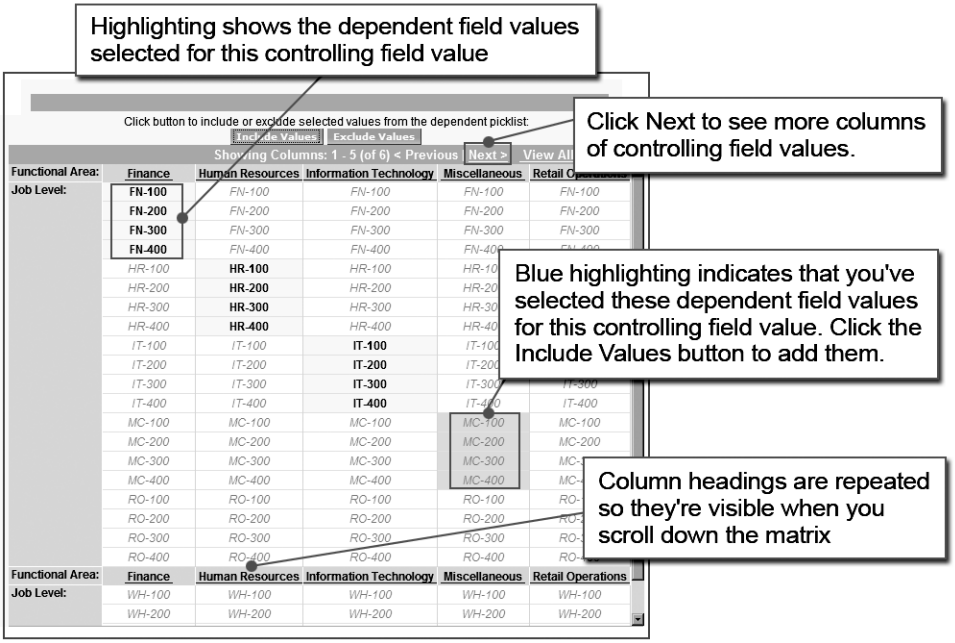

Figure 23: Field Dependency Matrix

To include a dependent field value, you simply double-click it. To exclude a dependent value from the list, double-click it again.

For example, let's try it out by including the values that should be displayed in the `Job Level` picklist whenever Finance is selected in the `Functional Area` picklist:

8. In the column labeled Finance, double-click FN-100, FN-200, FN-300, and FN-400.

Those four fields should now be shaded yellow in the `Finance` column.

Instead of double-clicking every `Job Level` value, we can also use SHIFT+click to select a range of values or CTRL+click to select multiple values at once. Once those values are highlighted in blue, we can click **Include Values** to include them, or **Exclude Values** to remove them. Let's try it out.

9. In the column labeled Human Resources, single-click HR-100 and then press and hold the SHIFT key while clicking HR-400.
10. Click **Include Values**.

Now we have values selected for both the Finance and Human Resources columns!

11. Continue highlighting the appropriate values for all of the remaining columns, as described in the following table.

Tip: To get to all of the values that you need to modify for this step, you'll need to click **Previous** or **Next** to see additional columns.

Table 7: Functional Area and Job Level Field Dependency Matrix

Functional Area **(Controlling picklist field)**	**Job Level** **(Dependent picklist field)**
Finance	• FN-100 • FN-200 • FN-300 • FN-400
Human Resources	• HR-100 • HR-200 • HR-300 • HR-400
Information Technology	• IT-100 • IT-200 • IT-300

Functional Area (Controlling picklist field)	Job Level (Dependent picklist field)
	• IT-400
Retail Operations	• RO-100 • RO-200 • RO-300 • RO-400
Warehousing	• WH-100 • WH-200 • WH-300 • WH-400
Miscellaneous	• MC-100 • MC-200 • MC-300 • MC-400

12. Click **Preview** to test the results in a small popup window.
13. Click **Save**.

Look at What We've Done

Now that we've created all those picklists, let's revisit the Positions tab to see what we have so far.

1. Go to the Positions tab.
2. Click **New**.
3. In the `Functional Area` picklist, select Finance.
4. Open the `Job Level` picklist.

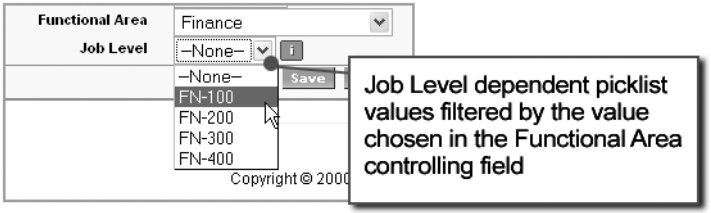

Figure 24: Dependent Picklist Fields

Our Recruiting app users are going to be very happy that they no longer have to deal with a long, onerous picklist. Now let's go add a field that's even more powerful and complex than a dependent picklist: a custom formula field.

Introducing Custom Formula Fields

Up to this point, the fields that we've defined have all had one thing in common—they each require a user to give them a value. Fields like that are very helpful for storing and retrieving data, but wouldn't it be great if we could somehow define a "smart" field? That is, what if we could define a field that looked at information that was already entered into the system and then told us something new about it?

Fortunately, *custom formula fields* give us the ability to do just that. Just as you can use a spreadsheet program like Microsoft Excel to define calculations and metrics specific to your business, we can use custom formula fields to define calculations and metrics that are specific to our Recruiting app.

For example, on our Position object, we've already created fields for minimum pay and maximum pay. If Universal Containers gives out yearly bonuses based on salary, we could create a custom formula field that automatically calculated the average bonus that someone hired to that position might receive.

How would we perform this calculation if we were using a spreadsheet? The columns in our spreadsheet would represent the fields that we defined on our Position object, and each row of the spreadsheet would represent a different position record. To create a calculation, we'd enter a formula in a new column that averages the values of `Min Pay` and `Max Pay` in a single row and then multiplies it by a standard bonus percentage. We could then determine the average bonus for every position record row in our spreadsheet.

Custom formulas work in a very similar way. Think of a custom formula like a spreadsheet formula that can reference other values in the same data record, perform calculations on them, and return a result. However, instead of using cell references, you use *merge field* references.

And, instead of typing characters in a box, you have a wizard to help you select fields, operators, and functions.

The net result is that anyone can quickly and easily learn to create formula fields. And, as with all platform tools, the cloud computing delivery model makes it easy to experiment. You can create a formula, view the results, and change the formula again and again, as many times as you want! Your underlying data is never affected.

Tip: When defining your own custom formula fields, leverage the work of others. You can find more than a hundred sample formulas on the Salesforce.com Community website at http://blogs.salesforce.com/features/2006/03/custom_formula_.html.

Calculating How Long a Position Has Been Open

Let's now think about another custom formula field that we could create for our Position object—a custom formula field that calculates how many days a position has been open. To do this, let's first think about the logic that we should use to define the field, and then we can go through the process of creating it in our Recruiting app.

Let's think about the data that we need to make this calculation: we need to know the current date and the date that the position was created. If we could somehow subtract these two, we'd have the number of days that the position has been open. Fortunately, it's easy to get both of these values:

- For the current date, we can simply use the platform's built-in `TODAY()` function. `TODAY()` returns today's date.
- For the date that the position was opened, we can use the `Open Date` field that we defined in the last chapter. Because we're using it in a custom field, we'll refer to it by its internal name, `Open_Date__c`.

Now that we have our two dates, we want to subtract them: `TODAY() - Open_Date__c`. Even if the two dates span different months or years, the platform is sophisticated enough to know how to handle all the intricacies of such a calculation behind the scenes. We just have to provide the dates, and the platform can do all the rest.

So far so good, but one problem still remains—what if the position has already closed? Our formula only works if we assume the position is still open. Once it closes, however, the value of our current formula will keep incrementing every day as `TODAY()` gets farther and farther away from the original `Open Date`. If we can, we want to use the `Close Date` field in the formula instead of `TODAY()` after a position closes. How can we do this?

Once again, all we need to do is dip into the extensive library of platform functions. The `IF()` function allows us to perform a test and then return different values depending on whether the result of the test is true or false. The `IF()` function's syntax looks like this:

```
IF(logical_test,
          value_if_true,
          value_if_false)
```

For the *`logical_test`* portion, we'll test whether the `Close Date` field has a value—if it does, the position obviously must be closed. We'll test for this with a third built-in function: `ISNULL()`. `ISNULL()` takes a single field and returns true if it does not contain a value and false if it does. So now our formula looks like this:

```
IF( ISNULL( Close_Date__c ) ,
          value_if_true,
          value_if_false)
```

By replacing *`value_if_true`* and *`value_if_false`* with the other formulas we talked about, we've now figured out our whole formula:

```
IF( ISNULL( Close_Date__c ) ,
             TODAY()  -  Open_Date__c ,
             Close_Date__c  -  Open_Date__c )
```

Great! Our formula calculates the number of days a position has been open, regardless of whether it's currently open or closed. Now, let's go define a field for it on our Position object.

Try It Out: Defining a "Days Open" Custom Formula Field

We'll begin building the formula field the same way we created our other custom fields.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Formula` data type, and click **Next**.

Step 2 of the New Custom Field wizard appears.

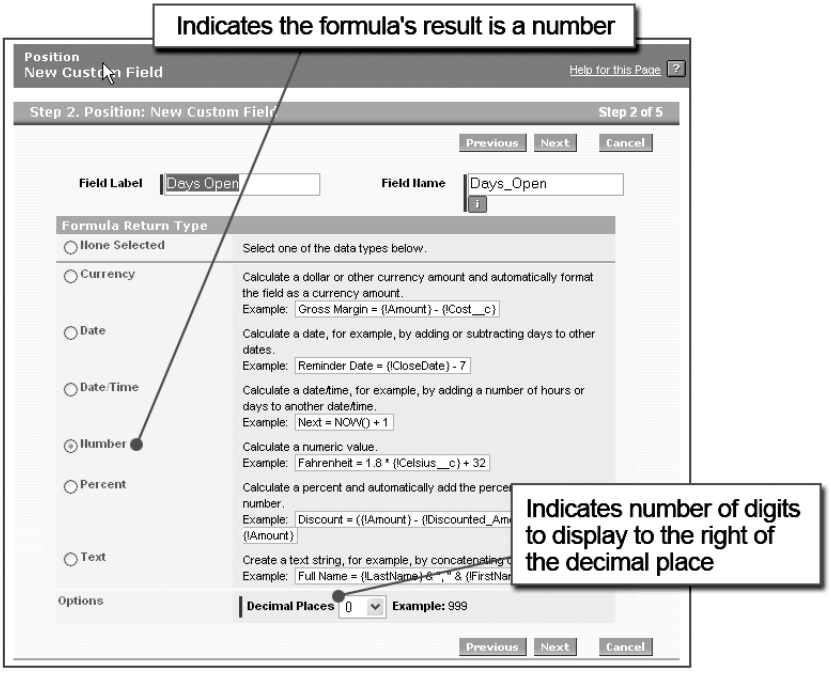

Figure 25: Custom Formula Field Wizard Step 2

5. In the `Field Label` field, enter `Days Open`.
6. Select the `Number` formula return type.

In this case, even though we're subtracting Date fields, we want to end up with just a regular numeric value.

7. Change the `Decimal Places` value to `0`, and click **Next**.

Now it's time to enter the details of our formula.

8. Click the Advanced Formula tab, as shown in the following screenshot.

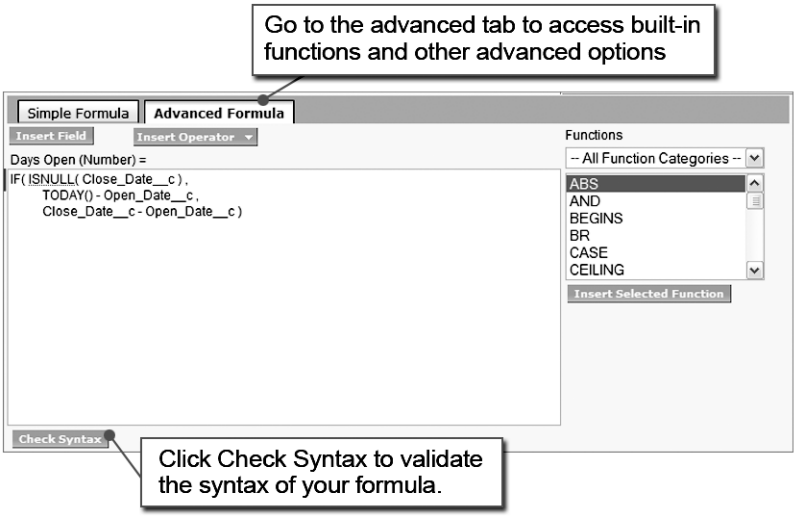

Figure 26: Custom Formula Field Editor

We want to use the Advanced Formula tab so we can access the platform's built-in functions through the Functions list on the right side.

9. From the `Functions` list, double-click `IF`.

Our formula now looks like this:

```
IF(logical_test, value_if_true, value_if_false)
```

Let's go ahead and define the logical test:

10. Delete *`logical_test`* from the formula, but leave your cursor there.
11. From the `Functions` list, double-click `ISNULL`.
12. Delete *`expression`* from the `ISNULL` function you just inserted, but leave your cursor there.
13. Click the **Insert Field** button. Two columns appear in an overlay.
14. In the left column, select Position.
15. In the right column, select `Close Date`.
16. Click **Insert**.

Did you notice that you didn't have to remember to use the internal name of the `Close Date` field? The platform remembered for you when it inserted the value. Our formula now looks like this:

```
IF( ISNULL( Close_Date__c ) , value_if_true, value_if_false)
```

Now, let's specify the value if our logical test evaluates to true:

17. Delete *value_if_true* from the formula, but leave your cursor there.
18. Press Enter on your keyboard, and space over 10 spaces.

Adding the carriage return and spaces makes our formula more legible for others.

19. From the `Functions` list, double-click `TODAY`.
20. Click the **Insert Operator** button and choose Subtract.
21. Click the **Insert Field** button.
22. In the left column, select Position.
23. In the right column, select `Open Date`.
24. Click **Insert**.

We're getting closer—our formula now looks like this:

```
IF( ISNULL( Close_Date__c ) ,
          TODAY()  -  Open_Date__c , value_if_false)
```

Finally, let's specify the value if our logical test evaluates to false:

25. Delete `value_if_false` from the formula, but leave your cursor there.
26. Press Enter on your keyboard, and space over 10 spaces.
27. Click the **Insert Field** button.
28. In the left column, select Position.
29. In the right column, select `Close Date` and click **Insert**.
30. Click **Insert Operator** and choose Subtract.
31. Click the **Insert Field** button.
32. In the left column, select Position.
33. In the right column, select `Open Date` and click **Insert**.

Our formula now matches our original:

```
IF( ISNULL( Close_Date__c ) ,
          TODAY()  -  Open_Date__c ,
          Close_Date__c  -  Open_Date__c )
```

Now that we've gone through those steps of the procedure, note that we could have just typed in the formula that we figured out in the last section. However, using the formula editor is a lot easier because you don't have to remember function syntax or internal names of fields and objects. Let's keep going and finish up this field:

34. Click **Check Syntax** to check your formula for errors.

35. In the `Description` text box, enter `The number of days a position has been (or was) open.`
36. Add an optional `Help Text` description if you wish.
37. Select `Treat blank fields as blanks`, and click **Next**.
38. Accept all remaining field-level security and page layout defaults.
39. Click **Save**.

Try It Out: Giving Fields Dynamic Default Values

We can also use custom formulas to give our fields dynamic default values. While some fields like the `Travel Required` checkbox or the `Job Location` picklist have default values that apply in every situation, there are other fields with defaults that can't be so easily defined. For example, at Universal Containers, recruiters are generally expected to fill a position within 90 days of it being opened. While we can't choose a single date that will always be 90 days after a position is opened, we *can* define a custom formula that takes the date the position is created and adds 90 days. The platform allows us to specify this formula as the `Hire By` field's default value:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **Edit** next to the `Hire By` field.
4. Next to the `Default Value` text box, click **Show Formula Editor**.

Look familiar? This is similar to the editor that we used to define our `Days Open` custom formula field.

5. From the `Functions` list, double-click `TODAY`.
6. Click the **Insert Operator** button, and choose Add.
7. Type `90`.

Your default value formula should be:

```
TODAY() + 90
```

8. Click **Save**.

It's that easy! Now to wrap up the fields on our Positions tab, let's set the default value of the `Open Date` field to the day that the record is created. To do this, follow these steps again, but use `TODAY()` as the `Default Value`.

Look at What We've Done

Let's revisit the Positions tab to take a look at what we've just done.

1. Click the Positions tab.
2. Click **New**.

Our `Days Open` formula field doesn't show up on our Position edit page—that's because it's a formula field and doesn't require any user input to display. However, we can see that our `Open Date` and `Hire By` fields already have default values: `Open Date` should be today's date, and `Hire By` is 90 days later. We can change these values if we want, or we can just leave them as they are.

In order to see our `Days Open` field, we'll have to define our first position record. Let's do that now.

3. Enter any values you want to define a new position. At the very least, you must enter a value for the required `Position Title` field.
4. Click **Save**.

The new position is now displayed in its own record detail page. At the bottom of the page, notice our `Days Open` formula field, just above the `Created By` field. It should show 0, since we just created the position. If you want to see the value change, edit the record and set the `Open Date` to a week earlier. Isn't that neat?

Introducing Validation Rules

Now that we've defined all the fields that we want on our Position object, let's see if we can articulate a couple of rules about the data that should be entered into those fields. Even though the recruiters and hiring managers at Universal Containers are bright people, everyone sometimes makes mistakes when filling out a form, and a good app should catch the obvious errors.

For example, does it ever make sense for the value of the `Min Pay` field to be more than the value of the `Max Pay` field? Or should `Close Date` ever be unspecified if the `Status` field is set to Closed - Filled or Closed - Not Approved? Clearly not. We can catch these sorts of errors in our app with yet another built-in feature of the platform: *validation rules*.

Validation rules verify that the data a user enters in your app meets the standards that you specify. If it doesn't, the validation rule prevents the record from being saved, and the user sees

an error message that you define either next to the problematic field or at the top of the edit page. Let's build a couple of validation rules now for our Recruiting app.

Note: You can find dozens of sample validation rules on the Salesforce.com Community website at http://blogs.salesforce.com/features/2006/12/data_validation.html.

Try It Out: Defining a Validation Rule for Min and Max Pay

For our first validation rule, let's start off simple: `Min Pay` should never be greater than `Max Pay`:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Validation Rules related list, click **New**.

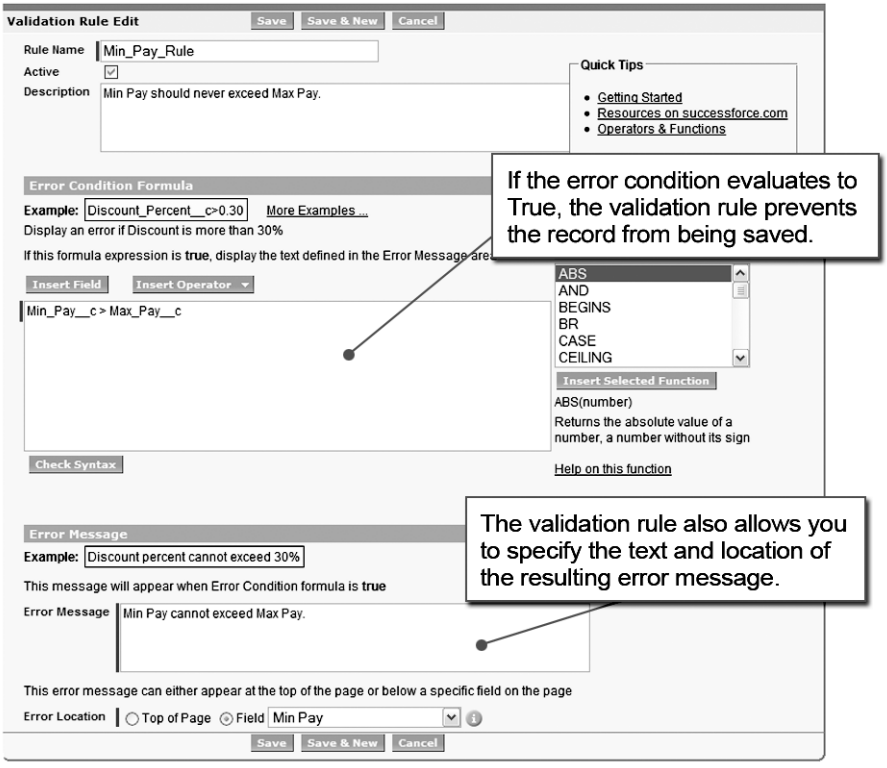

Figure 27: Validation Rule Edit Page

4. In the `Rule Name` text box, enter `Min_Pay_Rule`.

The name of a validation rule can't include any spaces, but if you forget, the platform helpfully changes them to underscores (_) for you.

5. Select the `Active` checkbox.

This checkbox specifies whether the validation rule should start working as soon as it's saved. Because this rule is pretty straightforward (and because we want to test it later!), it makes sense to turn it on right away.

6. In the `Description` text box, enter `Min Pay should never exceed Max Pay`.

Now it's time to define the meat of our validation rule: the *error condition*. If you have a sense of déjà vu when looking at the Error Condition Formula area of the page, don't be alarmed! Just like formula fields and default field values, a validation rule can leverage a number of built-in operators and functions to define a true-or-false error condition that determines whether data is valid. When this condition evaluates to true, an error message displays and the record can't be saved.

We want our error condition to be true whenever `Min Pay` is greater than `Max Pay`, so let's use our formula editor to specify that now:

7. Click the **Insert Field** button. Just like in the formula field editor, two columns appear in an overlay.
8. In the left columns, select Position.
9. In the right columns, select `Min Pay`.
10. Click **Insert**.
11. Click the **Insert Operator** button, and choose Greater Than.
12. Click the **Insert Field** button once again.
13. In the left column, select Position.
14. In the right column, select `Max Pay`.
15. Click **Insert**.

You should now have an error condition formula that looks like this:

```
Min_Pay__c  >  Max_Pay__c
```

Now the only thing that remains is to specify the error message when our error condition evaluates to true.

16. In the `Error Message` text box, enter `Min Pay cannot exceed Max Pay`.

17. Next to the `Error Location` field, select the `Field` radio button, and then choose `Min Pay` from the drop-down list.

Tip: Because our rule only requires a user to correct one field, our error message is displayed next to that field. If a rule requires a user to update multiple fields, it's more appropriate to place the error message at the top of the page.

18. Click **Save**.

Easy! Now that we've familiarized ourselves with a simple validation rule, let's define one that's a little trickier.

Try It Out: Defining a Validation Rule for Close Date

For our next validation rule, let's ensure that `Close Date` has a value whenever the `Status` field is set to Closed - Filled or Closed - Not Approved.

The hardest part of this validation rule is defining the error condition formula. When defining a condition like this, it's sometimes easiest to think about it in logical terms first, and then translate that logic to the functions and operators that are provided in the formula editor. In this case, our error condition is true whenever:

```
Close Date is Not Specified

AND

(Status is "Closed - Filled" OR
           "Closed - Not Approved")
```

Let's start with the first piece: "Close Date is Not Specified." To translate this into terms the formula editor understands, we'll need to use the `ISNULL()` function again. As you might remember from defining the `Days Open` custom formula field, `ISNULL()` takes a single field or expression and returns true if it doesn't contain a value. So, remembering that we have to use the internal field name of the `Close Date` field in our formula, `Close Date is Not Specified` translates to:

```
ISNULL( CloseDate__c )
```

Next, let's figure out how to translate "Status is 'Closed - Filled'." To test for picklist values, we'll need to use another function: `ISPICKVAL()`. `ISPICKVAL()` takes a picklist field name

and value, and returns true whenever that value is selected. So "Status is 'Closed - Filled'" translates to:

```
ISPICKVAL( Status__c , "Closed - Filled")
```

Now we just have to combine these translations with the functions for `AND()` and `OR()`. Both of them take an unlimited number of expressions, and 'AND' or 'OR' them all, respectively. For example:

```
AND ( exp1, exp2, exp3)
```

returns true when *`exp1`*, *`exp2`*, and *`exp3`* are all true. Likewise,

```
OR ( exp1, exp2, exp3)
```

returns true when any one of *`exp1`*, *`exp2`*, or *`exp3`* are true.

Put these functions all together with our other expression translations, and we get our completed error condition formula:

```
AND (
    ISNULL(  CloseDate__c ),
    OR (
         ISPICKVAL(  Status__c , "Closed - Filled"),
         ISPICKVAL(  Status__c , "Closed - Not Approved"))
)
```

Phew! Now we can quickly define our second validation rule using this formula:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Validation Rules related list, click **New**.
4. In the `Rule Name` text box, enter `Close_Date_Rule`.
5. Select the `Active` checkbox.
6. In the `Description` text box, enter `Close Date must be specified when Status is set to 'Closed - Filled' or 'Closed - Not Approved.'`
7. In the Error Condition Formula area, enter the following formula:

```
AND (
    ISNULL(  Close_Date__c ),
    OR (
         ISPICKVAL(  Status__c , "Closed - Filled"),
         ISPICKVAL(  Status__c , "Closed - Not Approved"))
)
```

8. Click **Check Syntax** to make sure you didn't make a mistake.

9. In the `Error Message` text box, enter `Close Date must be specified when Status is set to 'Closed.'`
10. Next to the `Error Location` field, select the `Field` radio button, and then choose `Close Date` from the drop-down list.
11. Click **Save**.

Look at What We've Done

Let's revisit the Positions tab to test the validation rules that we've just made.

1. Click the Positions tab.
2. Click **New**.

First let's try defining a new position with a value for `Min Pay` that's larger than `Max Pay`.

3. Specify any value for the required `Position Title` field.
4. In the `Min Pay` field, enter `80,000`.
5. In the `Max Pay` field, enter `40,000`.
6. Click **Save**.

Did you see what happened? An error message popped up that looked exactly like any other error message in the app!

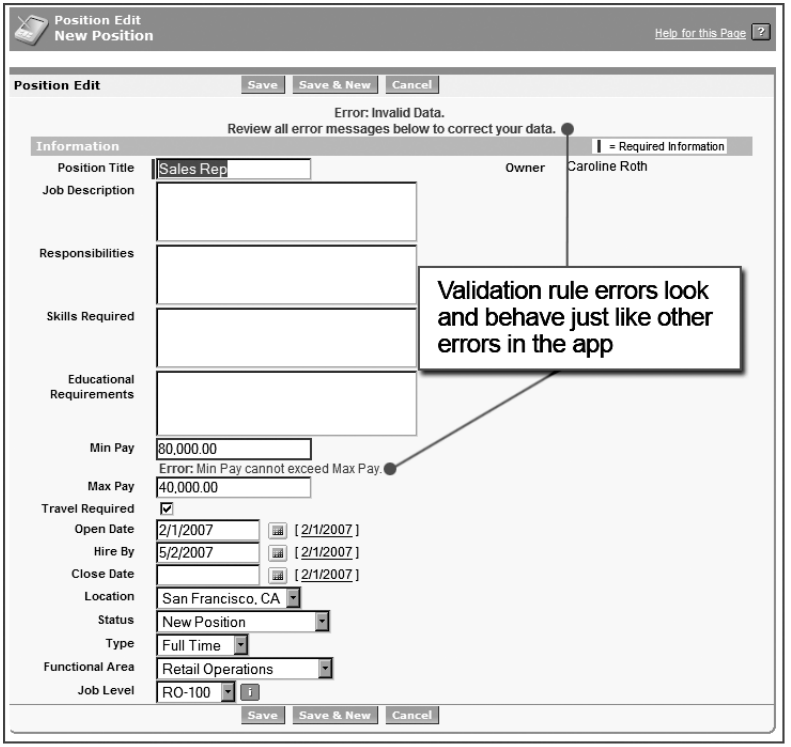

Figure 28: Error Message from a Validation Rule

Now let's test our other validation rule:

7. In the `Min Pay` field, enter `40,000`.
8. In the `Max Pay` field, enter `80,000`.
9. From the `Status` drop-down list, choose Closed - Not Approved.
10. Click **Save**.

Our second validation rule is triggered, this time because we didn't specify a value for `Close Date`. Once we do, the record saves normally.

The Positions tab is now fully functional, with a couple of validation rules to ensure that users don't make certain mistakes. But are the fields where we want them? Are the fields that must have values marked as required? In the next section, we'll fine-tune our Position custom object by modifying its page layout.

Introducing Page Layouts

After defining all those fields and validation rules, we now have a fully functional Position custom object. However, it doesn't look all that nice—all of the long text areas appear at the top, and it's hard to scan. Let's move some things around to make this page easier for our users. We can do that by customizing the Position object's page layout.

A *page layout* controls the position and organization of the fields and related lists that are visible to users when viewing a record. Page layouts also help us control the visibility and editability of the fields on a record. We can set fields as read-only or hidden, and we can also control which fields require users to enter a value and which don't.

Page layouts are powerful tools for creating a good experience for our users, but it's crucial that we remember one important rule: *page layouts should never be used to restrict access to sensitive data that a user shouldn't view or edit*. Although we can hide a field from a page layout, users can still access that field through other parts of the app, such as in reports or via the API. (We'll learn more about security that covers all parts of the app in *Chapter 7: Securing and Sharing Data* on page 127.)

Now let's see if we can organize the fields on our Position object in a way that's more user friendly.

Becoming Familiar with the Page Layout Edit Page

First let's take a look at the Page Layout edit page:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Page Layouts related list, click **Edit** next to the Position Layout.

Welcome to the Page Layout edit page! As you can see, this editor is different from the ones that we've already used in other areas of the platform. That's because we're designing a user interface and need to see a representation of how our page will look as we're working. Before going any further, let's give ourselves a quick orientation to how this page is set up.

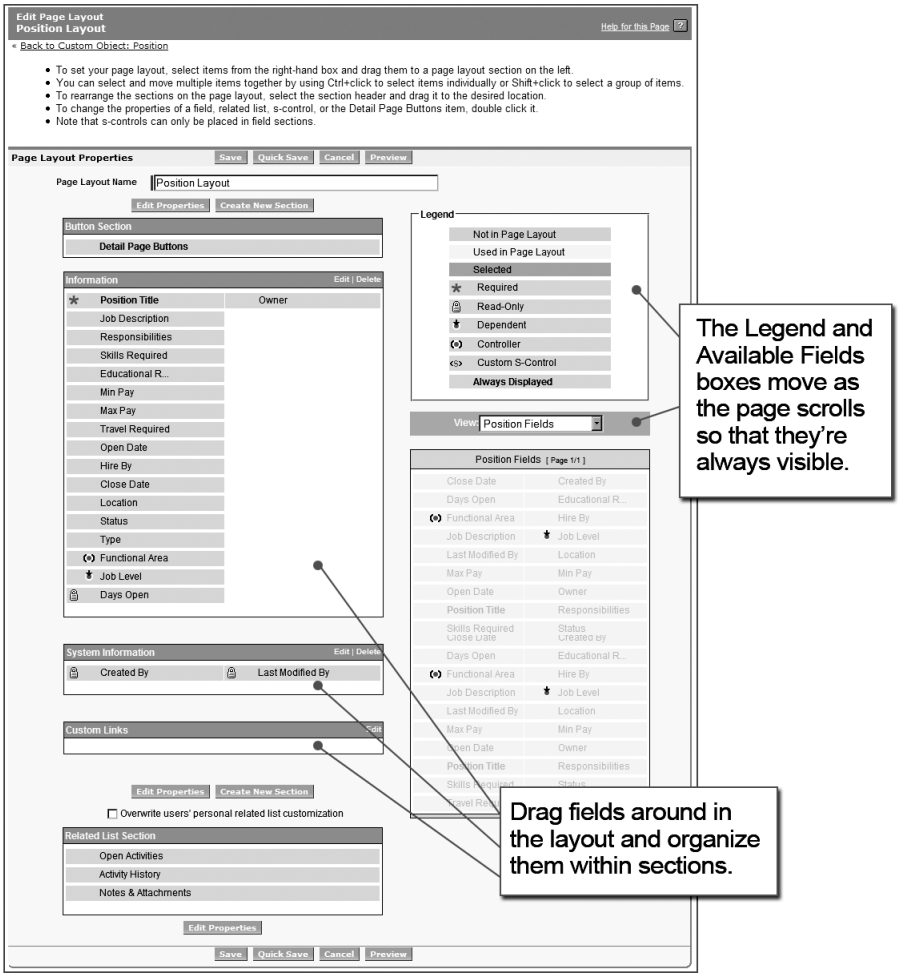

Figure 29: Page Layout Edit Page

On the left side of this page we can see a representation of what our page currently looks like. After an initial set of buttons, the Information section shows all of the fields we've created, plus an additional field for `Owner` that's in its own column. Below it is another section for System Information, then Custom Links, and finally, Related Lists.

As we scroll down to view the entire page, the boxes on the right side move with us so that they're always visible. These boxes include a legend for all of the icons we see on the left, plus a list of all of the Position fields, related lists, and other components that are available to put on our layout. Currently all of the fields in the box on the right are grayed out—that's because they've all been placed on our page layout by default. If we were to move one of the fields that's

in the layout to the left back to the Position fields box on the right, that field would effectively be removed from the layout, and its name would no longer be grayed out.

Now that we know what we're looking at, let's rearrange the fields in the way a user might want to see them.

Try It Out: Grouping Fields into a New Section

Let's start modifying our page layout by first defining a new section for salary information. On a page layout, a section is simply an area where we can group similar fields under an appropriate heading. This makes it easy for our users to quickly identify and enter the information for a record, especially if our object has a large number of fields:

1. Click **Create New Section**.
2. In the `Name` text box, enter `Compensation`.

The `Name` field controls the text that's displayed as the heading for the section.

3. In the `Columns` drop-down list, choose 2 (Double).

This option allows us to choose whether we want the fields in our section to be arranged in two columns or one. The default is two columns and is the most commonly-used choice. However, if our section is going to contain text area fields, the one-column layout gives them more space on the page for display.

4. In the `Tab Order` drop-down list, choose Left-Right.

This setting controls the direction that a user's cursor will move when using the Tab key to navigate from field to field.

5. Select the options for `Show Section Heading on Detail Page` and `Show Section Heading on Edit Page`.
6. Click **OK**.

Voilà! We have a new section for Compensation just under the Custom Links related list. Let's move it above the System Information section and add the `Min Pay` and `Max Pay` fields:

7. Click the heading of the Compensation section and drag it above the System Information section.
8. Now use drag-and-drop to move the `Min Pay` and `Max Pay` fields from the Information section to the new Compensation section, as shown in the following screenshot.

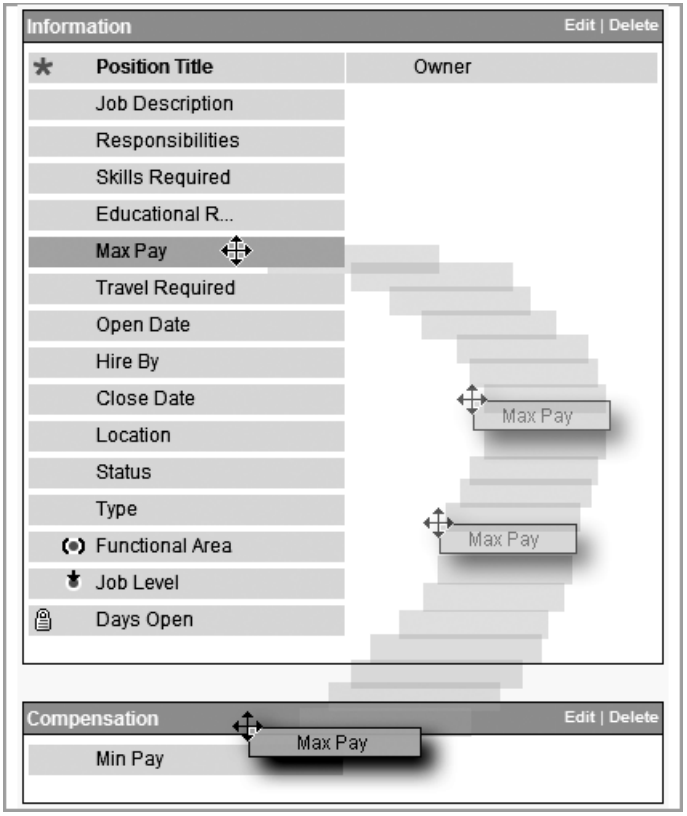

Figure 30: Dragging and Dropping Fields in a Page Layout

Now that we've gone through the process for building one section, let's build two more:

9. Create a new one-column Description section below the Compensation section, and drag `Job Description`, `Responsibilities`, `Skills Required`, and `Educational Requirements` into it.
10. Create a new two-column Required Languages section below the Description section, and drag `Apex`, `C#`, `Java`, and `JavaScript` into it.

Tip: You can use SHIFT+click or CTRL+click to select all of these fields together and then drag them as one unit into the Description section.

Finally, to finish up our layout, let's reorganize the fields in the Information section so they're more readable.

11. In the first column, arrange these fields as follows:

- `Position Title`
- `Status`
- `Type`
- `Functional Area`
- `Job Level`
- `Travel Required`
- `Hiring Manager`
- `Created By`

12. In the second column, arrange these fields as follows:

- `Owner`
- `Location`
- `Open Date`
- `Hire By`
- `Close Date`
- `Days Open`
- `Last Modified By`

That's much better—our fields are organized, and it's easy to locate all of the information we need. Now all we have left to do is make the `Min Pay` and `Max Pay` fields required when a user defines a new position. Once we do that, we'll be all done with our Position object!

Try It Out: Editing Field Properties

Let's make the `Min Pay` and `Max Pay` fields required:

1. On the Page Layouts edit page, double-click the `Min Pay` field.

This popup window allows us to edit the `Min Pay` field's properties. We can set the field to read-only and/or required:

- If it's read-only, a user who views a position record edit page won't be able to change its value.
- If it's required, a user won't be able to create a position record without specifying a value.

If we didn't want a user to see the `Min Pay` field at all, we could simply drag it off the layout and onto the Position Fields box on the right side.

Caution: Don't forget our earlier warning! Page layouts should never be used to restrict access to sensitive data that a user shouldn't view or edit. That's because page layouts control only a record's edit and detail pages; they don't control access to fields in any other part of the platform.

2. Select the `Required` checkbox, and click **OK**.
3. Repeat these steps for the `Max Pay` field.
4. Click **Save** to finish customizing the page layout.

Hooray! We're all done with our Position object's page layout.

Look at What We've Done

Congratulations. We've just built ourselves a simple Recruiting app that tracks details about an organization's open positions. Let's check out what we've done by revisiting the Positions tab and clicking **New**. Because of the changes that we've made in our page layout, our Position edit page should now look like this:

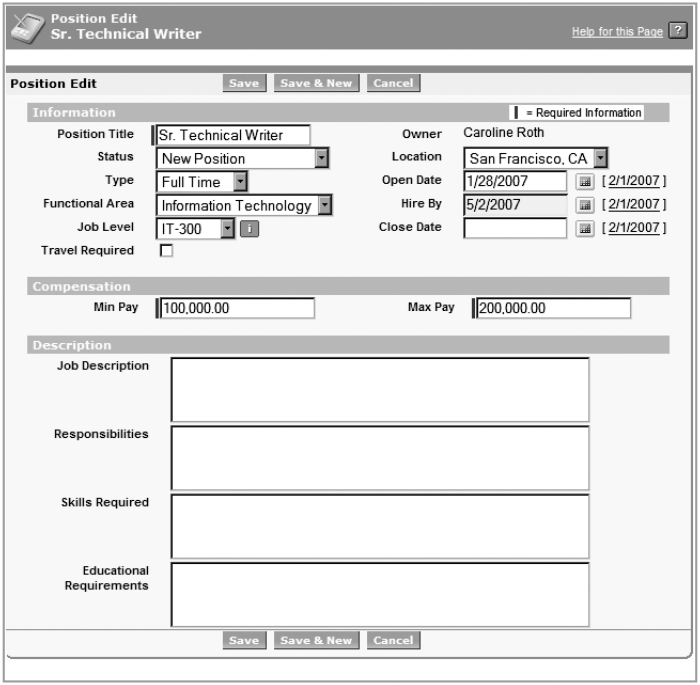

Figure 31: Final Version of the Position Edit Page

We have an object with a tab, we've added custom fields, and we've arranged them in a page layout. We've finished our simple app, and now we're well on our way to creating the more complex Recruiting app that we described earlier.

Things are going to get even more interesting in the next chapter. We'll add a few more custom objects to track things like candidates, job applications, and reviews, and then we'll enhance our Recruiting app even further by defining how our objects relate to one another. Before you know it, we're going to have an incredibly powerful tool, all implemented with a few clicks in the platform.

Chapter 6

Expanding the Simple App Using Relationships

In this chapter ...

- Introducing Relationships
- Introducing Relationship Custom Fields
- Adding Candidates to the Mix
- Bringing Candidates and Positions Together with Job Applications
- Introducing Search Layouts
- Managing Review Assessments
- Creating a Many-to-Many Relationship
- Putting it All Together

So far we've accomplished a fair amount—we've created the Recruiting app and built out a fully functional Position custom object with a tab and several types of fields. It's a good start, but there's more to do.

Having just one object in our Recruiting app is like having a party with just one guest—not all that interesting! We need to invite more "people" to the party by building custom objects to represent candidates, job applications, and reviews, and, even more importantly, we need to create *relationships* between them. Just like a party isn't all that fun if you don't know any of the other guests, an app isn't all that powerful unless its objects have links to other objects in the app. That's going to be the focus of this chapter, so let's get started!

Introducing Relationships

So what is a relationship, and why are they important for our app? Just as a personal relationship is a two-way association between two people, in terms of relational data, a relationship is a two-way association between two objects. Without relationships, we could build out as many custom objects as we could think of, but they'd have no way of linking to one another.

For example, after building a Position object and a Job Application object, we could have lots of information about a particular position and lots of information about a particular candidate who's submitted an application for it, but there would be no way of seeing information about the job application when looking at the position record, and no way of seeing information about the position when looking at the job application record. That's just not right!

With relationships, we can make that connection and display data about other related object records on a particular record's detail page. For example, once we define a relationship between the Position and Job Application objects we just talked about, our position record can have a related list of all the job applications for candidates who have applied for the position, while a job application record can have a link to the positions for which that candidate is applying. Suddenly the "people" at our Recruiting app "party" know some of the other guests, and the app just got a lot more interesting.

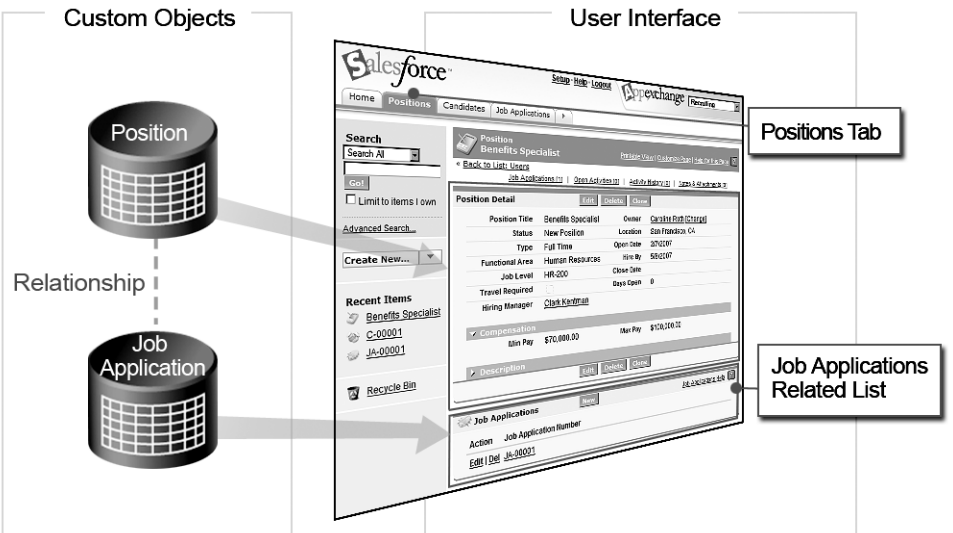

Figure 32: Relationships Allow Information about Other Object Records to be Displayed on a Record Detail Page

Introducing Relationship Custom Fields

As we learned in *Chapter 3: Reviewing Database Concepts* on page 25, we can define a relationship between two objects through the use of common fields. On the platform, we can define relationships between objects by creating a *relationship* custom field that associates one object with another. A relationship field is a custom field on an object record that contains a link to another record. When we place a relationship custom field on an object, we're effectively creating a many-to-one relationship between the object on which the relationship field is placed and the other object.

There are different types of relationship fields, each with different implications. The simplest and most flexible type is a *lookup relationship* field, which creates a simple relationship between two objects. For example, if we place a lookup relationship field on a Job Application object that references position records, many job application records can be related to a single position record. This will be reflected both with a new `Position` field on the job application record and with a new Job Applications related list on the position record. You can also put multiple lookup relationship fields on a single object, which means that our Job Application object can also point to a Candidate object.

A second type of relationship field, *master-detail relationship*, is a bit more complex, but more powerful. Master-detail relationships create a special parent-child relationship between objects: the object on which you create the master-detail relationship field is the child or "detail," and the object referenced in the field is the parent or "master." In a master-detail relationship, the ownership and sharing of detail records are determined by the master record, and when you delete the master record, all of its detail records are automatically deleted along with it. Master-detail relationship fields are always required on detail records, and once you set a master-detail relationship field's value, you cannot change it.

When do you use a master-detail relationship? If you have an object that derives its significance from another object. For example, say you have a Review custom object that contains an interviewer's feedback on a job application. If you delete a job application record, you will probably want all of its review records deleted as well, being that reviews of something that no longer exists aren't very useful. In this case, you want to create a master-detail relationship on the Review custom object with the Job Application object as the master object.

That's the sort of thing that we're going to do in this chapter. First, let's start with the really quick and easy example of putting a `Hiring Manager` field on our Position object—we'll create a many-to-one relationship between the Position object and the standard User object that comes with every organization, reflecting the fact that a hiring manager can be responsible for several positions at a time. Then we'll build out a few more objects and implement a more complex relationship involving positions, job applications, candidates, and reviews.

Try It Out: Relating Hiring Managers to Positions

For our first relationship, let's associate a hiring manager with a position by putting a lookup relationship field on the Position object. The lookup field will allow users to select the hiring manager for the position by selecting from all the users of the Recruiting app.

For example, if Ben Stuart, our recruiter, wants to assign Anastasia O'Toole as the hiring manager for the Benefits Specialist position, he'll be able to do so by clicking the lookup icon () next to the lookup relationship field that we are going to create. Her name will then appear on the Position detail page.

To create the lookup relationship field that accomplishes this, we'll need to go back to the now familiar Position object detail page.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select `Lookup Relationship`, and click **Next**.
5. In the `Related To` drop-down list, choose User, and click **Next**.

As we've mentioned, User is a standard object that comes with all organizations on the platform. It contains information about everyone who uses the app in your organization.

6. In the `Field Label` text box, enter `Hiring Manager`. Once you move your cursor, the `Field Name` text box should be automatically populated with Hiring_Manager.
7. Click **Next**.
8. Accept the defaults in the remaining two steps of the wizard.
9. Click **Save**.

Look at What We've Done

Now return to the Positions tab, and click **New**. The Position edit page includes a new `Hiring Manager` lookup field! If you click the lookup icon next to this field (), you can search through all of the users of the Recruiting app and select one as the hiring manager. That user's name now appears on the position record:

Figure 33: Hiring Manager Lookup Relationship

As you can see, it was easy to set up this simple relationship between positions and users. And as a general rule, you'll find that relationships are pretty easy to set up.

What gets a little tricky is when we start wanting to create relationships that don't represent a simple many-to-one relationship. We'll see an example of one of those in a little bit. Right now, let's build a custom object for candidates so we'll be able to create some more relationships in our Recruiting app.

Adding Candidates to the Mix

Let's add a Candidate custom object to our app so we can manage the information about our candidates. We'll also add fields to the object, modify the page layout properties, and create a candidate record. The process for creating the Candidate custom object is almost identical to the one we followed to create the Position custom object, so we'll zip through this quickly.

Try It Out: Creating the Candidate Object

To create our Candidate custom object, navigate back to **Setup ➤ Create ➤ Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 8: Values for Defining the Candidate Object

Field	Value
Label	Candidate
Plural Label	Candidates

Field	Value
Object Name	Candidate
Description	Represents an applicant who might apply for one or more positions
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Candidate Number
Data Type	Auto Number
Display Format	C-{00000}
Starting Number	00001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	Yes

To create the Candidates tab, select a `Tab Style` in the first step of the wizard, and then accept all the defaults until you get to the Add to Custom Apps page. On this page, select only the Recruiting App and click **Save**.

The Recruiting app now has three tabs: Home, Positions, and Candidates. Now let's add some custom fields to the Candidate object.

Try It Out: Adding Fields to the Candidate Object

To create custom fields on the Candidate object, click **Setup ➤ Create ➤ Objects**, and click **Candidate** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise, you can simply accept all defaults.

One difference you'll see in the Candidate object fields is that three of them—`First Name`, `Last Name`, and `Email`—have the `External ID` option selected. This option allows the values in these fields to be indexed for search from the sidebar of the application. If we didn't select these values as external IDs, we'd only be able to search for records based on the `Candidate Number` field. Setting the `Email` field as an external ID is also going to help us with importing data a little later in this chapter.

Table 9: Candidate Object Custom Fields

Data Type	Field Label	Other Values
Text	First Name	Length: 50 External ID: Selected
Text	Last Name	Length: 50 External ID: Selected
Phone	Phone	
Email	Email	External ID: Selected
Text	Street	Length: 50
Text	City	Length: 50
Text	State/Province	Length: 50
Text	Zip/Postal Code	Length: 15
Text	Country	Length: 50
Text	Current Employer	Length: 50
Number	Years of Experience	Length: 2 Decimal Places: 0
Text	SSN	Length: 9
Picklist	Education	Picklist values: • HS Diploma • BA/BS • MA/MS/MBA • Ph.D. • Post Doc

Data Type	Field Label	Other Values
Checkbox	Currently Employed	Default: Checked
Checkbox	US Citizen	Default: Checked
Checkbox	Visa Required	Default: Unchecked
Phone	Mobile	
Phone	Fax	

Try It Out: Modifying the Candidate Page Layout Properties

To finish up with this object, let's organize all of our fields on the page layout and mark some fields as required. To do so, let's go to the Page Layout Properties page.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Candidate**.
3. In the Page Layouts related list, click **Edit** next to the Candidate Layout.
4. Create three new double-column sections below the Information section: Address, Employment, and Additional Details. Drag the appropriate fields into them, as shown in *Figure 34: Candidate Object Page Layout* on page 93, and don't forget to click **Quick Save** so you can save your work as you go.
5. Set the `First Name`, `Last Name`, and `Email` fields to required as follows:
    a. Use CTRL+click to select all three required fields.
    b. Click the **Edit Properties** button.
    c. Select the `Required` checkbox in the Select All row, and click **OK**.
6. Click **Save**.

Your Page Layout Properties page should now look similar to the following screenshot.

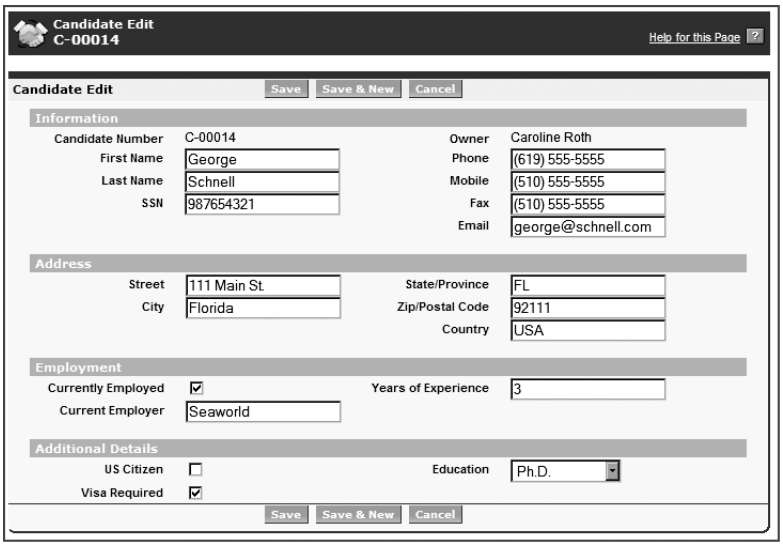

Figure 34: Candidate Object Page Layout

Look at What We've Done

Here's a quick way to verify that you did everything correctly.

1. Click the Candidates tab.
2. Click **New**.
3. Create a new record for a candidate named Ethan Tran.
4. Enter a value for each of the required fields. Salesforce will not verify the email address you enter in the `Email` field right now, so feel free to enter a fictitious one.
5. Click **Save**.

How does the page layout look? Are the fields where you want them? If you were able to successfully create a new candidate record, and everything looks okay, let's move on to the Job Application object!

Bringing Candidates and Positions Together with Job Applications

Our app can track candidates and open positions, but there's a crucial element that's missing: how do we know which candidates are interested in which positions? We can create lookup relationship fields on the Candidate object that let recruiters specify the positions in which the

candidate is interested, but what if we want to track additional information, such as whether the candidate is currently scheduled to interview for one of those positions? And wouldn't it be helpful if the recruiter has a way of storing the cover letters that candidates tailor for each specific job to which they are applying?

We can satisfy these requirements with a Job Application custom object that stores data about an individual candidate's application to a single position. Each time a candidate wants to apply for a position, the recruiter can create a job application record that contains the candidate's name and the position to which he or she is applying, as well as any cover letter that the candidate may have submitted specifically for that position. Recruiters will also be able to indicate the status of the candidate's application, such as whether he or she is scheduled for an interview or if the application has been rejected. After we create the Job Application object and its fields, we'll make a few small modifications to the Position, Candidate, and Job Application objects so that each position record displays the names of the candidates who have applied to it, and each candidate record displays the name of the positions to which the candidate has applied.

Try It Out: Creating the Job Application Object

You should be a pro at this by now! To create our Job Application custom object, navigate back to **Setup ➤ Create ➤ Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 10: Values for Defining the Job Application Object

Field	Value
Label	Job Application
Plural Label	Job Applications
Object Name	Job_Application
Description	Represents a candidate's application to a position
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Job Application Number
Data Type	Auto Number
Display Format	JA-{00000}

Field	Value
Starting Number	00001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	Yes

To create the Job Applications tab, select a `Tab Style` in the first step of the wizard, and then accept all the defaults until you get to the Add to Custom Apps page. On this page, select only the Recruiting app, and then click **Save**.

We're now just a few custom fields away from linking the Job Application object with the Position and Candidate objects.

Try It Out: Adding Fields to the Job Application Object

Here's another procedure that we've done several times before, but this time we only need to define four custom fields instead of the nearly twenty that we built for the Candidate object. We'll need to add a text field for the candidate's cover letter, a picklist field so that we can track the application's status, and two lookup relationship fields that will create relationships between the Job Application object and the Position and Candidate objects.

Although these fields are almost identical to the ones we created earlier, you'll notice when you're defining the lookup relationship fields that there's a new step in the custom field wizard *Step 6: Add Custom Related Lists*. This step of the wizard is where we can specify a heading for the Job Applications related list that will show up on both the Candidate and Position detail pages.

Why didn't we see this step earlier when we created our `Hiring Manager` lookup field? It turns out that User is a unique standard object: it doesn't have a tab, and you cannot add related lists to it. The platform knows this, so it leaves out the related list step whenever someone adds a lookup relationship field that references the User object.

Now that we're all squared away with that small difference, let's finish up these Job Application fields. Click **Setup ➤ Create ➤ Objects**, and then click **Job Application** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise you can simply accept all defaults.

Table 11: Add Custom Fields to the Job Application Object

Data Type	Field Label	Other Values
Lookup Relationship	Candidate	Related To: Candidate Related List Label: Job Applications
Lookup Relationship	Position	Related To: Position Related List Label: Job Applications
Text Area (Long)	Cover Letter	Length: 32,000 # of Visible Lines: 6
Picklist	Status	Picklist values: • New • Review Resume • Phone Screen • Schedule Interviews • Extend an Offer • Hired • Rejected Use first value as default value: Selected

Before we move on, there is one more important step: we need to enter a name for the child relationship that the **Position** field created between the Job Application and Position objects. This step is only necessary for the `Position` relationship field because of how we will use this relationship when we create our candidate map feature in *Chapter 10: Moving Beyond Native Apps* on page 265.

To name the child relationship:

1. Click **Setup ➤ Create ➤ Objects**, and click **Job Application**.
2. In the Custom Fields & Relationships related list, click Positions, then click **Edit**.
3. In the `Child Relationship Name` field, enter `Job Applications` and click **Save**. The Force Platform saves the child relationship name as `Job_Applications`.

Look at What We've Done

Tada! If you click on the new Job Applications tab and click **New**, you'll see the `Candidate` lookup field, a `Position` lookup field, the candidate's cover letter, and a `Status` picklist field.

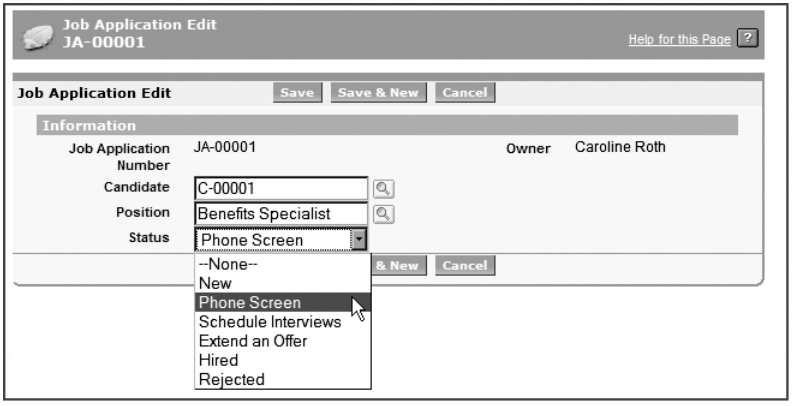

Figure 35: Custom Fields on the Job Application Edit Page

But there's more! Because we've built a couple of lookup relationships, our candidate and position record detail pages now each have a new Job Applications related list. And the Job Application detail page includes links to the candidate and position records that it references. All three objects are now related and linked to one another!

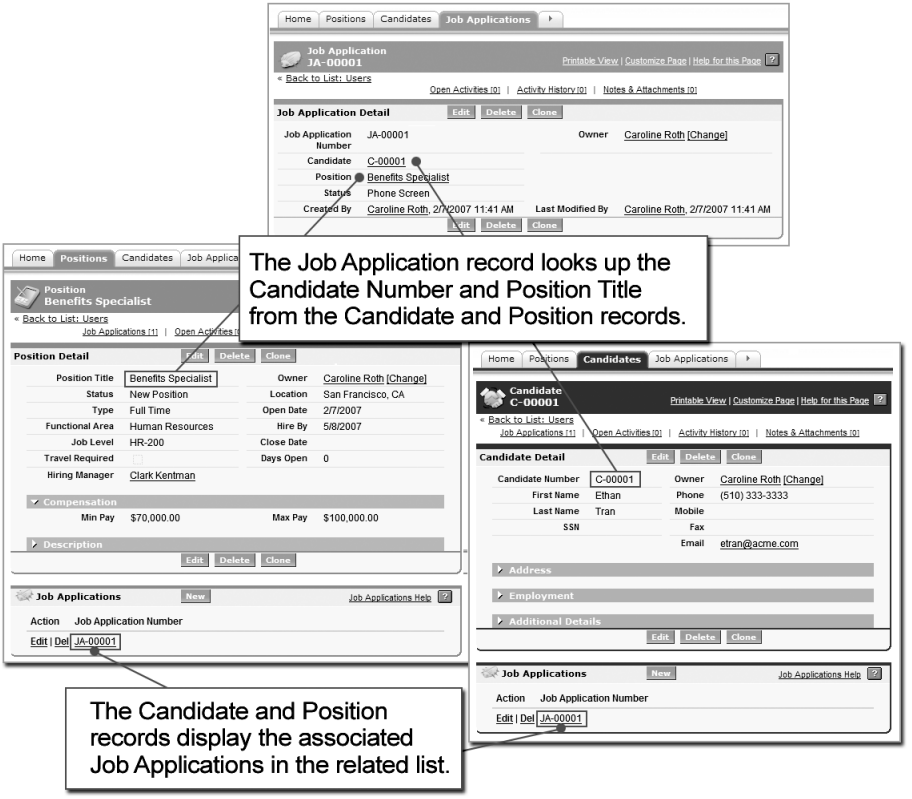

Figure 36: Job Application Links to Position and Candidate Data

Before we move on, let's see if we can clean up the usability of our app a bit more so our users don't have to identify candidates and job applications by number when they click the lookup button in the Job Application edit page, or when they look at the Job Applications related list on the Candidate or Position detail pages.

Introducing Search Layouts

By default, all lookup dialogs and related lists that result from new relationships, such as the ones we've defined in this chapter, only display the record name or number. For example, if you go ahead and create a job application, you might find the Candidate lookup dialog a little cryptic because the only listed field is `Candidate Number`, as shown in the following screenshot.

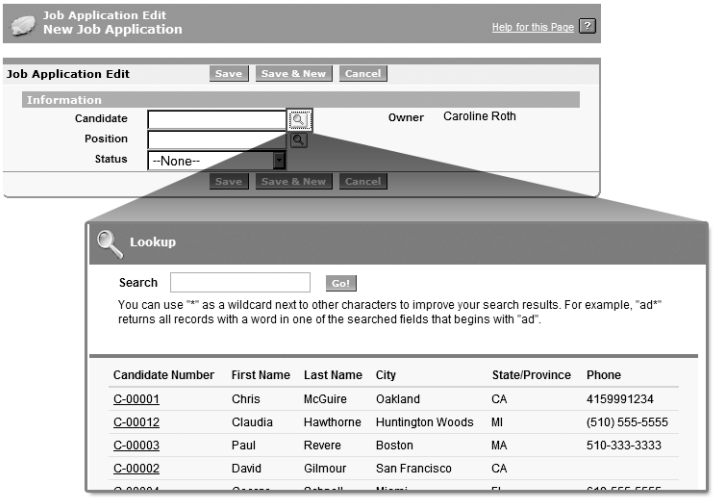

Figure 37: Default Candidate Lookup on the Job Application Object

Likewise, the Job Applications related lists on the Position and Candidate detail pages only display a job application number. It is much more useful if these related lists also include the associated candidate's name or position.

To fix these issues, we can add fields to the *search layouts* for the objects that we've defined. Search layouts are ordered groups of fields that are displayed when a record is presented in a particular context, such as in search results, a lookup dialog, or a related list. By adding fields, we can give users more information and help them locate records more quickly.

The Search Layouts related list on the custom object detail page is the place to modify these sets of fields. Go to **Setup ➤ Create ➤ Objects** and select the Candidate object. You'll see that the available search layouts include the following:

Table 12: Available Search Layouts

Layout Name	Description
Search Results	The search results that originate from searching in the left Sidebar Search of the application or in Advanced Search.
Lookup Dialogs	The lookup dialog results that originate from clicking the button next to a lookup field on an edit page.
Tab	The list of recent records that appears on the home page of a tab, and in related lists on other object detail pages.
Search Filter Fields	The filters that can be applied to search results.

Note: The List View layout also appears in the Search Layouts related list, but it's not for specifying fields. Instead, it allows you to specify the buttons that appear on the list view page for an object.

Try It Out: Adding Fields to the Candidate Lookup Dialog

Let's add fields to our Candidate lookup dialog:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Candidate**.
3. In the Search Layouts related list, click **Edit** next to the Lookup Dialogs layout.

The Edit Search Layout page includes a list of available fields from the Candidate object. You can choose up to ten fields to include in the lookup dialog, and order them in any way you choose, except that the object's unique name or number field (such as `Candidate Number`) must be listed first.

4. Move the following fields into the Selected Fields box under `Candidate Number`:
   - `First Name`
   - `Last Name`
   - `City`
   - `State/Province`
   - `Phone`
5. Click **Save**.

That's it! To try it out, return to the Job Applications tab, and click **New**. When you click the lookup icon next to the `Candidate` field, the dialog is now much more useful.

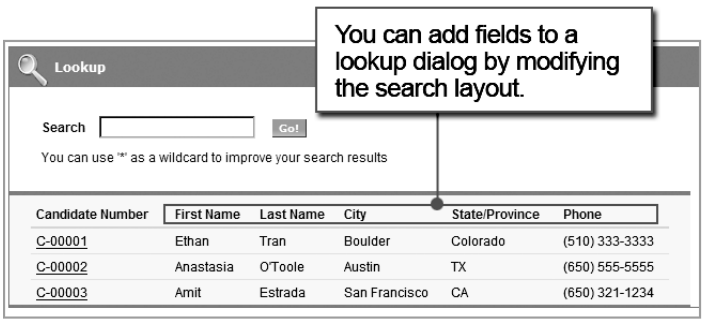

Figure 38: Modified Candidate Lookup on the Job Application Object

Try It Out: Updating Additional Search Layouts

Now that we've updated one search layout for lookups, the rest should be easy. Use the Search Layouts related list on the custom object detail page to modify the other search layouts as described in the following table.

Table 13: Additional Search Layouts

Object	Search Layout	Add These Fields
Candidate	• Search Results • Candidates Tab	• Candidate Number • First Name • Last Name • City • State/Province • Phone
Candidate	• Search Filter Fields	• Candidate Number • First Name • Last Name • Education • Years of Experience • City • State/Province • Country • Currently Employed
Position	• Search Results • Lookup Dialogs • Positions Tab • Search Filter Fields	• Position Title • Location • Functional Area • Job Level • Type • Hiring Manager • Status • Open Date • Close Date

Object	Search Layout	Add These Fields
Job Application	• Search Results • Lookup Dialogs • Job Applications Tab • Search Filter Fields	• Job Application Number • Candidate • Position • Status • Created Date • Owner First Name • Owner Last Name

Now let's create one more custom object to provide our hiring managers and interviewers with a place to enter their comments about job applications.

Managing Review Assessments

Interviewers, recruiters, and hiring managers need to be able to create reviews so that they can record their comments about each candidate's job application, and rate the candidate's suitability for the position. They also need to see the reviews posted by other people. To allow our users to perform these tasks, we'll need to create a custom Review object and relate it to the Job Application object.

The Review object has a many-to-one relationship with the Job Application object because one job application can have one or more reviews associated with it. A related list on the job application record will show the associated reviews, representing the "many" side of the relationship.

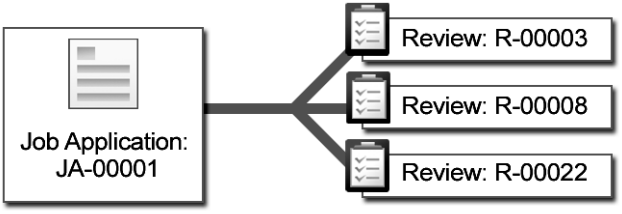

Figure 39: Review Has a Many-to-One Relationship with Job Application

However, instead of creating this relationship with a lookup relationship field, this time we'll use a master-detail relationship field. A master-detail relationship field makes sense in this case because reviews lose their meaning when taken out of the context of a job application, so

we'll want to automatically delete reviews when we delete the job application to which they're related.

Try It Out: Creating the Review Object

To create the Review object, navigate back to **Setup ➤ Create ➤ Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 14: Values for Defining the Review Object

Field	Value
Label	Review
Plural Label	Reviews
Object Name	Review
Description	Represents an interviewer's assessment of a particular candidate
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Review Number
Data Type	Auto Number
Display Format	R-{000000}
Starting Number	000001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	No

Notice that we didn't launch the tab wizard this time. Reviews don't need a tab of their own because they can be accessed via a related list on the Job Application detail page. When you

create an object with a tab, the platform provides access to that object's records in various places other than just the tab, such as in search results and the Recent Items list in the sidebar area of every page. Because most Recruiting app users won't need to see reviews unless it's in the context of a job application, we don't need to create a separate tab for them.

Now let's finish up the custom fields on the Review object.

Try It Out: Adding Fields to the Review Object

Let's start out by adding the master-detail relationship field, which will relate our Review object with the Job Application object. To create the master-detail relationship field, access the Review object detail page.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Review**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select `Master-Detail Relationship`, and click **Next**.
5. In the `Related To` drop-down list, choose Job Application, and click **Next**.
6. In the `Field Label` text box, enter `Job Application`.

Notice that the `Required` checkbox is automatically selected and cannot be changed. As mentioned earlier, master-detail relationship fields are always required on detail records.

7. Select the `Read/Write` radio button.

This sharing setting prevents people from creating, editing, or deleting a review unless they can also create, edit, or delete the associated job application. We'll learn all about sharing and security in the next chapter.

8. Click **Next**.
9. Accept the defaults in the remaining three steps of the wizard.
10. Click **Save**.

Your master-detail relationship is complete! Now that it's in place, let's think about the other types of fields that would be useful to people looking at a review record.

Most likely, you are going to want to see the name of the candidate and the position for which they are being reviewed. We could create a lookup relationship to the Position and Candidate objects, and then require reviewers to enter those fields when creating a review record, but what if they select the wrong value? Besides, wouldn't it be better if these fields were somehow automatically populated?

To solve this, we'll tap into the synergy of formulas and relationships to create *cross-object formulas*. Cross-object formulas are formulas that span two or more objects by referencing merge fields from related records. This means that formulas on our Review object can access fields on the Job Application object, and formulas on the Job Application object can access fields on both the Position and Candidate objects. We're going to take it even one step further by creating formula fields on our Review object that *span* the Job Application object to reference fields on the Candidate and Position objects. You'll quickly discover that using related data is much easier than it sounds!

Let's begin by building a formula field on the Review object that references the title of the position on the review's parent job application record.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Review**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Formula` data type, and click **Next**.
5. In the `Field Label` field, enter `Position`. Once you move your cursor, the `Field Name` text box should be automatically populated with Position.
6. Select the `Text` formula return type and click **Next**.
7. Click the **Insert Field** button.
8. Select Review in the first column.

When you choose Review, the second column displays all of the Review object's fields as well as its related objects, which are denoted by a greater-than sign (>). Notice that the `Created By` and `Last Modified By` fields also have greater-than signs. This is because these are lookup fields to the User object.

9. Select `Job Application >` in the second column. The third column displays the fields of the Job Application object.
10. Select `Position >` in the third column. The fourth column displays the fields of the Position object.

Be sure that you select `Position >` (with the greater than sign) and not `Position`. The one with the greater-than sign is the Position object, while the one without the greater than sign is the `Position` lookup field on the Job Application object. In most cases, formulas that access lookup fields return a cryptic record ID. Instead, we want our formula to return the position's title.

11. Choose `Position Title` in the fourth column.
12. Click **Insert**.

Your formula now looks like this:

```
Job_Application__r.Position__r.Name
```

The formula spans to the review's related job application (`Job_Application__r`), then to the job application's related position (`Position__r`), and finally references the position's title (`Name`). Notice that each part of the formula is separated by a period, and that the relationship names consist of the related object followed by `__r`.

13. Click **Next**.
14. Accept all remaining field-level security and page layout defaults.
15. Click **Save**.

That wraps up our first cross-object formula field. Let's try another. This time, we'll add a cross-object formula field on our Review object that displays the first and last names of the candidate being reviewed. We'll also up the ante by using the `HYPERLINK` function so that users can access the candidate's record by clicking the field.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Review**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Formula` data type, and click **Next**.
5. In the `Field Label` field, enter `Candidate`. Once you move your cursor, the `Field Name` text box should be automatically populated with Candidate.
6. Select the `Text` formula return type and click **Next**.
7. From the `Functions` list, double-click `HYPERLINK`.

The `HYPERLINK` function lets you create a hyperlink to any URL or record in Salesforce. The text of the hyperlink can differ from the URL itself, which is useful here because we want our hyperlink to display the first and last names of the candidate while the URL points to the candidate record itself.

8. Delete *`url`* from the `HYPERLINK` function you just inserted, but leave your cursor there.
9. Click the **Insert Field** button, and select `Review >`, `Job Application >`, `Candidate >`, `Record ID`, and click **Insert**.

Salesforce generates a unique ID for every record. By inserting the record ID of the candidate in our `HYPERLINK` function, we're enabling our formula field to locate and link to the candidate's record.

10. Delete *`friendly_name`* from the `HYPERLINK` function, but leave your cursor there.

11. Click the **Insert Field** button, and select `Review >`, `Job Application >`, `Candidate >`, `First Name`, then click **Insert**.
12. Enter a space, then click the **Insert Operator** button and choose Concatenate.

The Concatenate operator inserts an ampersand (&) in your formula, and joins the values on either side of the ampersand. Here we're going to use the Concatenate operator to join the first and last names of the candidate in a single field, even though they are stored in separate fields on the Candidate object. The Concatenate operator also lets us insert a space between the two names, as you'll see in the next step.

13. Enter another space, then type a blank space enclosed in quotes, like this:

    ```
    " "
    ```

    This appends a blank space after the first name of the candidate.
14. Enter a space, then click the **Insert Operator** button and choose Concatenate once more to add a second ampersand in your formula.
15. Click the **Insert Field** button, and select `Review >`, `Job Application >`, `Candidate >`, `Last Name`, then click **Insert**.
16. Delete *`[ target ]`* from the `HYPERLINK` function. This is an optional parameter that isn't necessary for our formula field.
17. Click **Check Syntax** to check your formula for errors. Your finished formula should look like this:

    ```
    HYPERLINK
         ( Job_Application__r.Candidate__r.Id ,
         Job_Application__r.Candidate__r.First_Name__c
         &
         " "
         &
         Job_Application__r.Candidate__r.Last_Name__c )
    ```

18. Click **Next**.
19. Accept all remaining field-level security and page layout defaults.
20. Click **Save**.

Whew! That one required a little more thought, but using a bit of brainpower here has tremendously improved the usability of our app, which you'll see in a moment when we test our changes to the Review object. Before we start testing, though, let's quickly add two more easy fields to finish our Review object. We need a text area field for the reviewer's assessment, and a number field in which the reviewer can give the candidate a numeric score.

Go to **Setup ➤ Create ➤ Objects** and select the Review object. Use the **New** button in the Custom Fields & Relationships related list to create the remaining custom fields for the Review

object according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise, accept all defaults.

Table 15: Add Custom Fields to the Review Object

Data Type	Field Label	Other Values
Text Area (Long)	Assessment	Length: 32,000 # of Visible Lines: 6
Number	Rating	Length: 1 Always require a value in this field in order to save a record Help text: `Enter a 1-5 rating of the candidate.`

When you're done, add a quick validation rule to ensure that the Ratings field only accepts the numbers 1 through 5. This will keep our review rating system consistent throughout our organization.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Review**.
3. In the Validation Rules related list, click **New**.
4. In the `Rule Name` text box, enter `Rating_Scale_Rule`.
5. Select the `Active` checkbox.
6. In the `Description` text box, enter `Rating must be from 1 to 5.`
7. Enter the following error condition formula:

   ```
   (Rating__c < 0) || (Rating__c > 6)
   ```

   This formula prevents the record from being saved if the value of the `Rating` field is less than one or greater than five.
8. In the `Error Message` text box, enter `Invalid rating. Rating must be from 1 to 5.`
9. Next to the `Error Location` field, select the `Field` radio button, and then choose `Rating` from the drop-down list.
10. Click **Save**.

Our Review object is complete! We've added several features that will help users access the data they need in order to asses each job application. There's just one more easy improvement we need to streamline our job application review process. It involves returning to our Job Application object and taking advantage of one of the benefits we gain by using a master-detail relationship.

Introducing Roll-Up Summary Fields

The rating system we created on the Review object lets users quickly see each reviewer's opinion of the candidate's suitability for the position. While each individual opinion is important, it would be even better to see these ratings compiled in a way that summarizes how the candidate did overall. For example, wouldn't it be great if we could have a `Total Rating` field on each Job Application record that shows the sum of all the job application's review ratings?

The good news is that we can! A simple roll-up summary field on the Job Application object can summarize data from a set of related detail records and automatically display the output on a master record. Use roll-up summary fields to display the sum, minimum, or maximum value of a field in a related list, or the record count of all records listed in a related list.

Try It Out: Creating Roll-Up Summary Fields

Begin creating your roll-up summary just as you create any other custom field:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Roll-Up Summary` data type, and click **Next**.

When creating a field on an object that is not the master in a master-detail relationship, the `Roll-Up Summary` data type is not available. This is because roll-up summary fields are only available on the master object in a master-detail relationship.

5. In the `Field Label` field, enter `Total Rating`. Once you move your cursor, the `Field Name` text box should be automatically populated with Total_Rating.
6. Click **Next**.
7. In the `Summarized Object` drop-down list, choose Reviews.
8. Under Select Roll-Up Type, select `SUM`.
9. In the `Field to Aggregate` drop-down list, select `Rating`.
10. Leave `All records should be included in the calculation` selected, and click **Next**.

11. Accept all remaining field-level security and page layout defaults.
12. Click **Save**.

Now our job application records aggregate the ratings of their related reviews. This data could be a little deceptive, though, since some job applications might get reviewed more than others. It would be more helpful if we could see the average rating.

Roll-up summary fields themselves don't allow you to average values together, but you can use them in formulas that do. Let's create a second roll-up summary field on the Job Application object, and then build a simple formula field that uses both roll-up summary fields to find the average rating.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Roll-Up Summary` data type, and click **Next**.
5. In the `Field Label` field, enter `Number of Reviews`. Once you move your cursor, the `Field Name` text box should be automatically populated with Number_of_Reviews.
6. Click **Next**.
7. In the `Summarized Object` drop-down list, choose Reviews.
8. Under Select Roll-Up Type, select `COUNT`.

We don't need to specify a `Field to Aggregate` this time since we're just counting the number of related detail records and are not interested in any specific field.

9. Leave `All records should be included in the calculation` selected, and click **Next**.
10. Accept all remaining field-level security and page layout defaults.
11. Click **Save**.

Both roll-up summary fields are in place now. Let's build a formula field called `Average Rating` that divides the value of the first roll-up summary field by the value of the second.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Job Application**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select the `Formula` data type, and click **Next**.
5. In the `Field Label` field, enter `Average Rating`. Once you move your cursor, the `Field Name` text box should be automatically populated with Average_Rating.
6. Select the `Number` formula return type and click **Next**.
7. Click the **Insert Field** button.

8. Select `Job Application >`, then `Total Rating`, and click **Insert**.
9. Click the **Insert Operator** button and choose `Divide`.
10. Click the **Insert Field** button again.
11. Choose `Job Application >`, then `Number of Ratings`, and click **Insert**. Your formula should look like this:

```
Total_Rating__c / Number_of_Reviews__c
```

12. Click **Next**.
13. Accept the defaults in the remaining three steps of the wizard.
14. Click **Save**.

That wraps up all the fields and relationships we need to manage our reviews. Let's quickly organize the presentation of our fields and then test everything we've created.

Try It Out: Customizing the Review Object's Page and Search Layouts

First, let's update the page layout of the Review object so that the `Assessment` text field is in a single column section of the same name.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Review**.
3. In the Page Layouts related list, click **Edit** next to Review Layout.
4. Click **Create New Section** and define a new one-column section named `Assessment`.
5. Move the Assessment section just below the Information section, and put the `Assessment` and `Rating` fields in it.
6. Click **Save**.

Now, let's configure our Review search layouts so that reviews are always displayed with the associated job application, position, and candidate.

1. In the Search Layouts related list on the Review object detail page, click **Edit** next to Lookup Dialogs and add the following fields:
   - `Review Number`
   - `Rating`
   - `Job Application`
   - `Candidate`
   - `Position`
   - `Created Date`

2. Repeat for the Search Filter Fields layout.

To update the Reviews related list that appears on the Job Application detail page, we'll have to edit the related list directly on the Job Application page layout. This is different from how we added fields to the Job Application related list on the position and candidate detail pages because the Review object doesn't have an associated tab, and therefore, doesn't have a tab search layout. Remember—the tab search layout is responsible for both the fields that appear in the list on the tab home page and the default fields that appear in related lists on other object detail pages.

Note: The tab search layout is responsible for the fields in the related list layout only if the related list properties have not been modified on other objects' page layouts. For example, if you modify the properties of the Job Application related list on the Position page layout, those changes will always override the field specifications of the Job Application tab search layout.

Because the Review object doesn't have a tab search layout, we have to set those fields another way.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Job Application**.
3. In the Page Layouts related list, click **Edit** next to Job Application Layout.
4. Scroll down to the Related List Section, select `Reviews` and click **Edit Properties**.
5. Add the following fields to the Selected Fields box:
   - `Review Number`
   - `Rating`
   - `Candidate`
   - `Position`
   - `Created Date`
6. From the `Sort By` drop-down list, choose Review Number.
7. Click **OK**.
8. Click **Save** on the page layout edit page.

Look at What We've Done

Terrific! Let's go see what we've made:

1. Click the Job Applications tab and select a record, or create one if you haven't already.

Tip: When you use the Candidate and Position lookup dialogs as you're creating a job application record, note that, by default, they only display the most recently viewed records. You can locate additional records by using the search box, which returns records based on the `Candidate Number` or `Position Title` fields, respectively.

Use the * wildcard with other characters to improve your search results. For example, searching on C* returns every candidate record. Likewise, searching on *e returns all position records that include the letter 'e' in the title.

After the job application is created, notice that the Reviews related list now appears on the Job Application detail page. That's because we related the Review object to the Job Application object with a master-detail relationship.

2. In the Reviews related list, click **New Review** to create a review.

Do you see how the platform automatically filled in the job application number in the review's edit page? That's one of the small, but important, benefits of using the platform to build an application like this—not only is it easy to create links and relationships between objects, but the platform anticipates what we're doing and helps us accomplish our task with as few clicks as possible.

3. Complete the fields on the review, and click **Save**.

Notice that the name of the candidate and the title of the position appear on the review detail page. If you click the candidate's name, his or her record displays.

Go ahead and click around the rest of the app, creating a few more positions, job applications, candidates, and reviews. Pretty neat, huh? Our data is all interconnected, and our edits to the search layouts allow us to view details of several related objects all at once.

Creating a Many-to-Many Relationship

Our Recruiting app now has quite a few many-to-one relationships, but what if we needed to create a many-to-many relationship? For example, what if we have an object that stored information about various employment websites, and we wanted to track which open positions we posted to those sites? This would require a many-to-many relationship because:

- One position could be posted on many employment websites.
- One employment website could list many positions.

Here's where we get a little creative. Instead of creating a relationship field on the Position object that directly links to the Employment Website object, we can link them using a *junction*

object. A junction object is a custom object with two master-detail relationships, and is the key to making a many-to-many relationship.

For our app, we're going to create a junction object called Job Posting. A job posting fits into the space between positions and employment websites—one position can be posted many times, and one employment website can have many job postings, but a job posting always represents a posting about a single position on a single employment website. In essence, the Job Posting object has a many-to-one relationship with both the Position and the Employment Website objects, and through those many-to-one relationships, we'll have a many-to-many relationship between the Position and Employment Website objects.

Tip: In many apps, the sole purpose of a junction object is simply relate two objects, so it often makes sense to give the junction object a name that indicates the association or relationship it creates. For example, if you wanted to use a junction object to create a many-to-many relationship between bugs and cases, you could name the junction object `BugCaseAssociation`.

Let's look at a typical scenario at Universal Containers. There are open positions for a Project Manager and a Sr. Developer. The Project Manager position is only posted on Monster.com, but the Sr. Developer position is more difficult to fill, so it's posted on both Monster.com and Dice. Every time a position is posted, a job posting record tracks the post. As you can see in the following diagram, one position can be posted many times, and both positions can be posted to the same employment website.

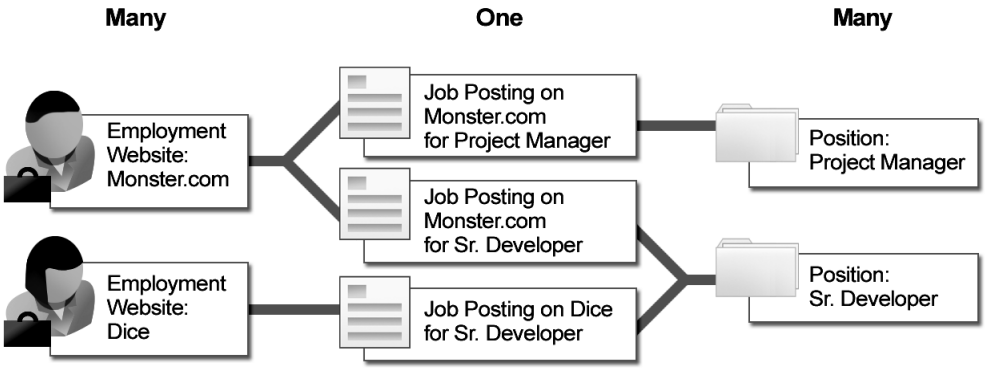

Figure 40: Using a Job Posting Object to Create a Many-to-Many Relationship Between Positions and Employment Websites

In relational database terms, each job posting record is a row in the Job Posting table consisting of a foreign key to a position record and a foreign key to an employment website record. The following entity relationship diagram shows this relationship.

Figure 41: Entity Relationship Diagram for the Position, Job Posting, and Employment Website Objects

Consequently, in order to define a many-to-many relationship between the Position and Employment Website objects, we'll need to create a Job Posting object with the following fields:

- A Position master-detail relationship
- An Employment Website master-detail relationship

Let's get started.

Try It Out: Creating the Employment Website Object

To create our Employment Website custom object, navigate back to **Setup ➤ Create ➤ Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 16: Values for Defining the Employment Website Object

Field	Value
Label	Employment Website
Plural Label	Employment Websites
Object Name	Employment_Website
Description	Information about a particular employment website
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Employment Website Name
Data Type	Text
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes

Field	Value
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	Yes

To create the Employment Website tab, select a `Tab Style` in the first step of the wizard, and then accept all the defaults until you get to the Add to Custom Apps page. On this page, select only the Recruiting App, and then click **Save**.

Let's wrap up the Employment Website object by adding a few custom fields.

Try It Out: Adding the URL Field to the Employment Website Object

Obviously, the Employment Website object needs to store the Web address of the employment website. We'll use the URL data type for this field. That way, when users click the field, the URL will open in a separate browser window. In addition to the URL, since most employment websites charge per posting, we'll want to keep track of how much it costs to post there, as well as our maximum budget for posting on the site.

Click **Setup ➤ Create ➤ Objects**, and then click **Employment Website** to view its detail page. In the Custom Fields & Relationships related list, use the **New** button to create custom fields according to the following table. Where necessary, we've indicated some additional values you'll need to fill in. Otherwise you can simply accept all defaults.

Table 17: Add Custom Fields to the Job Application Object

Data Type	Field Label	Other Values
URL	Web Address	Required
Currency	Price Per Post	Length: 5 Decimal Places: 2 Required
Currency	Maximum Budget	Length: 6 Decimal Places: 2

Data Type	Field Label	Other Values
		Required

Try It Out: Creating the Job Posting Object

Now it's time to create our Job Posting junction object! Navigate back to **Setup ➤ Create ➤ Objects**, click **New Custom Object**, and fill out the page according to the following table.

Table 18: Values for Defining the Job Application Object

Field	Value
Label	Job Posting
Plural Label	Job Posting
Object Name	Job_Posting
Description	Represents the junction object between a position and an employment website
Context-Sensitive Help Setting	Open the standard Salesforce Help & Training window
Record Name	Job Posting Number
Data Type	Auto Number
Display Format	JP-{00000}
Starting Number	00001
Allow Reports	Yes
Allow Activities	Yes
Track Field History	Yes
Deployment Status	Deployed
Add Notes & Attachments related list to default page layout	Yes
Launch New Custom Tab Wizard after saving this custom object	No

That was simple enough, but we're not quite done. We need to create the master-detail relationship fields that relate the Job Posting object with the Position and Employment Website objects.

Try It Out: Adding Fields to the Job Posting Object

To turn the Job Posting object into the junction object that relates the Position and Employment Website objects, we'll need to add two master-detail relationship fields. The first master-detail relationship will be the *primary relationship*. The detail and edit pages of our junction object (Job Posting) will use the color and any associated icon of the primary master object (Position). In addition, the junction object records will inherit the value of the Owner field and sharing settings from their associated primary master record.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Job Posting**.
3. In the Custom Fields & Relationships related list, click **New**.
4. Select `Master-Detail Relationship`, and click **Next**.
5. In the `Related To` drop-down list, choose Position, and click **Next**.
6. In the `Field Label` text box, enter `Position`. When you move your cursor, the `Field Name` text box should be automatically populated with Position as well.
7. Accept the remaining defaults, and click **Next** until you reach the final step of the wizard.

Here, we are given the chance to add the Job Postings related list to the Position object page layout. Instead of displaying information about related job postings, we want this list to show all the employment websites where this position is posted. So let's add the Job Posting related list, but rename it `Employment Websites`.

8. In the `Related List Label` text box, enter `Employment Websites`.
9. Accept the other defaults and click **Save & New**.

We're halfway through the creation of our many-to-many relationship. The next step is to create a second master-detail relationship on the Job Posting object to link it with the Employment Website object.

The second master-detail relationship creates a *secondary relationship*. Unlike the primary relationship, the secondary relationship has no affect on the look and feel of the junction object. However, just as in the primary relationship, the sharing settings of the master record in the secondary relationship also affect who can access the junction record, and deleting a record of the secondary master object will automatically delete its associated junction object records. So

in our app, if you delete an employment website record, all of its associated job posting records are deleted as well, even if the position is open.

10. Click **Setup ➤ Create ➤ Objects**.
11. Click **Job Posting**.
12. In the Custom Fields & Relationships related list, click **New**.
13. Select `Master-Detail Relationship`, and click **Next**.
14. In the `Related To` drop-down list, choose Employment Website, and click **Next**.
15. In the `Field Label` text box, enter `Employment Website`. When you move your cursor, the `Field Name` text box should be automatically populated with Employment_Website as well.
16. Click **Next**. Because we are creating a master-detail relationship, these settings cannot be changed.
17. Click **Next**. These settings cannot be changed as well.
18. Click **Next** to view the final step of the wizard.

This time we are given the chance to add the Job Postings related list to Employment Website object page layout. We'll eventually configure this related list to show all the positions that are posted on this website, so let's add the Job Postings related list but rename it `Positions`.

19. In the `Related List Label` text box, enter `Positions`.
20. Accept the other defaults and click **Save**.

Now our many-to-many relationship is complete! Or is it?

While we have an Employment Websites related list on the Position object and a Positions related list on the Employment Websites object, both related lists still display job posting records. This won't do.

In order to achieve our goal of listing multiple positions on an employment website record and multiple employment websites on a position record, we need to customize the fields in these related lists.

Customizing Related Lists in a Many-to-Many Relationship

The capability to customize related lists in a many-to-many relationship is more robust than the capability to customize related lists in a lookup relationship. When you have a lookup relationship between two objects (like the one we created between the Job Application and Candidate objects), the related list on one object can only display fields from the object to which it is directly related; it cannot span to other objects the way formulas can. For example, the Job Applications related list on a candidate record can display any job application field,

but it can't display any fields from the Position object, even though the Job Application object has lookup relationships with both the Candidate and Position objects.

Fortunately for us, many-to-many relationships allow for greater flexibility. When working with a many-to-many relationship, the junction object's related list on one master object can display the other master object's fields. We're going to take advantage of this by configuring the Positions related list on each employment website record to display fields from the Position object and vice versa, thus allowing these two objects to span to each other. It's all coming together now!

Try It Out: Customizing the Positions and Employment Websites Related Lists

Let's start by modifying the Employment Websites related list on the Position object.

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Position**.
3. In the Page Layouts related list, click **Edit** next to the Position Layout.
4. Scroll down to the Related List Section and double-click on Employment Websites.

In the popup window that appears, you'll notice the `Available Fields` column lists fields from both the Job Posting object and the Employment Website object. If there wasn't a master-detail relationship between job postings and employment websites, the `Available Fields` column list would only list job posting fields.

5. Move the `Employment Website: Employment Website` and `Employment Website: Web Address` fields to the `Selected Fields` column, and use the up and down arrows to arrange the fields in the following order:
   - `Employment Website: Employment Website`
   - `Employment Website: Web Address`
   - `Job Posting: Job Number`
6. Click **OK**.
7. A message appears reminding you to save your page layout. Click **OK**.
8. Click **Save** on the page layout.

Now do the same for the Positions related list on the Employment Website object as follows:

1. Click **Setup ➤ Create ➤ Objects**.
2. Click **Employment Website**.
3. In the Page Layouts related list, click **Edit** next to the Employment Website Layout.
4. Scroll down to the Related List Section and double-click Positions.

5. Move the following fields to the `Selected Fields` column, and use the up and down arrows to arrange them in the following order:
   - `Position: Title`
   - `Job Posting: Job Number`
   - `Position: Functional Area`
   - `Position: Location`
   - `Position: Open Date`
6. Click **OK**.
7. A message appears reminding you to save your page layout. Click **OK**.
8. Click **Save** on the page layout.

Look at What We've Done

Our many-to-many relationship is complete! Let's see it in action.

1. Create a few sample position and employment website records.
2. Scroll down to the Employment Websites related list at the bottom of any position record, and click **New Job Posting**. The Job Posting edit page appears.
3. Use the lookup icon to select the employment website where you want to post the position, and click **Save**.

The Employment Websites related list on that position now shows the name and Web address of the website to which you just posted, as well as the job posting number. Click the name of the employment website in the related list and scroll down to view the Positions related list, which shows all the positions posted to that website.

Now you know how easy it is to make related information just a click away!

Putting it All Together

We just created several objects and a lot of relationships. The following simple diagram shows us what we've accomplished so far.

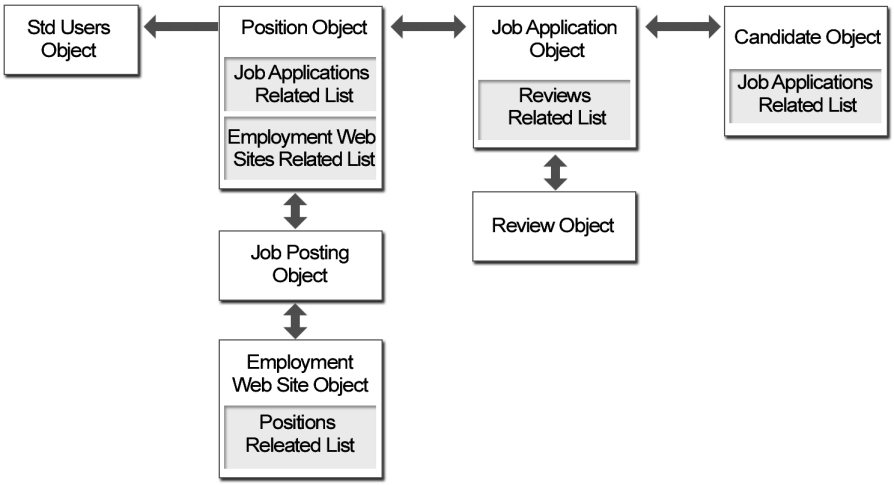

Figure 42: Recruiting App Relationships

All of these relationships, objects, and fields are shown below in an entity relationship diagram. An entity relationship diagram (ERD) is a conceptual representation of structured data, and is especially useful for planning and understanding an app.

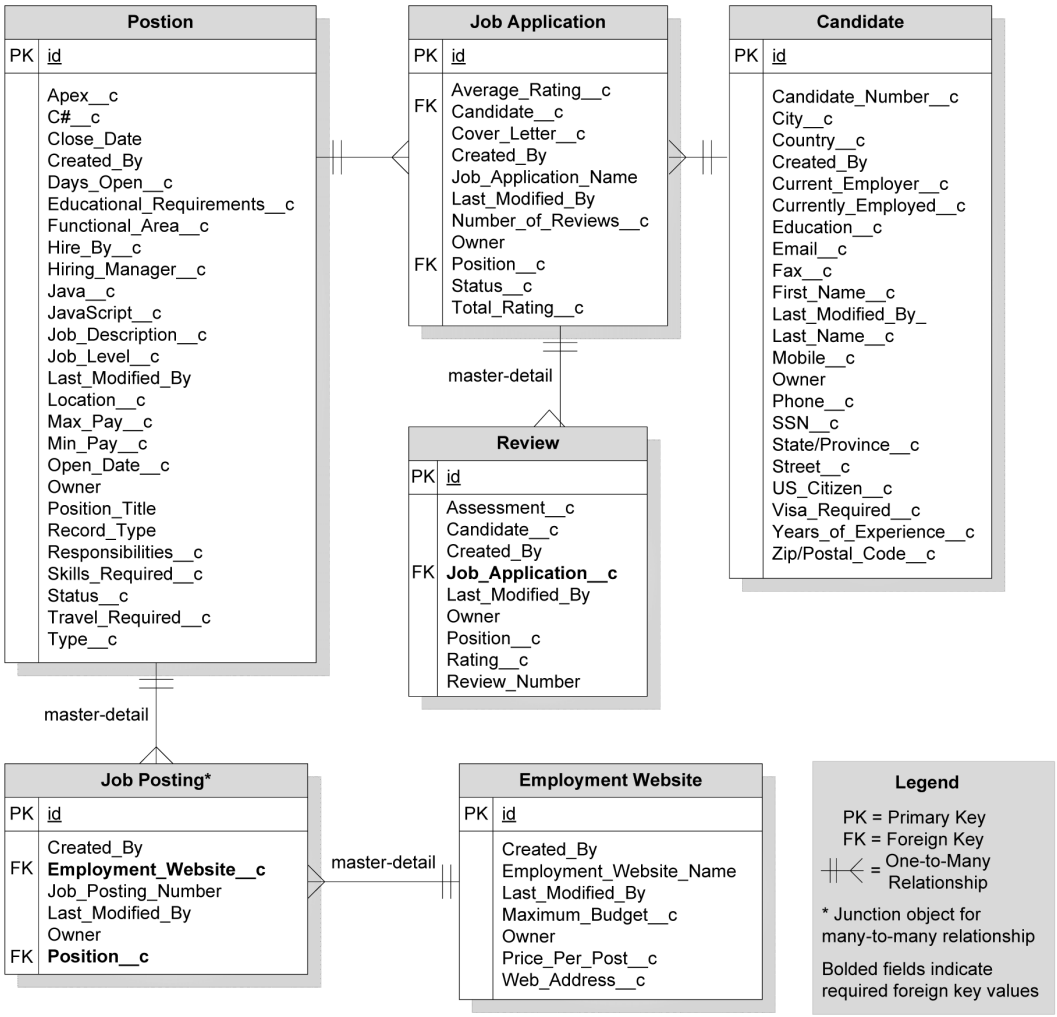

Figure 43: Recruiting App Entity Relationship Diagram

We've now built all of our Recruiting app objects and tabs, and we've defined lots of custom fields—everything from text fields and picklists to more complex formula fields and lookup relationship fields. We've created a robust user interface so that our recruiters and hiring managers can enter and retrieve data about positions and related candidates, job applications, and reviews, and we did all of this without writing a single line of code!

Remember when we assigned Clark Kentman as the hiring manager for the Benefits Specialist position? Let's look at what Clark can do now: He can create and update his positions, and track which websites he's posted them on. He can look at details about any candidates who have applied for the Benefits Specialist job, and he can review their related job applications.

He can also check the status of the job applications. He no longer has to go to Human Resources to search through Microsoft Word documents and spreadsheets to manage his tasks in the hiring process. The Recruiting app is well on its way to becoming a fully-functional and useful application!

However, before we leave this chapter behind, let's get ourselves prepared for the rest of this book by creating and importing some real data. It'll help us when we get to our next chapter on security and sharing if we have some records that we can work with.

Try It Out: Downloading Sample Data

In addition to entering data via our tabbed pages, we can also use the handy Import Wizard to import multiple records at a time. The ability to easily import data into your custom objects is one of the Force Platform's key benefits. Let's download some sample data so we can add more records to our custom objects without tons of typing.

1. Download the `RecruitingApp-3_0.zip` file containing the sample CSV (comma-separated values) import files from http://wiki.apexdevnet.com/index.php/Force_Platform_Fundamentals.
2. Extract the zip file to `C:\dev\recruiting` (or any directory on your computer).
3. Go to `C:\dev\recruiting`. This directory contains three CSV files: `Positions.csv`, `Candidates.csv`, and `JobApplications.csv`. (The directory also contains one other file that you'll use later when we build composite components for our app in *Chapter 10: Moving Beyond Native Apps* on page 265.)

Before we import anything, we need to make a modification to the import file for positions. The sample `Positions.csv` you downloaded contains fictional users in the Hiring Manager column. The names of these users most likely won't match any user in your organization, and if you import the file "as is," the Import Wizard won't be able to find any matching users, and the `Hiring Manager` field on each position record will be left blank. So let's go ahead and make that change.

4. Go to `C:\dev\recruiting`, and open `Positions.csv` in Excel, a text editor, or any other program that can read CSV files.
5. In the Hiring Manager column, replace the fictional users with the first and last name of a user in your organization.
6. Save the file, making sure to maintain the CSV format.

Note: If your locale isn't English (United States), the date and field values in `Positions.csv` are also invalid. You'll need to change them before you import.

Try It Out: Using the Import Wizard

Now, let's walk through the process of importing position records using the Import Wizard and the `Positions.csv` file you downloaded.

1. Click **Setup ➤ Data Management ➤ Import Custom Objects**.
2. Click **Start the Import Wizard!**. The Import Wizard appears.
3. Select `Position` for the type of record you're importing, and click **Next**.
4. Choose `Yes` to prevent duplicate position records from being created as a result of this import. Accept the other defaults for matching, and click **Next**.
5. Select `None` for the record owner field. We didn't include a User field in the CSV file to designate record owners. The Import Wizard assigns you as the owner of all new records.
6. Choose the `Hiring Manager` lookup relationship field so you can link position records with existing User records in the Recruiting app, and click **Next**.
7. Select `Name` as the field you want to match against as the Import Wizard compares `Hiring Manager` names in your import file with User names in the system, and click **Next**.
8. Click **Browse**, and find `C:\dev\recruiting\Positions.csv`. Click **Next**.
9. Use the drop-down lists to specify the Salesforce fields that correspond to the columns in your import file. For your convenience, identically matching labels are automatically selected. Click **Next**.
10. Click **Import Now!**

Use the following table to repeat the import process for candidate records. You'll notice the wizard skips the two steps about lookup relationship field matching—because the Candidate object doesn't have any lookup relationship fields, the Import Wizard automatically leaves those steps out.

Table 19: Importing the `Candidates.csv` File

For this wizard step...	Select these options...
1. Choose Record	Candidate
2. Prevent Duplicates	No—insert all records in my import file
3. Specify Relationships	None
4. File Upload	Browse to `C:\dev\recruiting\Candidates.csv`
5. Field Mapping	Accept all defaults
6. Verify Import Settings	Click **Import Now!**

Finally let's do it one more time for job application records. In this iteration, we're going to make use of the `Email` field, an external ID on the Candidate object, to match up job applications with the correct candidate records.

Table 20: Importing the `Job_Applications.csv` File

For this wizard step...	Select these options...
1. Choose Record	Job Application
2. Prevent Duplicates	No—insert all records in my import file
3. Specify Relationships	`Which user field...?` None `Which lookup fields...?` Candidate, Position
4. Define Lookup Matching	`Which field on Candidate...?` Email (External ID) `Which field on Position...?` Position Title
5. File Upload	Browse to `C:\dev\recruiting\Job_Applications.csv`
6. Field Mapping	`Email (col 0)`: Candidate `Position Title (col 1)`: Position
7. Verify Import Settings	Click **Import Now!**

Great! While the files are importing, you can go to **Setup ➤ Monitoring ➤ Imports** to check on their status.

Once the import operations have completed, return to the Positions, Candidates, or Job Applications tab and click **Go!** next to the `View` drop-down list. You'll see a list of all the new records you just imported.

We've just added a bunch of data to our app without a lot of work. In the next chapter, we'll take a look at all the ways we can control access to this data using the built-in tools of the platform. We'll get into the nitty-gritty about security, sharing rules, permissions, roles, and profiles.