Force.com Workbook

Salesforce Platform Workshop, Spring '16

@salesforcedocs

Last updated: April 19, 2016

CONTENTS

ABOUT THE WORKBOOK

This workbook shows you how to create a cloud app in a series of tutorials. While you can use the Salesforce platform to build virtually any kind of app, most apps share certain characteristics, such as:

- A database to model the information in the app
- A user interface to expose data and functionality to those logged into your app
- Business logic and workflow to carry out particular tasks under certain conditions

In addition, apps developed on the Salesforce Platform automatically support:

- A public website and mobile apps to allow access to data and functionality
- A native social environment that allows you to interact with people or data
- Built-in security for protecting data and defining access across your organization
- Multiple APIs to integrate with external systems
- The ability to install or create packaged apps

The workbook tutorials are centered around building a very simple warehouse management system. Your warehouse contains computer hardware and peripherals: laptops, desktops, tablets, monitors, that kind of thing. To keep track of how merchandise moves out of the warehouse, you use an invoice. An invoice is a list of line items. Each line item has a particular piece of merchandise, and the number of items ordered. The invoice rolls up all the prices and quantities for an invoice total. It's a very simple data model, but just enough to illustrate the basic concepts.

Development proceeds from the bottom up; that is, you first build an app and database model for keeping track of merchandise. You continue by modifying the user interface, adding business logic, etc. Each of the tutorials builds on the previous tutorial to advance the app's development and simultaneously help you learn about the platform.

Audience

These tutorials are intended for developers new to the Salesforce platform and for Salesforce admins who want to delve more deeply into app development.

Version

You should be able to successfully complete all procedures using the Summer '14 version of Salesforce.

Supported Browsers

Microsoft Edge

Salesforce supports Microsoft Edge on Windows 10 for Salesforce Classic. Note these restrictions.

- The HTML solution editor in Microsoft Edge isn't supported in Salesforce Knowledge.
- Microsoft Edge isn't supported for the Developer Console.
- Microsoft Edge isn't supported for Salesforce CRM Call Center built with CTI Toolkit version 4.0 or higher.

Can I Use My Tablet or Phone?

Most of the tutorials can be completed using tablet or phone, although screen size may be an issue with some tutorials, and a keyboard is convenient for code. In addition, note the following.

- Tutorials that require moving data from a local file system to the cloud may not be possible depending on the capabilities of the device. For example, if you try to upload a CSV file, your device might only allow you to browse for photos.
- Some tutorials require you to switch between different users, which is much easier if you have two different browsers open at the same time. If your device is only capable of using one browser, you have to log in and out each time you switch users.

Sign Up for Developer Edition

This workbook is designed to be used with a Developer Edition organization, or *DE org* for short. DE orgs are multipurpose environments with all of the features and permissions that allow you to develop, package, test, and install apps.

1. In your browser, go to `http://sforce.co/YrZZJ3`.
2. Fill in the fields about you and your company.
3. In the `Email Address` field, make sure to use a public address you can easily check from a Web browser.
4. Type a unique `Username`. Note that this field is also in the *form* of an email address, but it does not have to be the same as your email address, and in fact, it's usually better if they aren't the same. Your username is your login and your identity on `developer.salesforce.com`, so you're often better served by choosing a username such as `firstname@lastname.com`.
5. Read and then select the checkbox for the `Master Subscription Agreement` and then click **Submit Registration**.
6. In a moment you'll receive an email with a login link. Click the link and change your password.

Optional: Install the Warehouse App

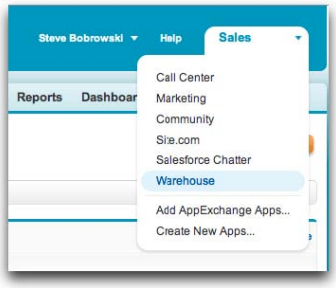
If you want to skip over the 100-level tutorials, you can install the Warehouse app as a package. A package is a bundle of components, usually an app, that you can install in your org.

The packaged app option is useful for advanced developers or admins who already know their way around custom objects, fields, relationships, basic UI, and app logic. However, if you're an experienced developer new to the platform, it's still a good idea to go through the 100-level tutorials, especially for the mobile content.

To install the Warehouse app:

1. Click the installation URL link: https://login.salesforce.com/packaging/installPackage.apexp?p0=04ti0000000Pj8s
2. If you aren't logged in already, enter the username and password of your DE org.

3. On the Package Installation Details page, click **Continue**.
4. Click **Next**, and on the Security Level page click **Next**.
5. Click **Install**.
6. Click **Deploy Now** and then **Deploy**.
7. Once the installation completes, you can select the Warehouse app from the app picker in the upper right corner.



8. To create data, click the Data tab.
9. Click **Create Data**.

Note: If you installed the package by mistake, or you want to delete it, from Setup, enter *Installed Packages* in the Quick Find box, select **Installed Packages**, and then delete the package.

CREATE AN APP AND DATABASE

Duration: 40–60 minutes

The Salesforce platform makes it easy to build custom apps and databases in the cloud. In this lesson, you learn how to build a basic app with just a few clicks and then enhance the underlying database as you go along. You also learn how to validate data entry and load data.

Create a Warehouse App

Level: Beginner; **Duration:** 5–10 minutes

Running apps in the cloud is great because there is no server to configure, no software to install, and no ongoing maintenance of your infrastructure. This tutorial teaches you how to build a cloud app.

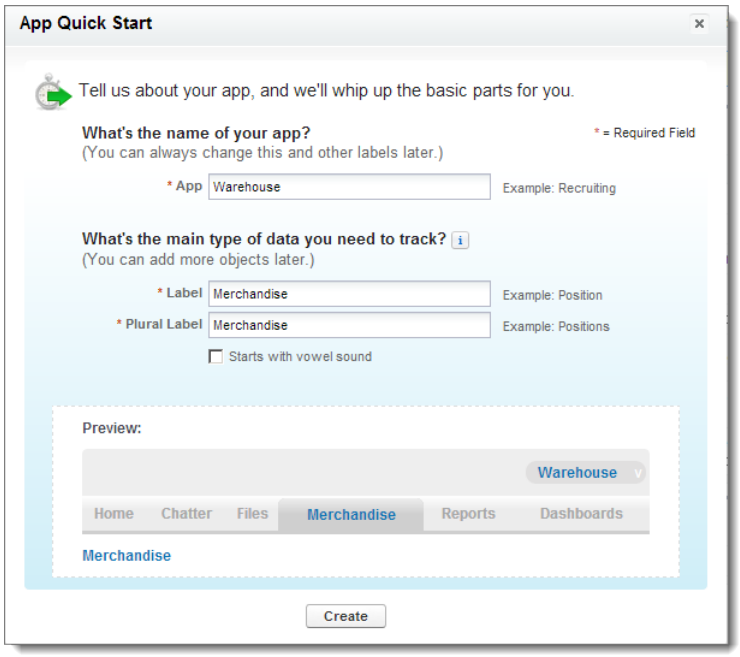
At the heart of this app is what you want to sell: merchandise. When you create an app, you automatically create a data object that keeps track of all the elements of a particular merchandise item, such as its name, description, and price. On the Salesforce platform, these data objects are called *custom objects*. If you're familiar with databases, you can think of them as a table.

An object comes with standard fields and screens that allow you to list, view, and edit information about the object. But you can also add your own fields to track or list just about anything you can think of. When you complete this tutorial, you'll have a working app with its own menu, a tab, and a custom object that tracks the name, description, and price of all your merchandise, as well as screens that allow you to view and edit all of this information.

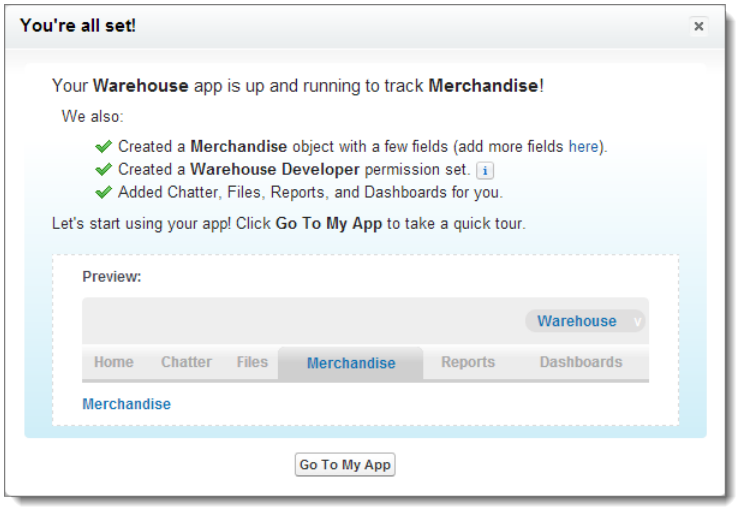
Step 1: Build a Cloud App and Database

You can create an app with just a few clicks. In this tutorial, you use the App Quick Start wizard to create an app that can help you manage merchandise records in a warehouse.

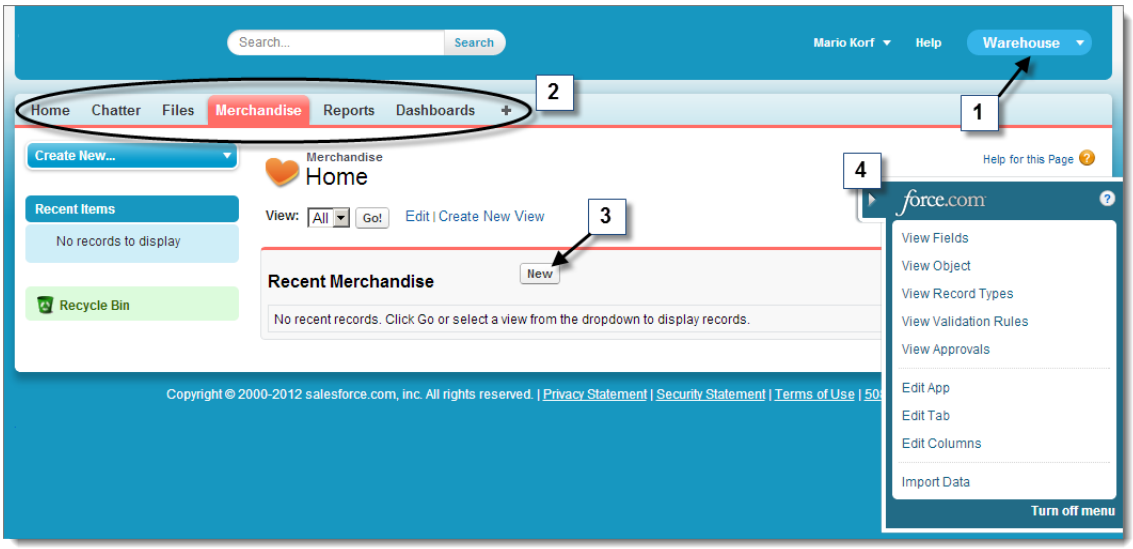
1. Launch your browser and go to `https://login.salesforce.com`.
2. Enter your username (in the form of an email address) and password.
3. From the Force.com Setup page, click **Add App** in the Getting Started section. (If you're starting from somewhere else, look in the upper right corner, and click **Setup**.)
4. Fill in the form as follows:
   - For the `App`, type *`Warehouse`*.
   - For the `Label`, type *`Merchandise`*.
   - For the `Plural Label`, type *`Merchandise`*.



5. Click **Create** and you see right away some of the functionality that's automatically added.



6. Click **Go To My App** to see your new app.
7. Click **Start Tour** and follow along for a quick overview of your app's built-in user interface.



1. **Force.com app menu**—Shows the apps that are available to you. The app you just created is selected.
2. **Tabs**—Provide an easy way to find and organize objects and records. In the Merchandise tab, which is open, you can create, view, and edit records. The other tabs are the standard feature tabs that are included with every app.
3. **Create records**—Click **New** to add records to your custom object. If you click this button now, you see only one data entry field in the object, but you'll create more later.
4. **Force.com Quick Access menu**—Quickly jump to relevant app customization features. The menu is available from any object list view page and record detail page, but only for users with the "Customize Application" permission.

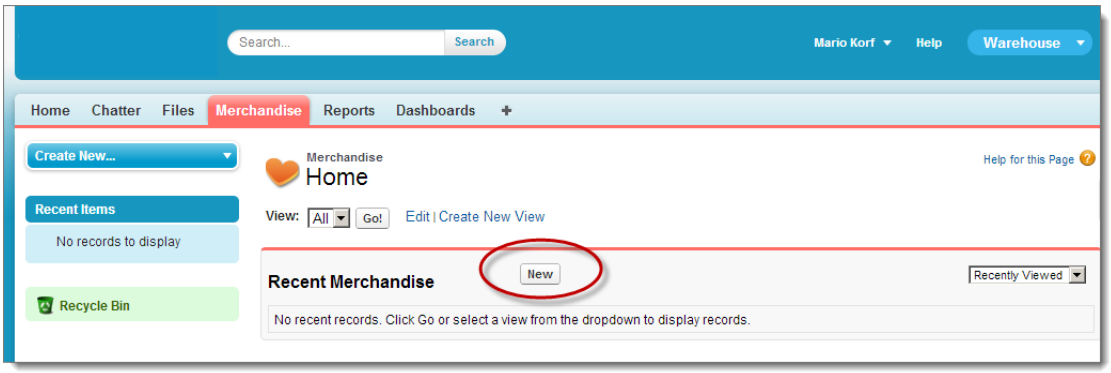
Tell Me More....

An app is composed of tabs, but the tabs don't have to be related to each other. In fact, you can modify custom apps to group all of your most frequently used tabs together in one place. For example, if you refer to the Accounts tab a lot, you can add that to the Warehouse app. You can switch between apps you created, bought, or installed by selecting them from the menu.

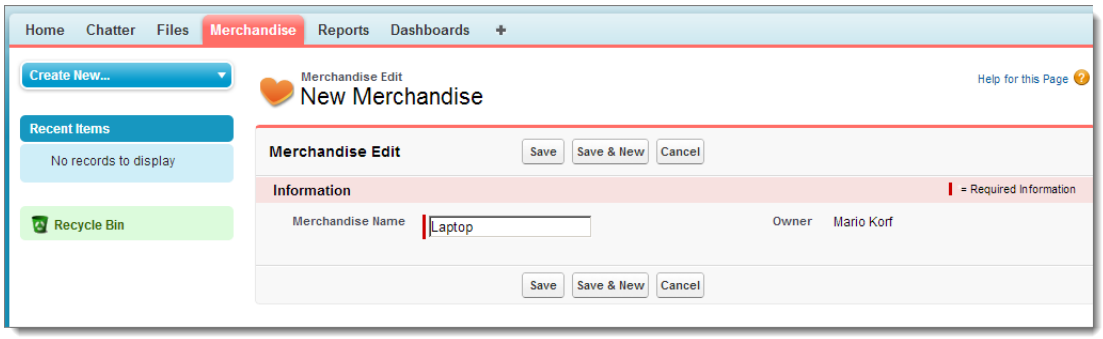
Step 2: Try Out the App

Your app doesn't do much yet, but you can start using it right away.

1. To try out your new app, click **New** to create a new Merchandise record.

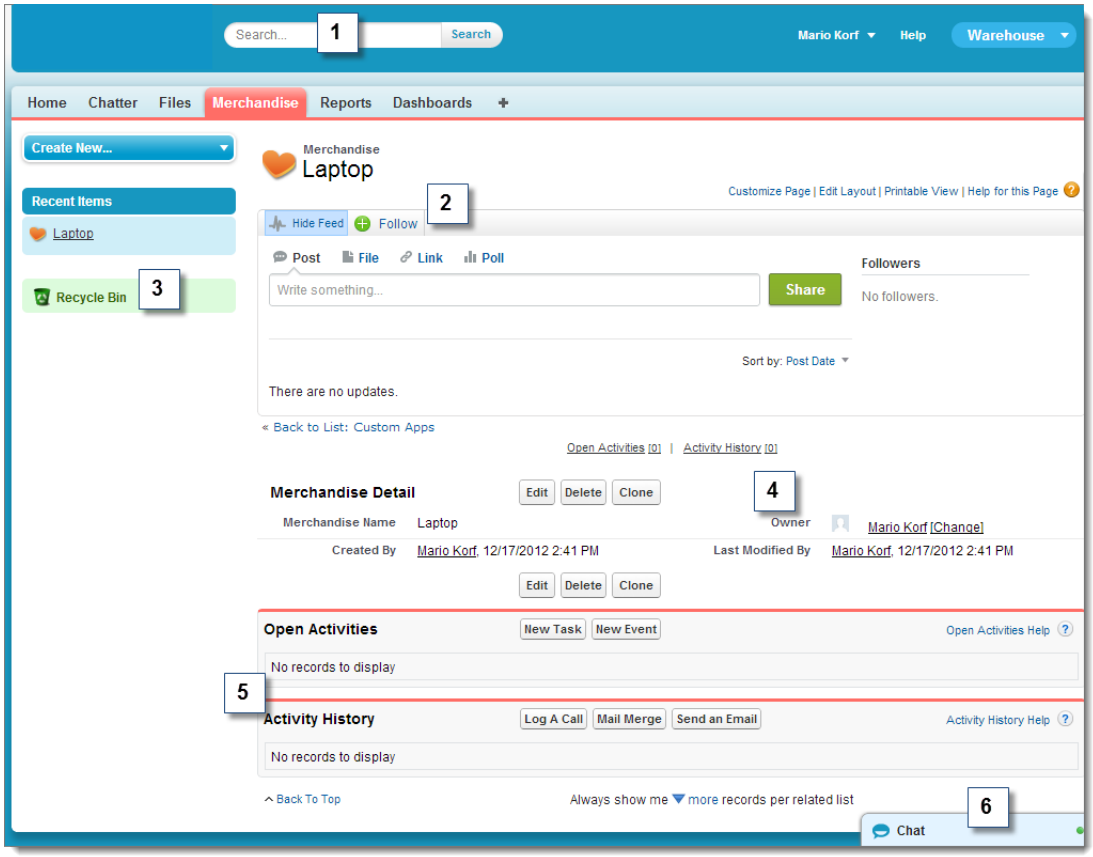


2. Add a new merchandise record for *Laptop* and click **Save**.



Step 3: Explore the App

Building a simple app is really fast! But don't let this basic app fool you. Salesforce is a powerful platform that lets you build much more sophisticated apps just as easily, and without code. Look closely around the screen to see all of the functionality included by default.



1. Every app has full-text search functionality for all text fields of an object and Chatter feeds.
2. Every object in Salesforce automatically has an attached "feed," called Chatter, that lets authorized app users socialize about and collaborate on the object. Using Chatter, users can post updates in an object's feed, comment on posts, and follow (subscribe to)

the feed to get pushed updates when they happen. For example, on a Merchandise record, one user might post a question about the record, to which followers and other users can comment in reply.

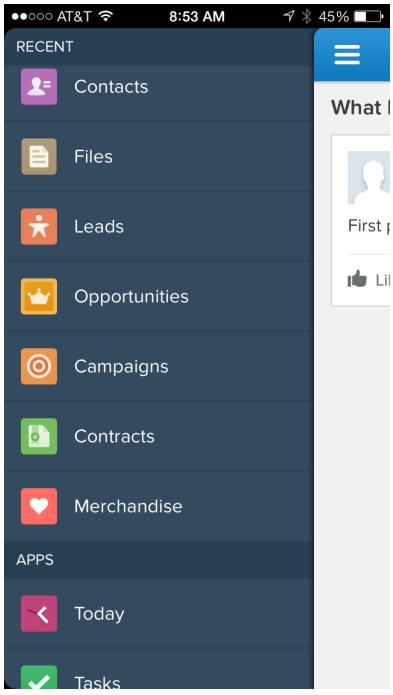
3. Every DE org has a recycle bin that you can use to view and restore deleted records.
4. Every record in Salesforce has an "owner," which serves as the basis for a powerful security system that supports ownership-based record sharing.
5. You can also manage activities related to a record from the Open Activities and Activity History related lists. Activities include tasks to perform (making phone calls or sending email), calendar events, and requested meetings.
6. Every DE org has a Chat window that lets users interact with one another.

Access the App from a Mobile Device

Level: Beginner; **Duration:** 5–10 minutes

The simple app you created is already accessible as a mobile app. What? Truly! Most things you create in Salesforce are available via a mobile device, giving your users full access to the information they need, no matter where they are. As you continue to develop in this workbook, everything you do in the full site is reflected in the Salesforce1 mobile app.

For the warehouse use case, you can imagine workers in a warehouse typically need to make a physical check of the inventory. Rather than lug around a laptop or transfer data by pen and paper, they can update on the go, right on the phone. This in turn might be useful to a service technician on the road, who can instantly see which products are and aren't available.



Step 1: Set Up Mobile Access

There are two ways to access Salesforce1: using a downloadable app or a mobile browser app.

1. First, you need to be able to access Salesforce1:

- To use the downloadable app, use your mobile device's browser to go to `www.salesforce.com/mobile`, select the appropriate platform, and download Salesforce1.
- To enable the mobile browser app, from Setup, enter "Salesforce1 Settings" in the `Quick Find` box, then select **Salesforce1 Settings**, and then **Enable the Salesforce mobile browser app**. Now, when you navigate to `login.salesforce.com` from your mobile browser, Salesforce will recognize that you're working from a mobile device and redirect you to the Salesforce1 mobile browser app.

2. Open Salesforce1 from your mobile device.
3. Enter your Salesforce credentials and tap **Log in to Salesforce**. You might be asked to verify your mobile device.

Tell Me More....

The downloadable mobile app is usually preferable because the following features aren't supported in the mobile browser app.

- **Today** helps users plan for and manage their day by integrating calendar events from their mobile device with their Salesforce tasks, contacts, and accounts.
- **Push notifications** alert users to important things when they aren't using the app.

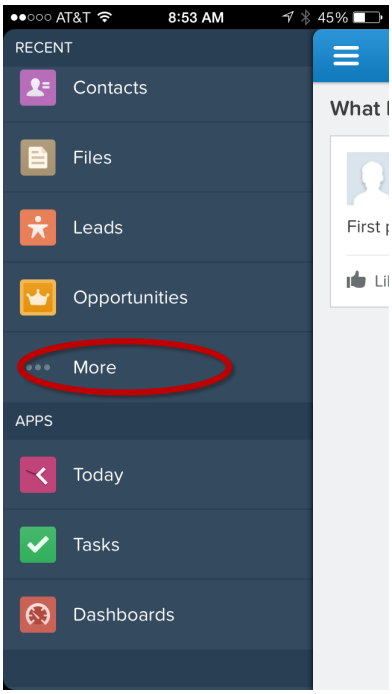
Step 2: Try Out the Mobile App

Start the mobile app and then look at how the Merchandise tab and its fields appear on a mobile device.

1. If you logged in using the downloadable app, you're prompted to allow access to your data. Tap **OK** and continue.
2. On the first screen, you're prompted to create your first post. Go ahead and tap the Post action in the action bar.

3. Enter some text like *`First post!`*, and then tap **Share**.
4. Tap in the left corner to open the navigation menu.

5. Scroll down and tap **More**.



6. Tap **Merchandise**.
7. You can easily create a new piece of merchandise from the mobile device. Tap **New**.
8. Name it `E-reader`, and then tap **Save**.

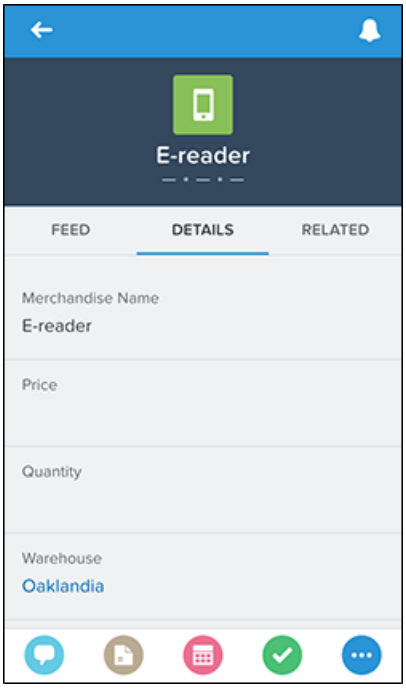
Tell Me More....

You probably noticed that unlike the full Salesforce site, there isn't a Home tab, and there doesn't appear to be a Warehouse app. Additionally, it took some effort to find the Merchandise tab. Why is that?

- Each tab is represented through a menu item in the Recent section of the Salesforce1 navigation menu. Since your app's Merchandise tab is new, it doesn't appear on the **Recent** section yet until you start using it. After you've used the app a bit, the default tabs (account, case, etc.) are replaced by the tabs you use most frequently.
- Salesforce apps, such as the Sales app or a your custom Warehouse app, don't appear in Salesforce1, because the mobile app figures out which *records* you look at most often. Rather than using the Force.com app menu to customize the tabs a user sees regularly, the smart search items under the **Recent** section reorder based on the user's history of recent objects.
- Don't get the idea that the layout and navigation are entirely dynamic. You can customize the fields, actions, apps, and the navigation of virtually the entire mobile app. You'll get into that in later tutorials.

Step 3: Explore the Mobile App

You get a lot of functionality out of the box with Salesforce1. Take a moment to explore what's there.



1. You should still be on the detail page for your new merchandise item. You can Edit, Clone, or Delete this record from its *detail page*.
2. Swipe left and you'll see there's a page for activities related to this item. This is the *related information page*.
3. Swipe right from the detail view and you'll see there's a blank page for the feed. There will be feed items here just as soon as you make some changes.
4. Tap from the action bar at the bottom of the page, and then notice the list of icons that represent actions. This area is called the *action menu*.
5. Try out an action by tapping **Post**.
6. Enter some text, such as `Adding an e-reader to inventory` and then tap **Submit**. You can automatically see the post you created in the feed for the e-reader. Anyone who follows that item will get updates for it.
7. Tap , and this time tap **New Task**.
8. In Subject, type `Enter a price`, and for Due Date, tap the calendar and choose **Today**.
9. Tap **Save**.
10. From the detail page, swipe left and tap **Open Activities**, and you'll see the task you created for yourself.

As you can see, you get the same functionality from the mobile app as you do in the full site—just the controls and navigation are different.

Tell Me More....

- In the related information page you saw activities listed, and you might be wondering if you can add other related things. Yes! You can add notes, attachments, Visualforce pages, and *mobile cards* to this page, which you'll get to later.
- You saw a number of things you can do from the action bar, such as create a post, log a call, create a case, and so on. Of course you can add and remove items from the tray and rearrange the order. This is all done in the page layout editor and is covered in later lessons.

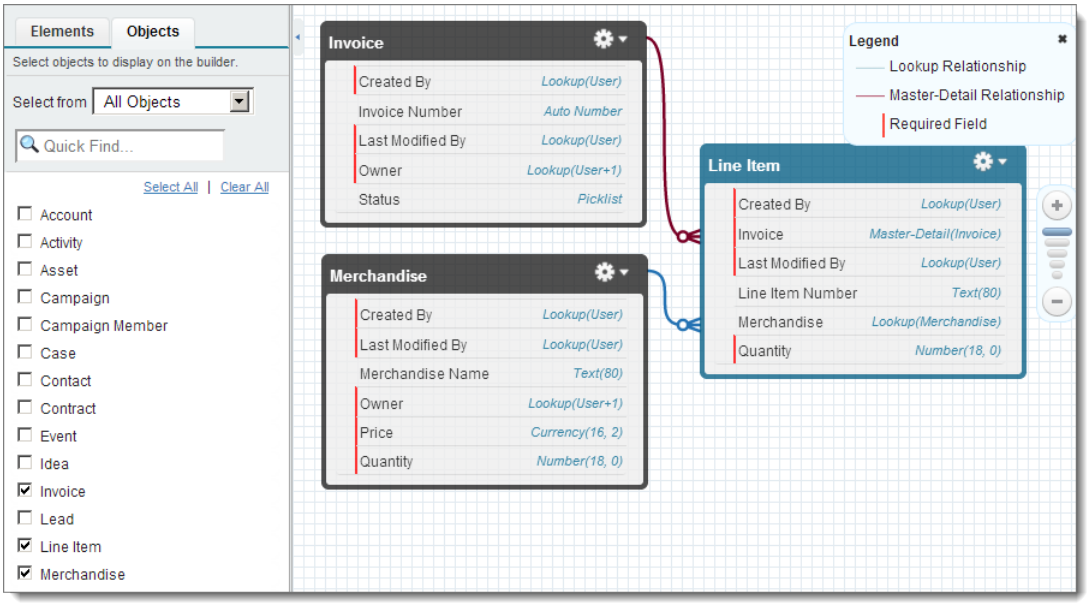
Add Fields to an Object

Level: Beginner; **Duration:** 5–10 minutes

In the first tutorial, you created a cloud app for managing merchandise in a warehouse. Behind the scenes, the platform created a database for the app. This tutorial is the first of many that teach you how to continue building the database for your app. A *database* organizes and manages data so that users can work with it efficiently. Traditional relational databases use tables to manage discrete, possibly related, collections of information, organized further into datatype-specific columns (attributes) and rows (records). In Salesforce, you refer to these as *objects*.

Your DE org comes with many *standard objects* (for example, Accounts, Products, Tasks) that support pre-built apps. Any new objects you create are called *custom objects*. The Merchandise object is one such custom object. In this tutorial you add two new *custom fields* (`Price` and `Inventory`) to supplement the *standard fields* the object already has (`Name, Owner, CreatedBy, LastModifiedBy`).

The following image is a sneak peek of the data model you will be building, which allows you to view your objects, fields, and relationships.



Step 1: Add the Price Field to the Merchandise Object

A merchandise object should have fields that are used for tracking various information, such as how much individual units cost and how many units are in stock. You can add custom fields to list or track just about anything you can think of.

1. From Setup, enter *`Objects`* in the `Quick Find` box, then select **Objects**.
2. Click **Merchandise**, scroll down to Custom Fields and Relationships, and click **New**.
3. The New Custom Field Wizard helps you quickly specify everything about a new field, including its name, labels to use for app pages, help information, and visibility and security settings. Create the Price field as follows:
   a. For Data Type, select `Currency`, and click **Next**.
   b. Fill in the custom field details:
      - `Field Label`: *`Price`*
      - `Length`: *`16`*

- `Decimal Places`: *2*
- Select the `Required` checkbox

c. Leave the defaults for the remaining fields, and click **Next**.

d. Click **Next** again to accept the default field visibility and security settings.

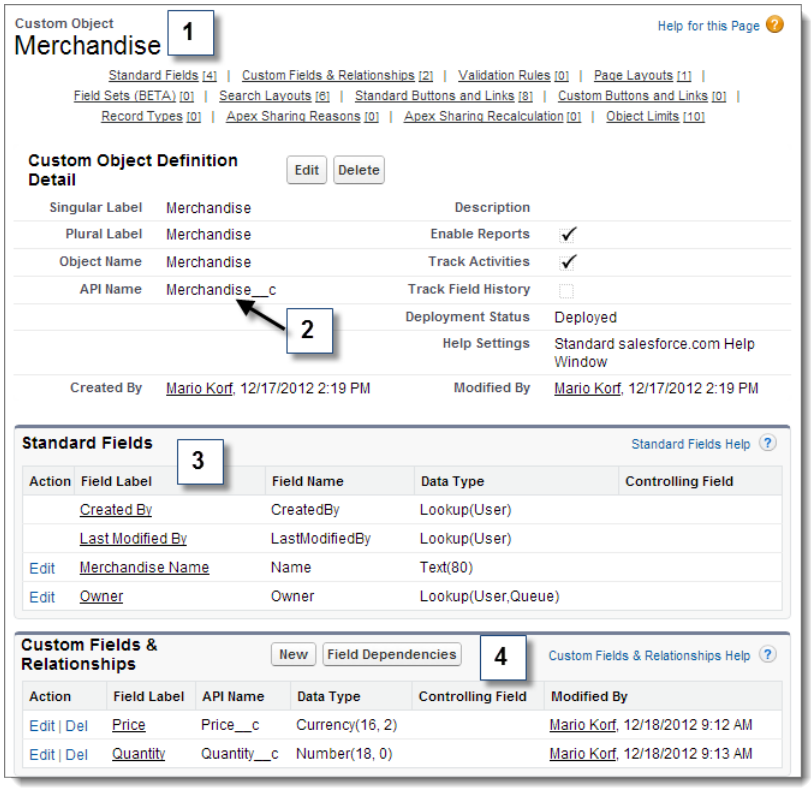
e. Click **Save & New** to save the `Price` field and to return to the first step of the wizard.

Step 2: Add the Quantity Field to the Merchandise Object

You should already be in the New Custom Field wizard, so you can create the Quantity field in the same manner.

1. For Data Type, select `Number` and click **Next**.
2. Fill in the field details:
   - `Field Label`: *`Quantity`*
   - Select `Required`
3. Leave the defaults for the remaining fields, and click **Next** and **Next** again.
4. Click **Save**.

Take a look at this image to familiarize yourself with the Merchandise custom object.



1. **Merchandise detail page**—Shows you everything you need to know about your Merchandise custom object. Soon you'll add relationships, validation rules, and other neat features to this object.

2. **API name**—When you created the Merchandise object, you didn't specify an API name, but one was generated for you. This name is how the object is referenced programmatically. All custom objects end in `__c`, which differentiates them from standard objects.
3. **Standard fields**—Some fields are generated automatically; these are standard fields. For example, the Merchandise object has a standard field for Owner, which means it automatically tracks who created each record.
4. **Custom fields**—Includes the fields you just created in this step. Like custom objects, custom fields have API names that end in `__c`.

Tell Me More....

- The custom fields you created so far are nothing fancy, but you can do fancy! The platform has support for nearly any type of data you want to track, such as currency, email, geolocation, URLs, date/time, and so on. Fields don't just contain static values, they can be derived from formulas, or take their values from other objects.
- Why do you need an API name as well as the object and field label? A label is what the user sees, so it should be easy to read and may contain spaces. The API name is used internally in code, and can't contain spaces or illegal characters. For example, a field labeled "Customer ph# :" would be named `Customer_ph` in the code (the system replaces spaces with underscores and removes the # and : characters).

Step 3: Try Out the App

At this point you have a nice representation of your warehouse items—each has a name, price, and quantity. Time to create some more inventory.

In the first tutorial, you created one Merchandise record with just a name (Laptop). In this tutorial you create a few more Merchandise records that include the new `Price` and `Quantity` fields.

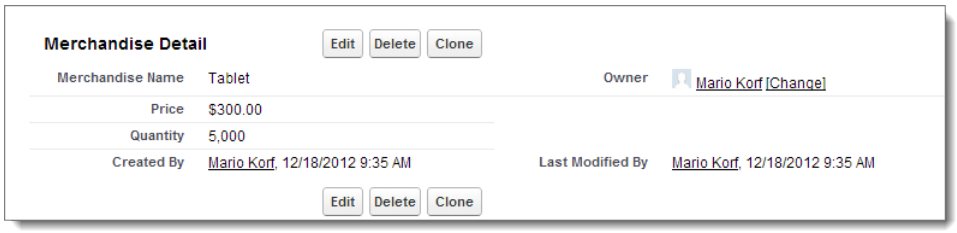
1. Click the Merchandise tab to leave Setup and return to the app.
2. Click **Laptop** in the Recent Merchandise listing.
3. Click **Edit**, and then specify the price and quantity as follows.
   - `Price`: *`500`*
   - `Quantity`: *`1000`*

   Before you move on, take note how the platform automatically added the new `Price` and `Quantity` fields to your Laptop record. When you add new fields through the wizards, the fields are added to your existing objects *and* exposed automatically on your app's user interface. Nice!
4. Click **Save**.

5. If you created the E-reader item in the mobile tutorial, edit that item in a similar manner. If not, create a new Merchandise record called E-reader with the following field values.
   - `Price`: *100*
   - `Quantity`: *1500*
6. Click **Save & New** and create a merchandise record for Desktop with these attributes.
   - `Price`: *1000*
   - `Quantity`: *500*
7. Click **Save & New**, and create a merchandise record for Tablet with these attributes.
   - `Price`: *300*
   - `Quantity`: *5000*
8. Click **Save**.

Tell Me More....

Take a close look at one of your merchandise records. Notice the `Owner`, `CreatedBy`, and `LastModifiedBy` fields. These are *standard fields* which the platform automatically manages. Users have the ability to edit the `Name` standard field, along with the custom fields `Price` and `Quantity`.



Also take a look at the Recent Items in the sidebar. This handy feature lets you view and navigate to the most recently changed records in your database. The linked names in this sidebar come from each object's `Name` field.

Create a New Object

Level: Beginner; **Duration:** 10–15 minutes

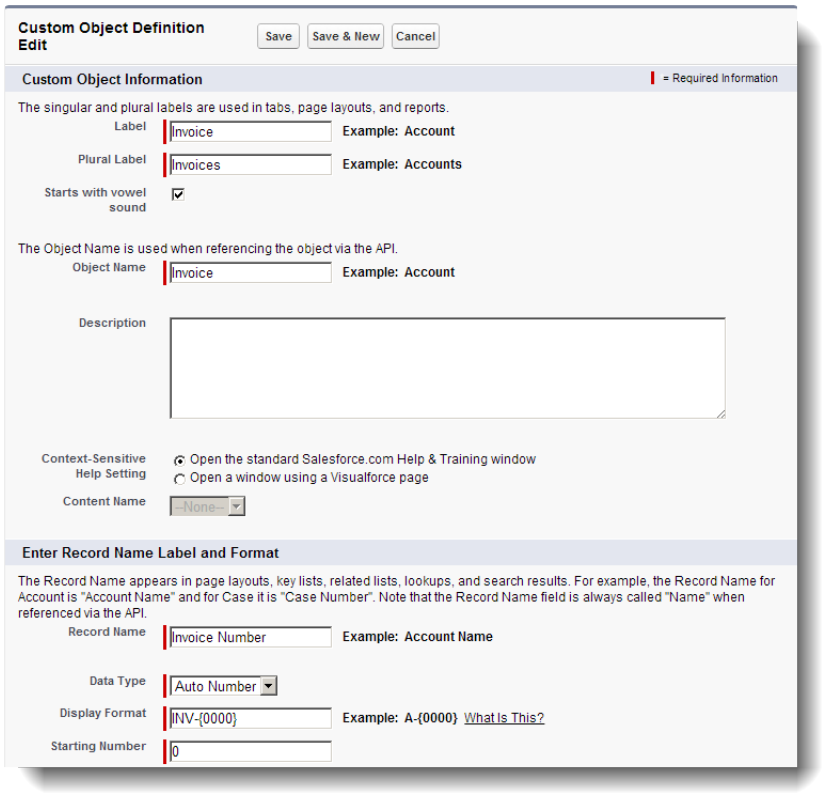
To make the Warehouse app more realistic, you need invoices to track orders going in and out of the warehouse. In this tutorial, you learn how to extend the app further by:

- Creating another custom object, for keeping track of invoices. This object needs a `Status` field to track whether the invoice is open, closed, or somewhere in-between.
- Adding a tab to the app so users can work with invoices.
- Reordering tabs for easier navigation.

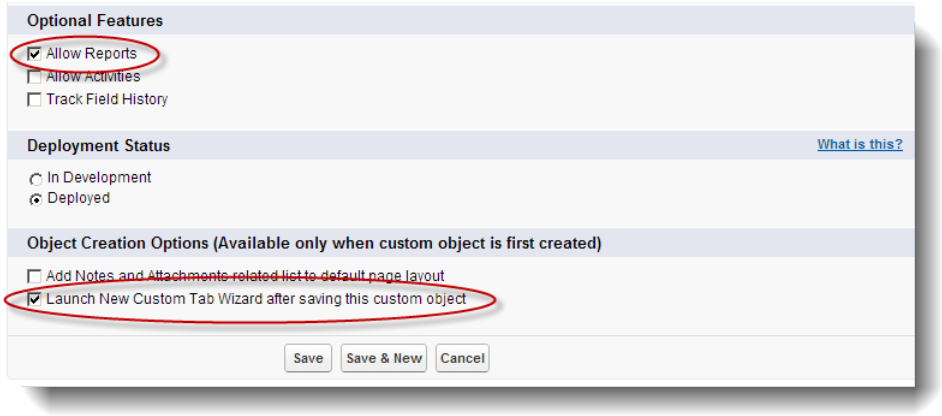
Step 1: Create the Invoice Object Using the Wizard

An invoice is required to move inventory into or out of the warehouse. In this step, you create an invoice object that allows you to create multiple invoice statements, each with a unique number, status, and description.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**.
2. Click **New Custom Object**.
3. Fill in the custom object definition.
   - In the Label field, type *Invoice*.
   - In the Plural Label field, type *Invoices*.
   - Select Starts with vowel sound.
   - In the Record Name field, type *Invoice Number* (replace *Name* with *Number*).
   - For Data Type, select *Auto Number*.
   - In the Display Format field, type *INV-{0000}*. (Note there are no spaces.)
   - In the Starting Number field, type *0*.



4. In the Optional Features section, select Allow Reports (in case you create reports later).
5. Select Launch New Custom Tab Wizard after saving this custom object.



6. Click **Save**.

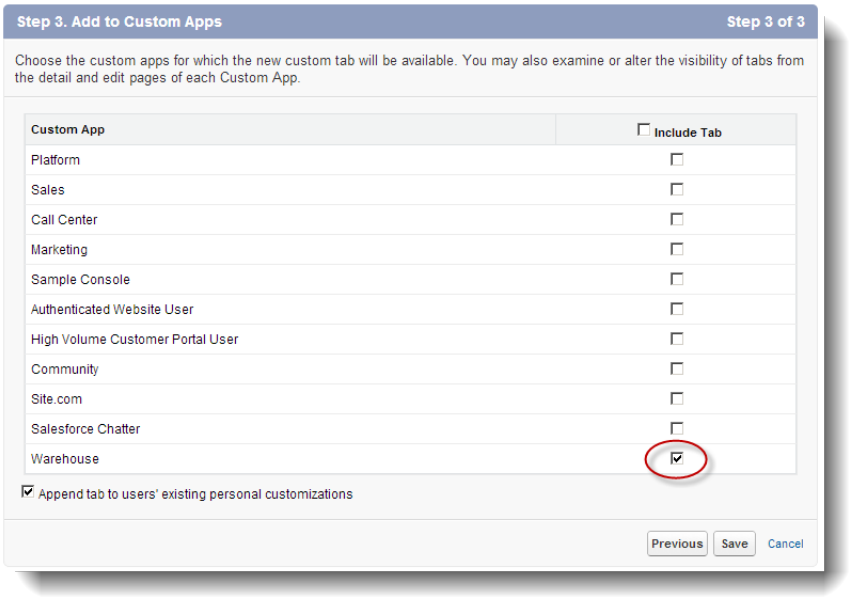
Tell Me More....

- The checkbox for vowel sounds ensures that the correct article is used: "a" or "an."
- The Auto Number data type tells the platform to automatically assign a number to each new record that is created, beginning with the starting number you specify. Because of the display format you chose, the invoice numbers will be INV-0000, INV-0001, and so on.
- You could have started invoices at any number, but we started invoices at INV-0000 to remind you that the platform is zero-based.

Step 2: Add an Invoice Tab to the App

When you click the Merchandise tab, a list of Merchandise records appears. Similarly, you need to create a tab that displays Invoices.

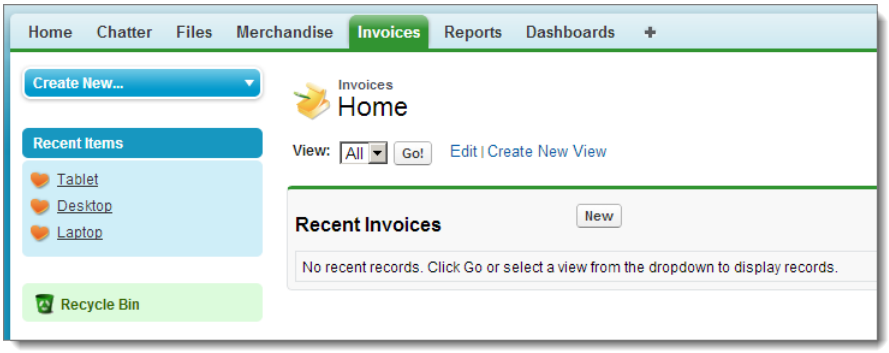
1. If you don't see **Launch New Custom Tab Wizard**, from Setup, enter *Tabs* in the Quick Find box, select **Tabs**, and then click **New** in the Custom Object Tabs section. Then select your Invoice object.
2. In the Tab Style lookup, choose **Form** and click **Next** and then **Next** again.
3. It makes sense to display this new tab for the Warehouse app. On the Add to Custom Apps page, clear the checkbox next to all apps except Warehouse.
4. Click **Save**.



Step 3: Reorder Tabs in the App

Take a look at the tabs across the top of your screen and you see the new Merchandise tab isn't next to the Invoice tab. You can put tabs in any order you like, so go ahead and put them next to each other.

1. From Setup, enter *Apps* in the Quick Find box, select **Apps**, and then click **Edit** next to your Warehouse app.
2. In the Selected Tabs list, select Invoices and use the up arrow to move it under Merchandise.
3. Click **Save** and then take a look at the tabs.



Step 4: Add a Status Field to the Invoice Object

If you try to create an invoice now, you won't be impressed. There aren't any fields that you can modify because they are all standard, auto-managed fields. In this step, you extend the Invoice object to add a new Status picklist field to track the status of each invoice.

1. In Setup, enter *Objects* in the Quick Find box, then select **Objects** and then click **Invoice**.
2. Scroll down to the Custom Fields & Relationships related list and click **New**.
3. For Data Type, select Picklist and click **Next**.

4. Fill in the custom field details.
   a. `Field Label`: *`Status`*
   b. Type the following picklist values in the box provided, with each entry on its own line.

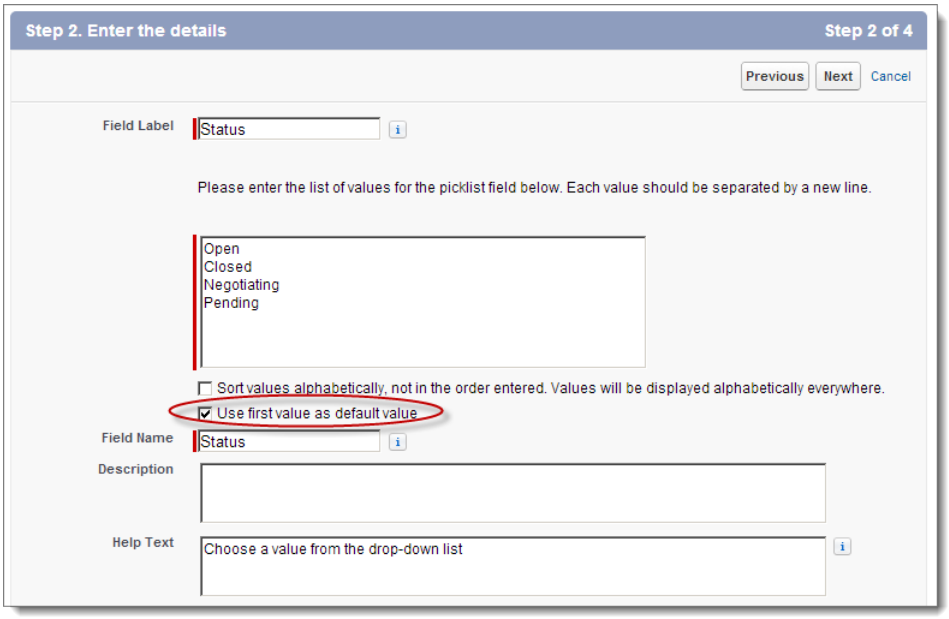
      `Open`

      `Closed`

      `Negotiating`

      `Pending`

   c. Select `Use first value as default value`.
   d. In the `Help Text` field, type *`Choose a value from the drop-down list`*.



5. Leave the defaults for the remaining fields and click **Next**, **Next**, and **Save**.

Tell Me More....

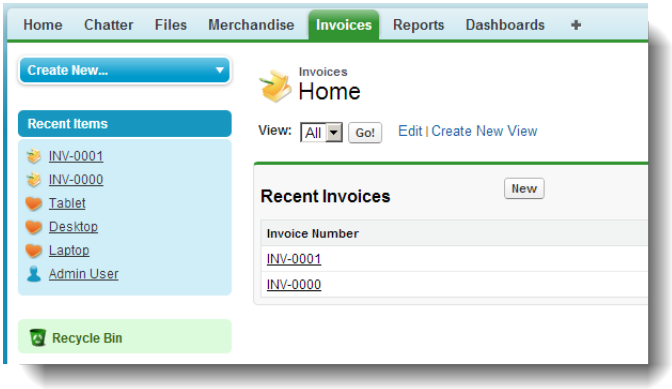
Before moving on to the next step, recall the Help text you added to the `Status` field. When users hover their pointer over the `Status` field, a pop-up bubble appears with whatever help text you specify. Although it's beyond the scope of this tutorial, understand that you can easily create unique translations for app labels and help text so that the app supports multiple languages, again, without writing a single line of code. Very cool.

Step 5: Try Out the App

Although your app isn't completely done yet, you can create invoices and save them. It's not a problem that the invoice is still missing some fields. When you add more fields to the Invoice object, the new fields automatically appear on the records that already exist.

1. Click the Invoices tab.
2. Click **New**. Notice that you can choose a status for the invoice, but leave it as Open.

3. Click **Save**.
4. Click the Invoices tab again and notice there's a new invoice, with the number INV-0000. Create another new invoice, this time with a Closed status.
5. Click the Invoices tab again and see your two invoices.



The database is starting to look better, but it's still incomplete. An invoice is made up of line items that list the type and quantity of merchandise being ordered. In the next tutorial, you'll add another object—Line Items—and then relate that object to the other ones we've created.

Tell Me More....

- You only have a few records so far, but how would the page look if you had hundreds of records? Thankfully, the default list view for a tab shows you only the most recent records you touched and lets you page through sets with standard navigation controls.
- Another built-in feature is *list views*. A list view is a customized presentation of data that shows only the fields you want, based on filters you define. For example, suppose you're only interested in open invoices with a price greater than $1000. You can create a custom list view that shows exactly those records. This is covered in a later tutorial.

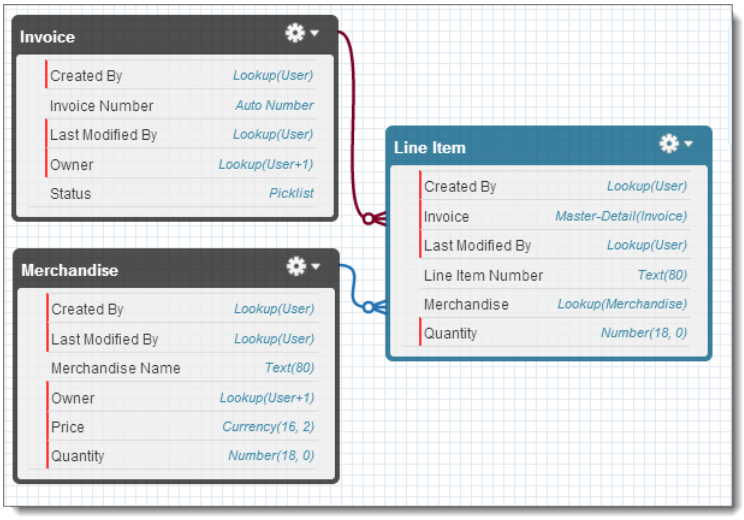
Relate Objects

Level: Beginner; **Duration:** 10–15 minutes

In previous tutorials you created objects that stood on their own. The fields on the Merchandise object had no relation to the fields on the Invoice object. In this tutorial, you create a Line Item object, and what's special about this new object is that its fields are related to both the Invoice and Merchandise objects.

- An invoice has one or more line items. In fact, you might say that a particular invoice "owns" its line items. That kind of relationship is called a *master-detail relationship*, where the detail records refer to a master record.
- Line items also relate to merchandise through another kind of relationship called a *lookup*. You already saw something similar in the `Status` field. When you create a new invoice, you can choose a status from the picklist. A lookup field is different because the values come dynamically from a custom object rather than statically from a picklist.

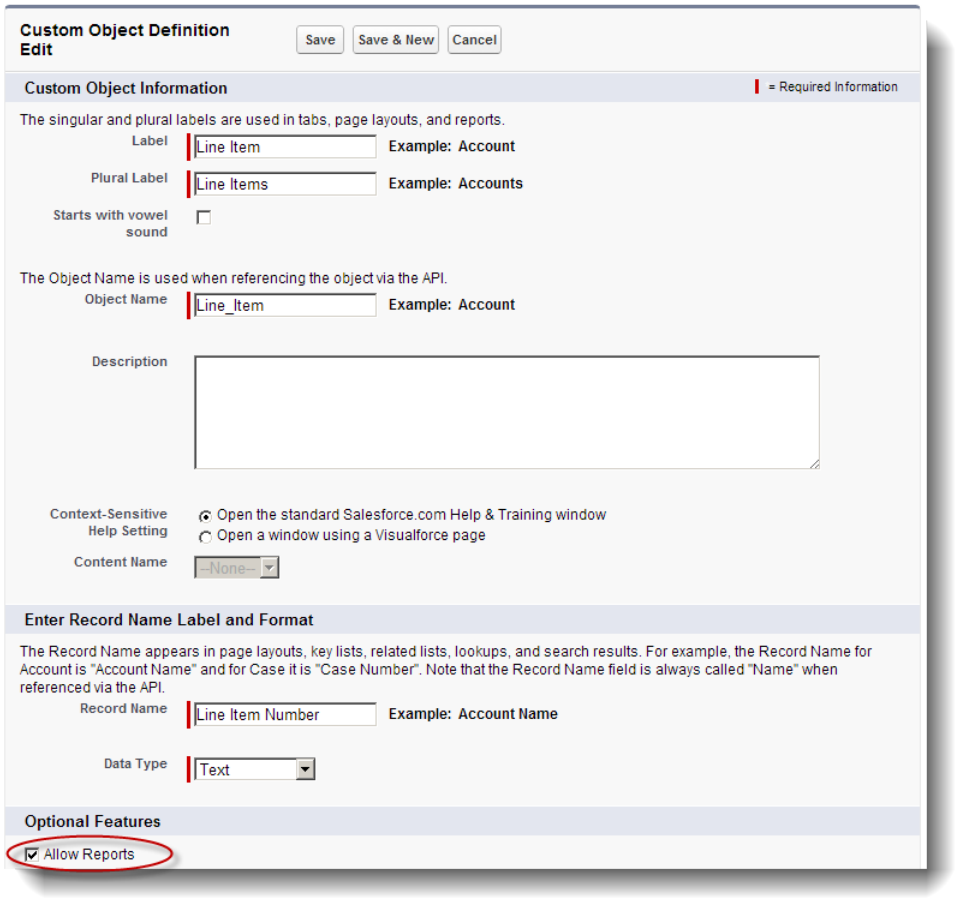
Master-detail relationships and lookups might seem confusing now, but once you implement them, it will all be very clear.



Step 1: Create the Line Item Object

Each invoice is made up of a number of invoice line items, which represent the number of merchandise items being sold at a particular price. You are first going to create the Line Item object, and then relate it to the Invoice and Merchandise objects.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**.
2. Click **New Custom Object** and fill in the custom object definition.
   - Label: *Line Item*
   - Plural Label: *Line Items*
   - Record Name: *Line Item Number*
   - Data Type: *Text*



3. In the Optional Features section, select `Allow Reports`, and click **Save**.

Tell Me More....

You might be wondering why Line Item Number is a text field, when what you enter is a number. If line items are numbered, why not make it an auto-number field, like Invoice? The short answer is that it's easier to work with text when working with records, and this tutorial is intended to be easy.

Step 2: Add a Quantity Field

Every line item needs to track the quantity ordered. So the next thing you need to do is add a Quantity field. Recall that the Merchandise object also has a Quantity field to track how many items are in stock, and the steps to create the field are the same.

1. On the Line Item detail page, scroll down to Custom Fields and Relationships and click **New**.
2. For Data Type, select `Number` and click **Next**.
3. Fill in the field details:
   - `Field Label`: *`Quantity`*
   - Select `Required`
4. Leave the defaults for the remaining fields, and click **Next**, **Next**, and then **Save**.

Step 3: Relate Line Items to Invoice

Now that you have all the objects representing the data model, you need to relate them to each other. Line items are related to both an invoice (an invoice is composed of a number of line items) and merchandise (a line item takes its price from the merchandise).

1. On the detail page of the Line Item object, scroll down to the Custom Fields & Relationships related list and click **New**.
2. For Data Type, select `Master-Detail Relationship` and click **Next**.
3. In the Related To field, select your Invoice custom object, and click **Next**.
4. For `Field Label` and `Field Name` enter *`Invoice`*.
5. Accept the defaults on the next three screens by clicking **Next**.
6. On the final screen click **Save & New**.

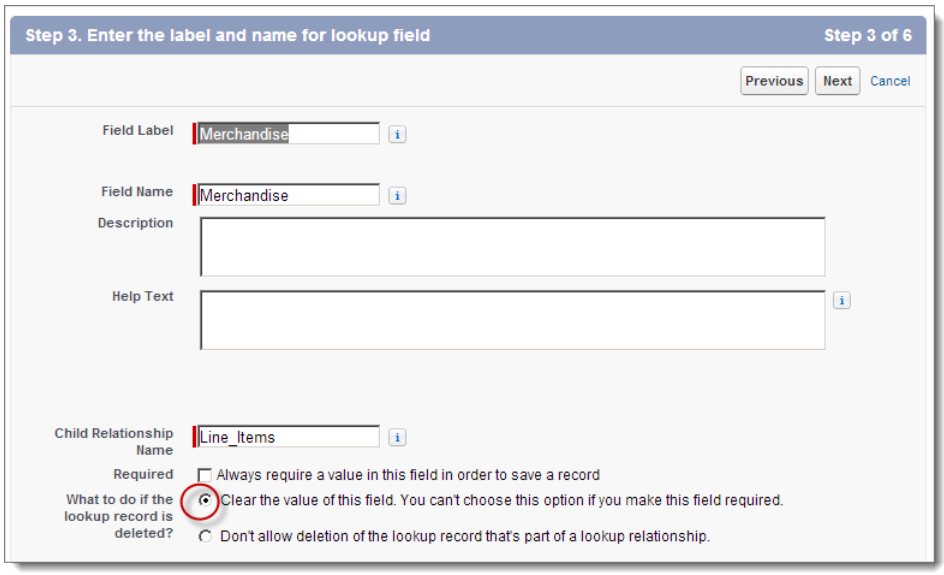
Tell Me More....

One way to think of this master-detail relationship is that an invoice now "owns" its line items. In other words, an invoice can now contain multiple line items. One of the neat things about master-detail relationships is that they support roll-up summary fields, allowing you to aggregate information about the child records. You'll use that feature in a later tutorial.

Step 4: Look Up Merchandise Items

The other kind of relationship you need to create is called a *lookup*. As the name implies, the field gets its information by looking it up dynamically in another object. In the last step, you used **Save & New**, so you should already be on the New Custom Field dialog.

1. For Data Type, select `Lookup Relationship` and click **Next**.
2. In the Related To field, select Merchandise and click **Next**.
3. For `Field Label` and `Field Name` enter *`Merchandise`*.
4. Verify your screen looks like the following image and then click **Next**.



5. Accept the defaults on the subsequent screens by clicking **Next**, and **Next** again.

6. On the final screen, deselect the checkboxes for `Merchandise Layout` and `Append related list to users' existing personal customizations` (you don't want a list of line items on the Merchandise page).
7. Click **Save**.

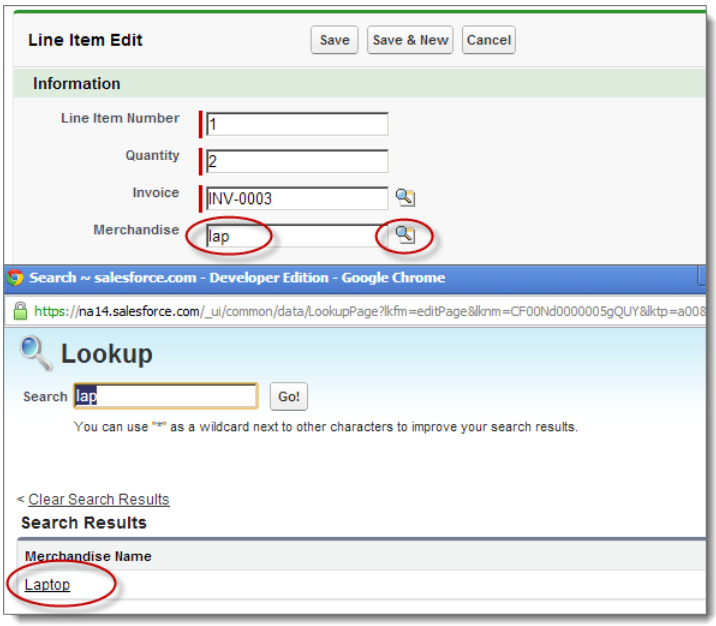
Tell Me More....

At this point you have two relationships, a master-detail relationship that allows an invoice record to contain many line items, and a lookup relationship that relates a particular line item to a piece of merchandise.

Step 5: Try Out the App

As you saw in the previous tutorial, the platform automatically generates a user interface for the objects you create, so that you can view, edit, delete, and update records. Because you also related the objects, the user interface provides a way of navigating between related records as well. You can see how all that works by creating another invoice record.

1. Click the Invoices tab and then **New** and **Save**.
2. Click **New Line Item**.
3. For `Line Item Number`, type *1*.
4. For `Quantity`, type *2*.
5. In the `Merchandise` field, type the first few letters of *`laptop`* and click the Find icon.



6. Click **Laptop** and then **Save**.

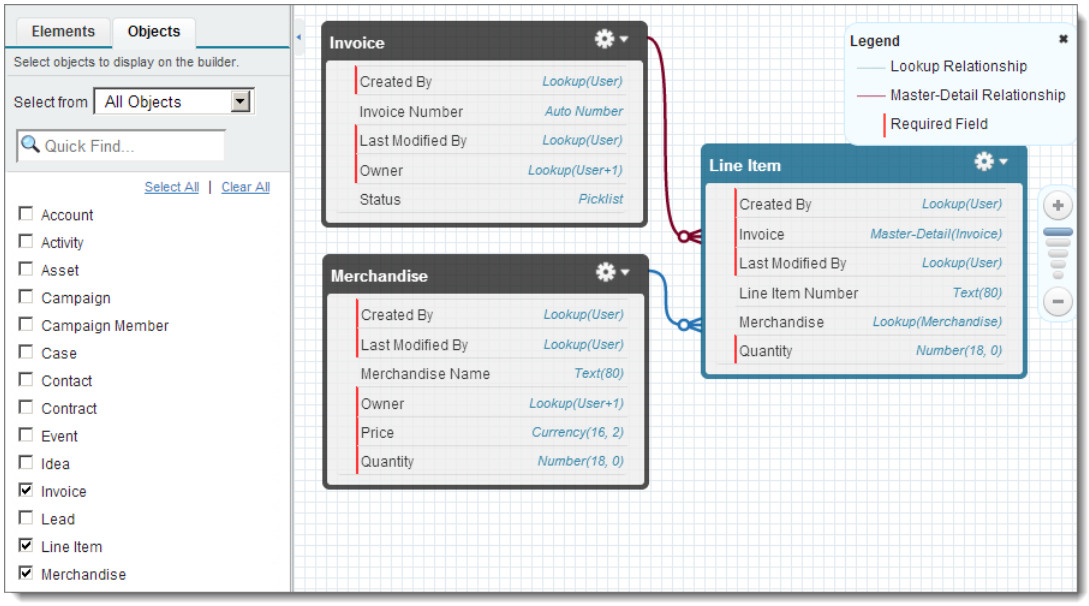
Tell Me More....

If you're familiar with the products in your inventory, you can type the first few letters of a piece of merchandise and click **Save**. You don't need to click the Find icon, the system automatically finds the merchandise and adds it when you save the record. There's a lot of built-in functionality!

Step 6: View the Schema

You now have three custom objects, several fields, and two kinds of relationships. If you have all of that in your head, awesome. However, most people find this is an ideal time to use the old adage "a picture is worth a thousand words."

1. From Setup, click **Schema Builder**.
2. In the left pane, click **Clear All** to remove the standard objects from the schema.
3. Select the checkboxes for `Merchandise`, `Invoice`, and `Line Item`.
4. Click **Auto-Layout** to arrange the objects, or manually adjust the layout if necessary.



The Schema Builder shows your objects, fields, and relationships in a standard entity-relationship diagram. In a relationship, the "crows feet" at the end of the line tell you which side is the "many" side of a one-to-many relationship (one invoice can contain multiple line items). When you're done looking at the schema, click **Close**.

Note: Schema Builder isn't just for viewing your schema, it also supports drag-and-drop development for creating new objects and fields. However, unlike the wizards you used so far, fields added using Schema Builder are not automatically added to page layouts. You must configure page layouts before your new fields are visible to users. Field visibility and page layouts are covered in subsequent tutorials.

Summary

At this point, you have created three custom objects: Merchandise, Invoice, and Line Item. On each of those objects you created custom fields to represent text, numbers, and currency. Two of these fields have system-generated values: the `Status` picklist, which defaults to Open, and the `Invoice Number` field, which is automatically assigned by the Auto-number data type. You also created user-defined fields, such as the `Quantity` entered for each line item. Finally, you expanded on the basic data model by creating two fields that get their values from other objects; these are the relationship fields you created in this tutorial.

The master-detail relationship allows you to aggregate information, so that an invoice can contain multiple line items, and those line items can be aggregated. The lookup relationship allows you to pull in dynamic content, so that each piece of merchandise on a line item automatically gets a price. The relationships also provide additional benefits. You can add up the price of each invoice line item

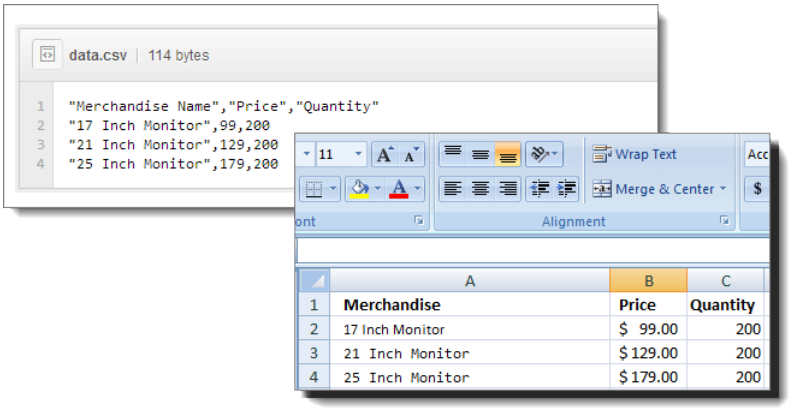
and create a sum total for the invoice, and you can navigate to the related records in a user interface. You'll learn how to do those things declaratively in the next tutorial, and later in code as well. Onward!

Load Data Using the Custom Object Import Wizard

Duration: 5–10 minutes

Most organizations keep important data in all sorts of places, including documents and spreadsheets. In this tutorial, you learn how to load data that currently lives in a personal spreadsheet into the Warehouse app, where everyone in your organization can view and manage the data.

There are several ways to load data, and this tutorial shows you only one method, using a the Custom Object Import Wizard. This wizard uses a CSV file as its source. A CSV file is a plain text file with each field separated by commas—thus the name "comma-separated values."



Prerequisites

Text Editor

This tutorial requires a text editor and the ability to upload a file from your computer. If you're using a tablet or mobile device, you may not be able to complete this tutorial depending on the capability of the device.

Step 1: Create the Data File

The first step is to make a simple data file that you can use for this tutorial.

1. To save you time, download the necessary CSV-formatted text file, from this URL.
   https://raw.github.com/joshbirk/workshop2013/master/files/Merchandise.csv
2. Right-click and save the file locally. It should look like:

```
"Merchandise Name","Price","Quantity"
"17 Inch Monitor",99,200
"21 Inch Monitor",129,200
"25 Inch Monitor",179,200
```

Tell Me More....

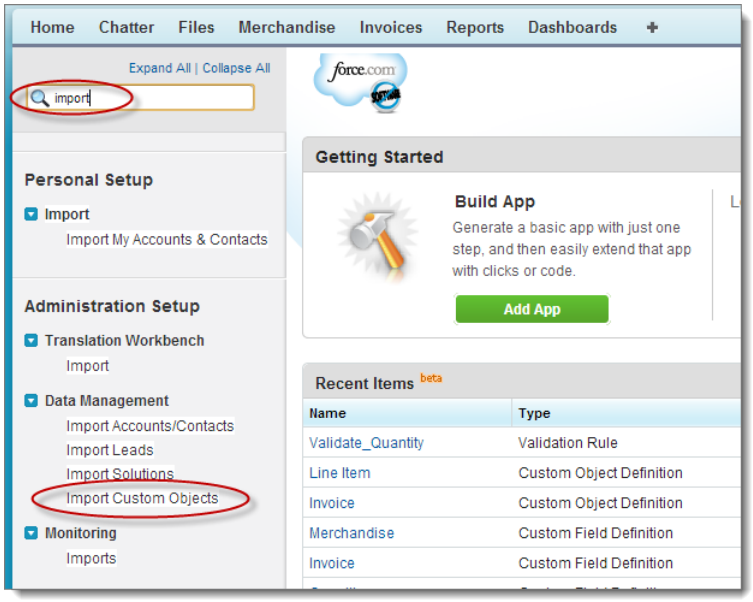
Note that in the CSV file:

- The field names are on the first line. These names match the labels for fields in the Merchandise object.
- Text fields are delimited by quotes, allowing you to include spaces and special characters inside a text field. Fields that have a number data type don't require quotes.

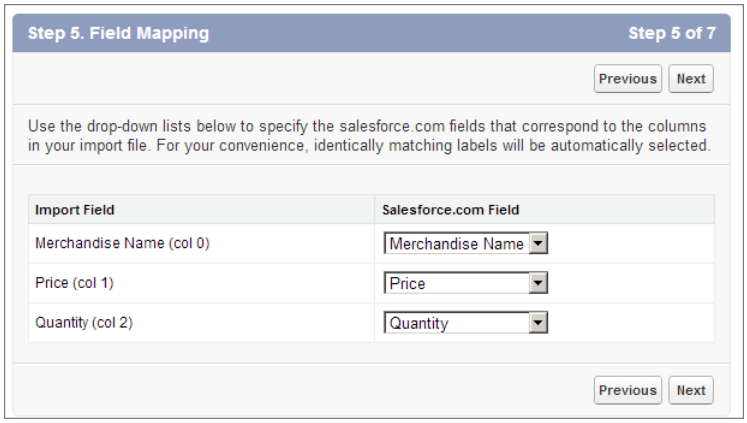
Step 2: Load the Data

Loading data from a CSV file into a custom object is simple using the Custom Object Import Wizard.

1. From Setup, in the Quick Find field, type `import` and then click **Import Custom Objects**



2. At the bottom of the page, click **Start the Import Wizard!**
3. When the wizard starts, select `Merchandise`, then click **Next**.
4. Select `No`, and then click **Next**.
5. Select `None`, and then click **Next**.
6. Click **Choose File** or **Browse...** and select the data file you created earlier, then click **Next**.
7. Notice on the Field Mapping step you can match headings in your CSV file to field names in Salesforce. That was already done in the CSV file, so you can click **Next**.



8. Click **Import Now!** and then **Finish**.

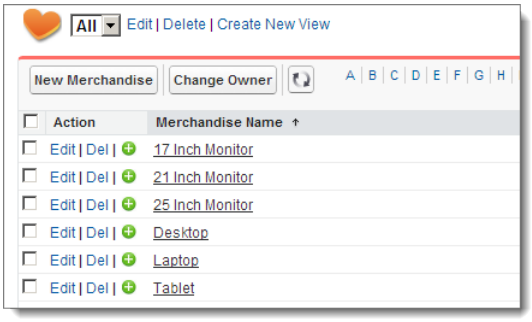
Tell Me More....

Once you finish the wizard, the platform queues the data load. For large sets of data, it may take a while for the data load to happen, and you'll be notified by email when the data load completes. If you want to monitor this process more closely, in Setup, click **Imports**.

Step 3: Try Out the App

Once the data load is completed, go back to your app and confirm that the new Merchandise records are in place.

1. Click the Merchandise tab.
2. Next to the View drop-down list, make sure All is selected and click **Go!**



CUSTOMIZE A USER INTERFACE

Level: Beginner; **Duration:** 30–40 minutes

At this point you already know how to create a basic app and do things like create and relate objects, add formulas and validation, and customize the standard user interface. In this tutorial, you'll go a step further by learning advanced point-and-click development to further enhance the underlying database and improve the UI.

Create Views of Data

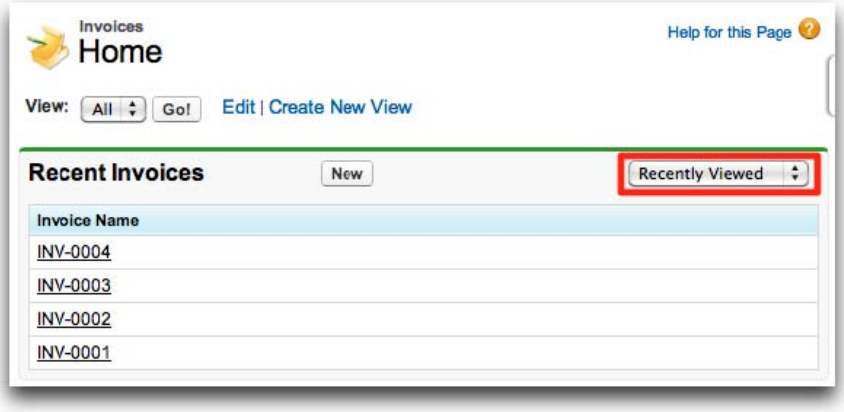
Level: Beginner; **Duration:** 5 minutes

A custom object tab in an application is a navigational element that, when clicked, displays data for the corresponding object. For example, when you click on **Invoices** in the Warehouse app, you see a default list view of the most recent invoices that you've touched. This tutorial teaches you more about views and how to create custom views to meet specific needs.

Step 1: View a List of Invoices

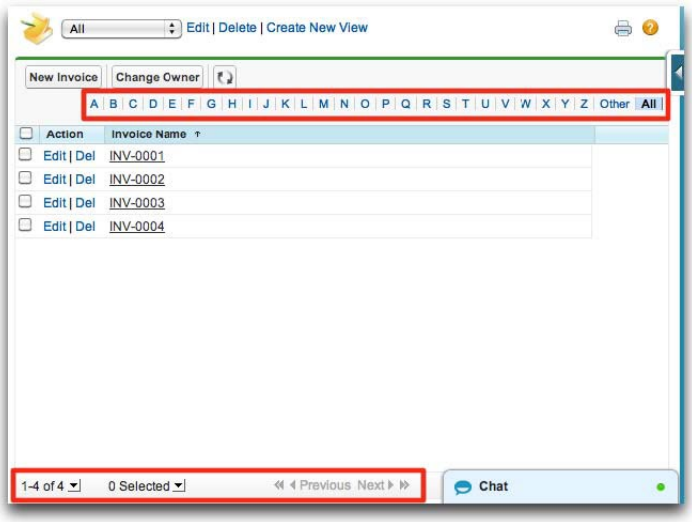
Notice how the All view sorts records alphabetically and provides navigation controls for large lists. So, right out of the box, you have several default views that list invoices. But what if you need a custom view? No problem.

1. Select the Warehouse app and click the Invoices custom object tab. By default, the Recent Invoices view displays your most recently viewed records — notice the pick list in the upper right corner of the view. You can update the view display by changing the picklist to Recently Created and various other options.



2. Click **Go!** to switch from the Recent Invoices view and display a list of All invoices.

   Notice how the All view sorts records alphabetically and provides navigation controls for large lists.



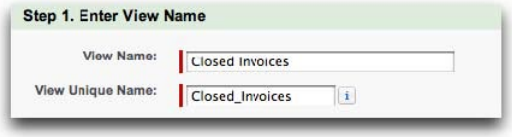
Tell Me More....

Right out of the box, you have several pre–built views that list invoices, with navigation and sorting. But what if you need a custom view? Let's say you want to see only closed invoices. No problem.

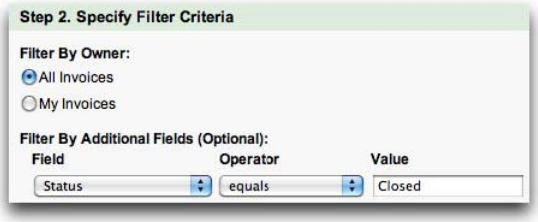
Step 2: Create a New View

In this step, you create a custom view that only shows invoices with a status of `Closed`.

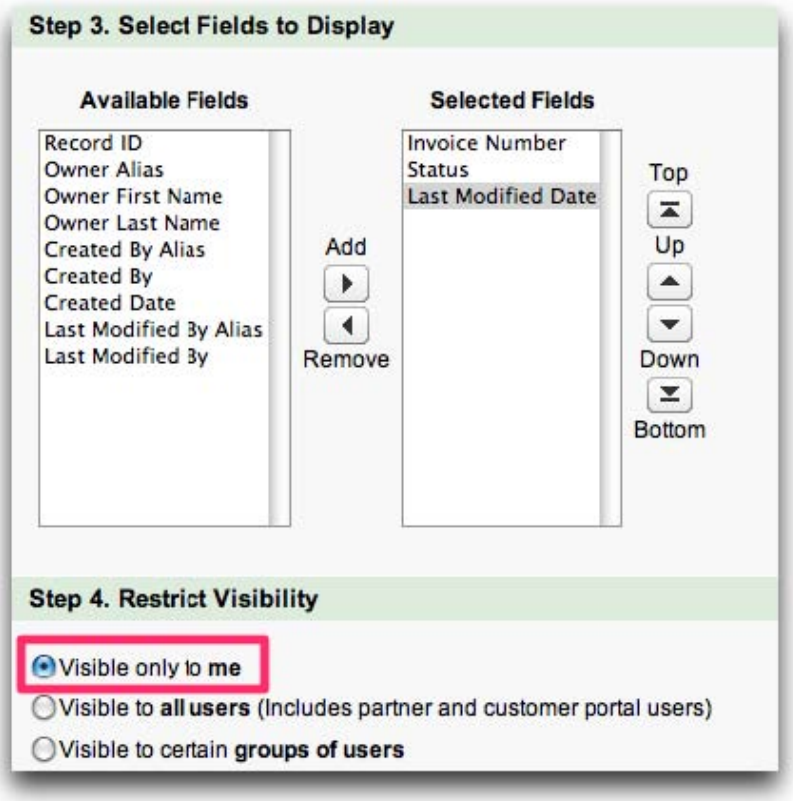
1. On the Invoices tab, click **Create New View** and name it *`Closed Invoices`*.



2. Select `All Invoices`, and specify a filter criteria: *`Status equals Closed`*.



3. A custom view shows only the fields you select. Update the Selected Fields list with only `Invoice Number`, `Status`, and `Last Modified Date`.



4. Select `Visible only to me` and then click **Save**.

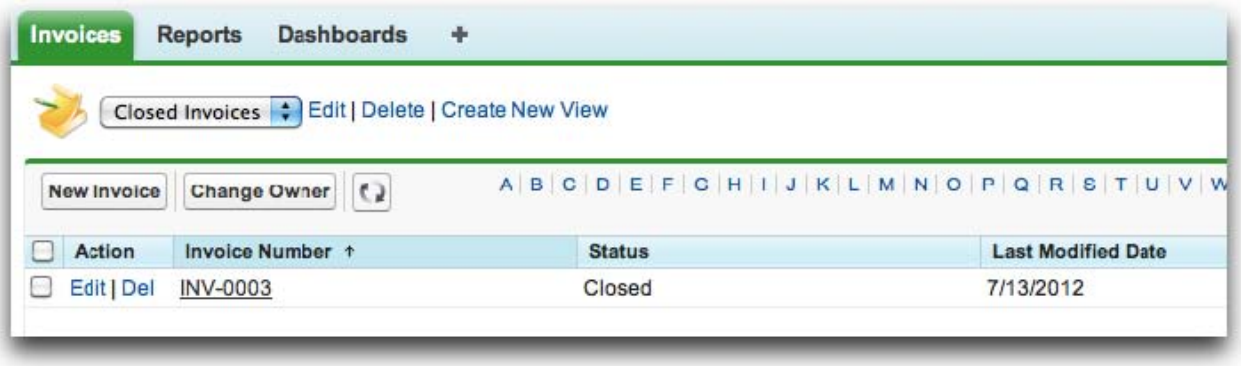
Tell Me More....

Notice that you restricted the visibility of this view. That's a really important feature because you can create views of data for everyone in your company, groups of people, or a view that only you can see.

Step 3: Try Out the App

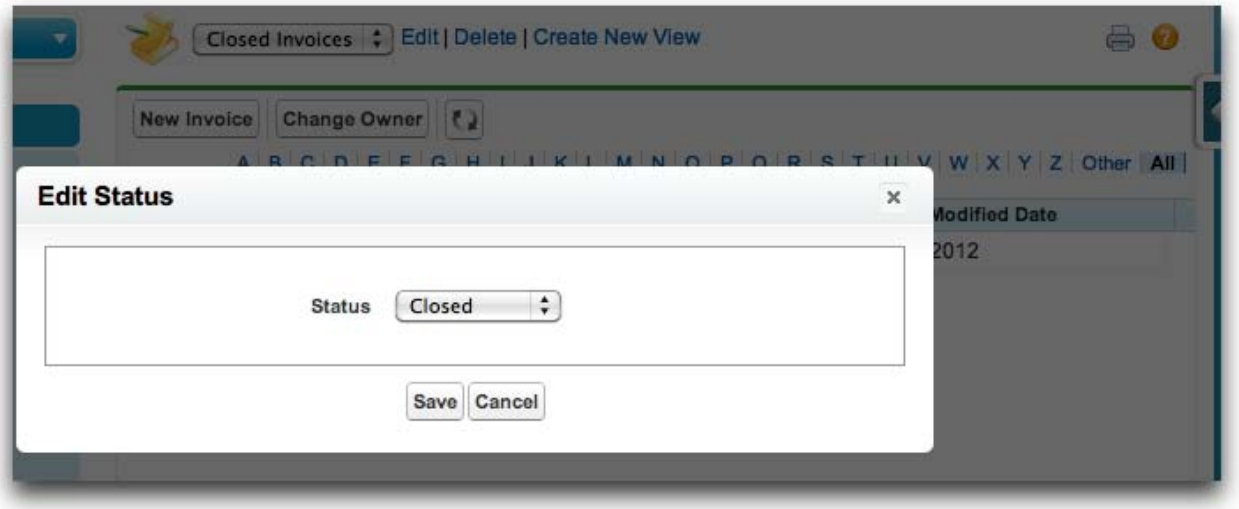
In this step, we'll test out our new view in the app.

1. To display the new Closed Invoices view from anywhere in the app, click the Invoices tab, select Closed Invoices, and click **Go!**
2. When your screen refreshes, you might not have any invoices in the new Closed Invoices view. If this is the case, edit one or more invoices and change the status to Closed. Now go back to your view of closed invoices and notice the power of custom views.



Tell Me More....

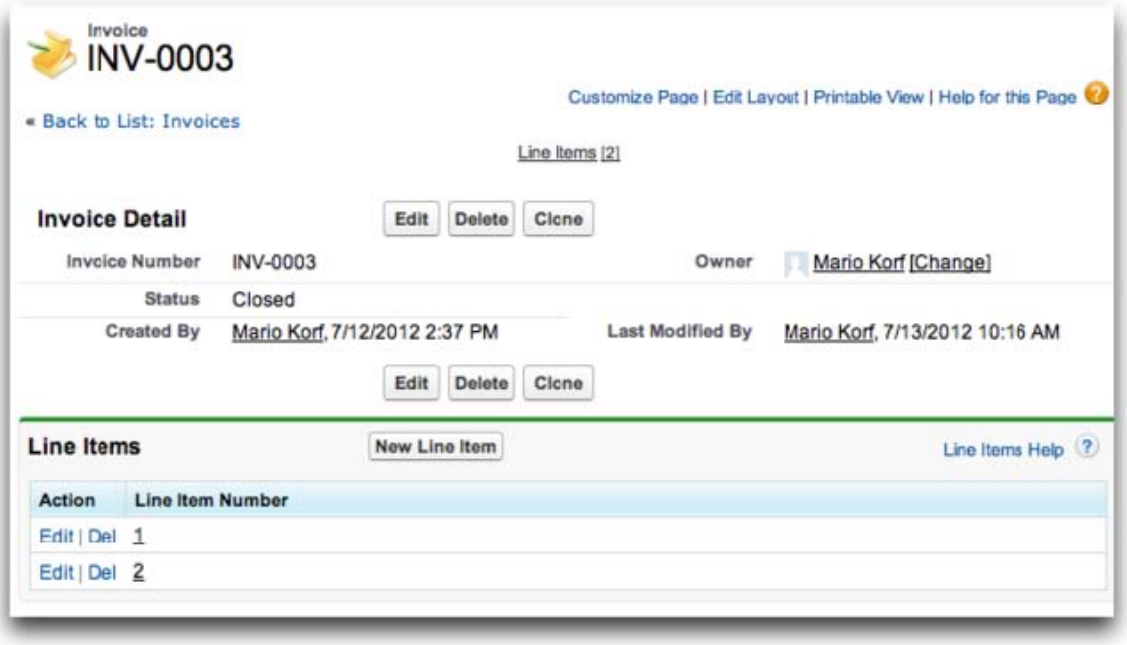
At this point, you might think that views are read-only displays of data for a custom object—not so. In the new Closed Invoices view, move over the `Status` field for a specific invoice. Notice that a pencil icon appears in the field, indicating that the field is editable inline, right from the view. Double-click the Status field and the app provides you a way to edit the field.



Modify a Page Layout

Level: Beginner; **Duration:** 20-30 minutes

In Create Views of Data on page 29 you learned how to create a customized view for *lists* of data. However, you can also customize what's on the detail page for a particular record, or the *page layout*. Click an invoice and take a look at the default page layout for all invoices, which should look similar to this image:



This tutorial teaches you more about page layouts and how to modify them.

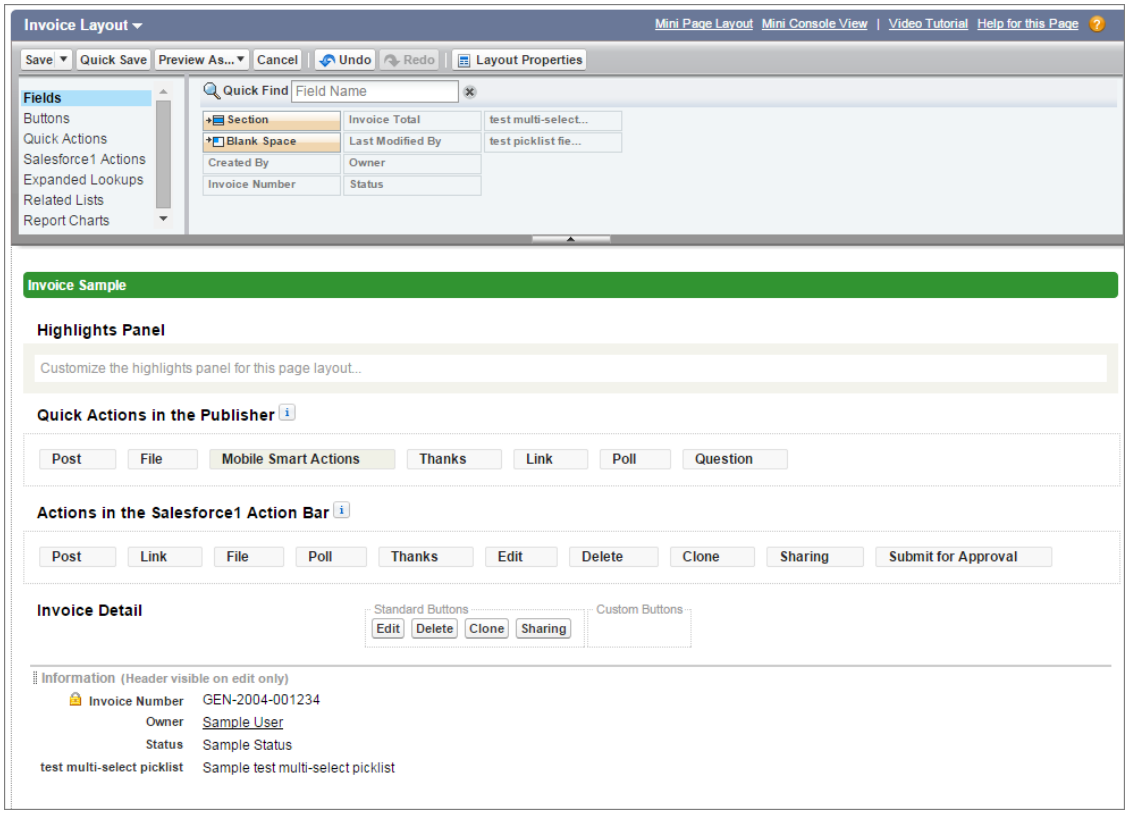
Step 1: Open the Page Layout Editor

Use one of the following ways to open the page layout editor.

- While on the record page that you want to modify:
  - Click **Edit Layout**.
  - Click the Quick Access menu on the right, and choose **Edit Layout**.
- From Setup, enter *Objects* in the Quick Find box, then select **Objects**, click the object you want to change the layout of, scroll down to the Page Layouts section, and then click **Edit** next to the layout you want edit.

Step 2: Understand a Page Layout

The editor has upper and lower sections. The upper section is a retractable toolbox called the *palette*. The lower section is the preview pane. When you scroll down the page, the palette moves with you, which makes it easy to edit longer pages.

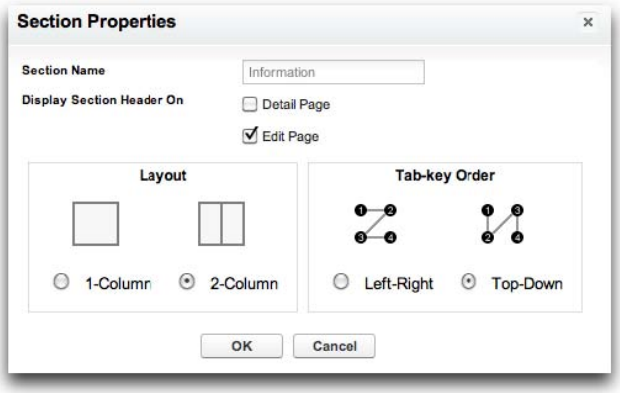


In the page layout, several sections organize related information.

- The Highlights Panel is useful for displaying key information at the top of the page.
- The Quick Actions in the Salesforce Classic Publisher section is useful for customizing the actions that appear in the publisher in the full Salesforce site.
- The Salesforce1 and Lightning Experience Actions section enables you to customize the actions that appear in Salesforce1.
- At the top of the Invoice Detail is the area for standard buttons (Edit, Delete, and so on) and custom buttons.
- Next comes the Invoice Detail, which has three default sections.
  - Information typically contains fields that users can manipulate at some point during the lifecycle of a record (creation and updates). By default, this section has two columns for fields.
  - System Information typically contains fields that the platform automatically maintains—fields that users cannot edit. This section is also a two-column layout.
  - Custom Links typically contains custom navigation links.
- Below Invoice Detail is a section for Mobile Cards. By default, this section is empty. Mobile cards only appear in Salesforce1.
- Last on the page is a related list for related Line Items.

You can make many changes to the page layout.

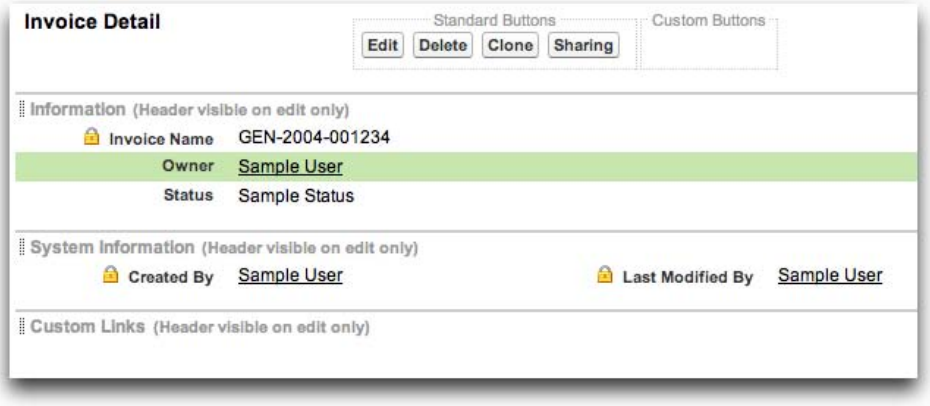
1. Hover over a section title. The mouse pointer changes, indicating that you can drag the section to a new location relative to other sections.
2. Hover over the upper-right corner of any section. Two buttons appear: one for removing the section (don't click it!) and another for editing its properties. Go ahead and click . You can now edit the name of the section (only for non-default sections), when to display the section header, the section layout (one or two columns), and the tab-key order among section fields. Click **Cancel**.



Step 3: Rearrange Fields on a Page Layout

In this step, make some simple changes to the Invoice Detail area of the page layout.

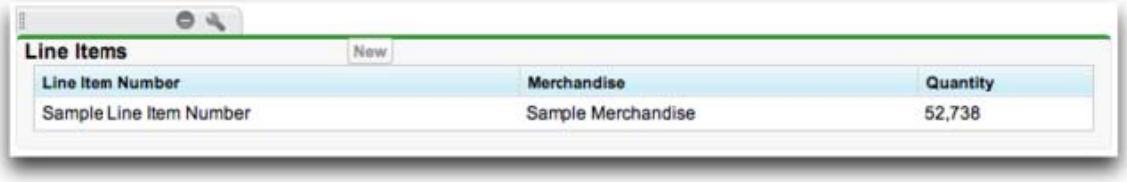
1. Click for the Information section (see above if you forgot how to find this) and change the section layout to one column. Click **OK**.
2. Drag the `Owner` field above the `Status` field. When you're done, the modified Invoice Detail area should look like this.



Step 4: Add Fields to the Related List

As it is now, the related list of Line Items is not very informative—it only has the line item numbers. In this step, improve the related list by adding some new fields.

1. Click **Related List Properties** (the wrench icon above the Line Items section), add `Merchandise` and `Quantity` to the Selected Fields list, then click **OK**. When you return to the page layout editor, the related list preview should now appear similar to this:

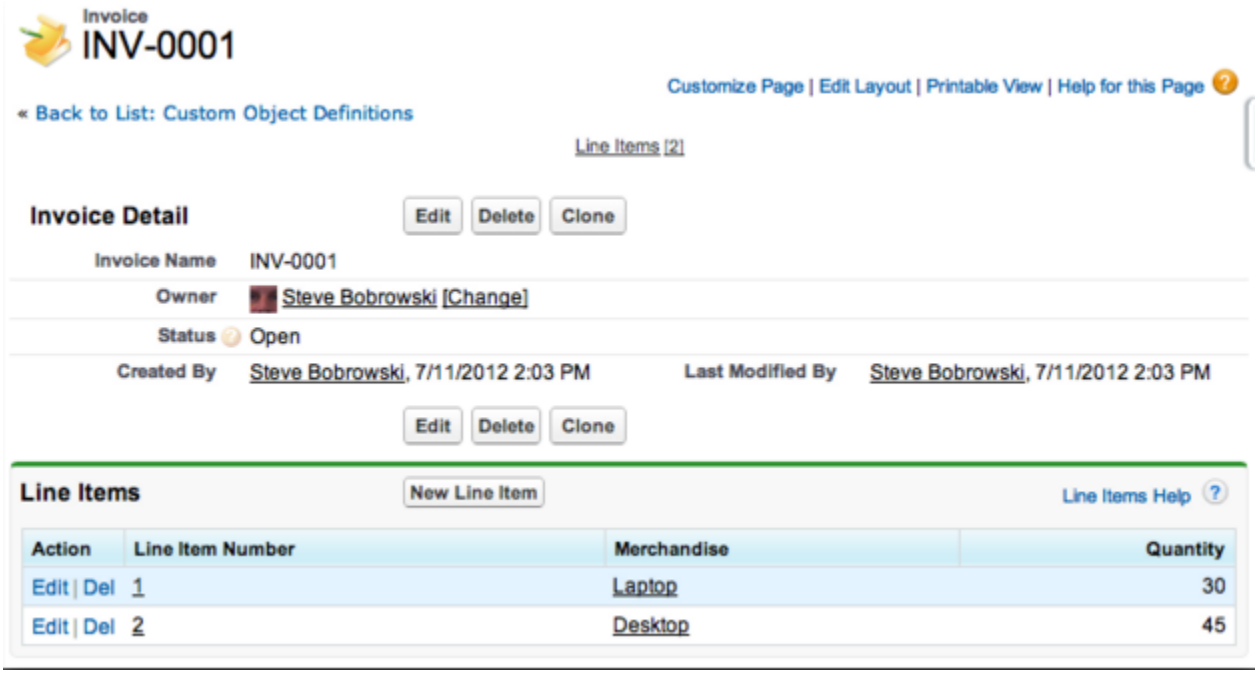


2. That's it—you're done modifying the page layout. At the top of the page, in the toolbox, click **Save**.

Step 5: Try Out the App

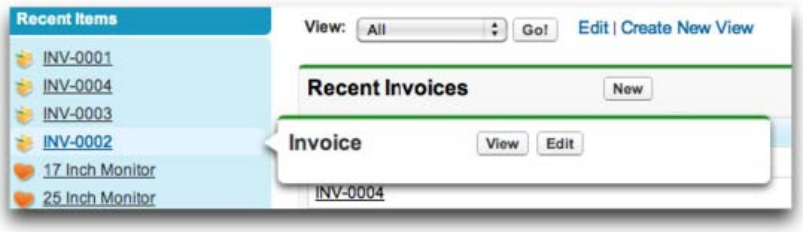
Check out the results of your work.

1. Click the Invoices tab to return to your app, and then click an invoice that has at least one line item.
2. Notice the rearranged fields in the Invoice Detail area, as well as the new fields in the Line Items related list.



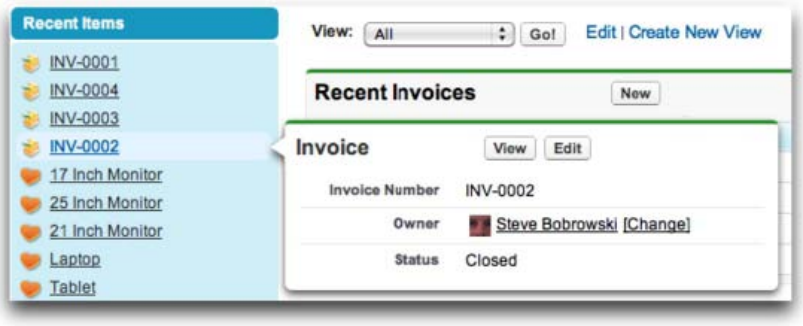
Step 6: Edit a Mini Page Layout

When you are in the Warehouse app, notice the Recent Items sidebar. Specifically, move over a recent invoice and notice that you get a mini page popup that previews the invoice information. See below—that's not very informative, is it?



It's easy to change this default mini page layout as well.

1. Return to the page layout editor for Invoice.
2. Click **Mini Page Layout** at the top of the palette.
3. Add `Invoice Number`, `Owner`, and `Status` to the list of selected fields, and then click **Save**. The improved popup should look more like:



Customize a Layout for Mobile Access

Level: Beginner; **Duration:** 25–30 minutes

A well-designed page layout can often be used by both desktop and mobile devices. So far in this tutorial, none of the objects are large and unwieldy on a mobile screen. However, you can imagine that an object with a hundred fields might be difficult to use on a phone. In cases like this, it's useful to create a mobile-specific page layout. Just like the page layout you modified earlier, a mobile-optimized layout can be assigned to different roles, so that people who primarily use a phone get the mobile version, while desktop-only users get the standard version.

In this tutorial you learn how to:

- Modify an existing page layout so that it's optimized for a mobile device—If your users access your app from desktop and mobile devices, then you might want to optimize your page layouts so that they work with various form factors. However, if your users are entirely or mostly mobile, they might find a mobile-specific layout is more productive.
- Create a compact layout—Compact layouts determine the fields that show up in a record's highlights area and the record's feed items in the full site. Compact layouts are a great way to display a record's key fields at a glance.
- Add mobile cards to the related information page—Mobile cards can show lookup information or Visualforce pages.

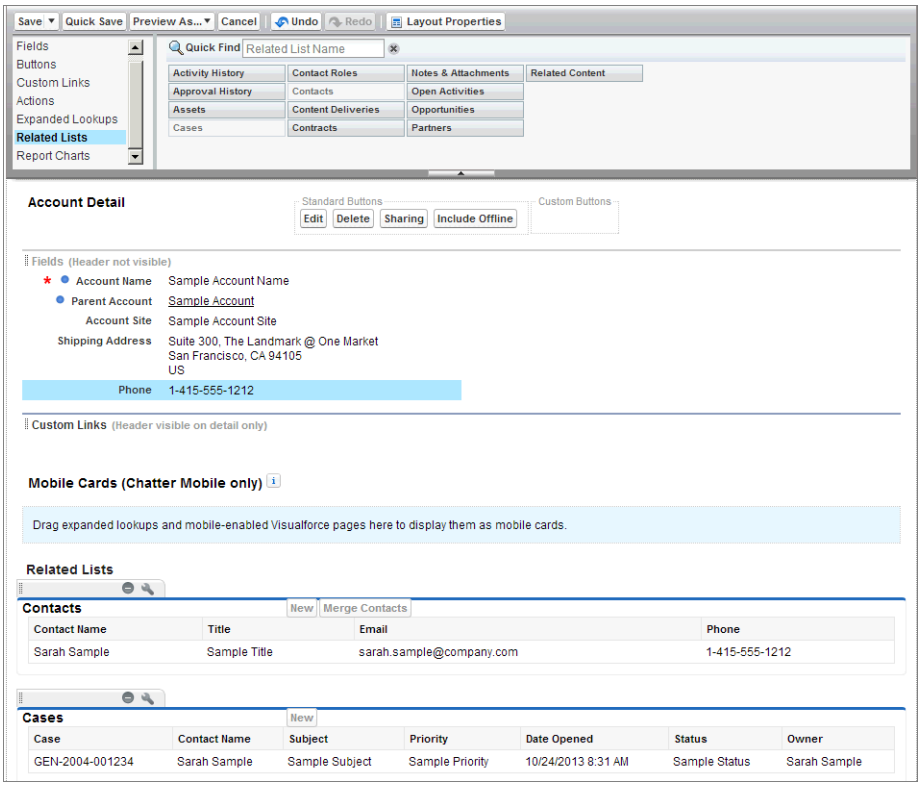
Note: There's another kind of mobile layout called a global publisher layout, which determines where global actions go. You'll learn about that layout when you create global actions in Quickly Create Records Using Global Actions on page 94.

Step 1: Create a Page Layout for Mobile Users

The objects in the Warehouse app don't have enough fields to demonstrate why a mobile page layout is necessary, so in this step, you customize the page layout for the Account object. This object isn't used in the Warehouse app, but it's a useful exercise for any mobile layout.

1. First navigate to an existing account by clicking the (+) tab and then **Accounts**.

2. In the View drop-down list, select All Accounts.
3. Click the first account: **Burlington Textiles Corp of America**. Notice that there's a lot of information on this page, and it might be a challenge to navigate on a small screen.
4. From Setup, enter `Accounts` in the `Quick Find` box, then select **Page Layouts**.
5. Name the page layout `Account Mobile Layout` and then **Save**.
6. Add a few fields that are important to mobile technicians. Drag the `Account Site`, `Shipping Address`, and `Phone` fields onto the Fields section of the preview pane.
7. Click the **Related Lists** category in the palette, and drag the **Cases** and **Contacts** elements to the Related Lists section. Related lists show up on the record related information page in Salesforce1. When mobile users assigned to this page layout views an account record's related information, they'll see preview cards they can click to see information about the cases and contacts for that account.



8. Click **Save** and then **No** when asked if you want to override users' customized related lists.
9. Now you need to assign the mobile-optimized page layout to a user profile. Click **Page Layout Assignment** and then **Edit Assignment**.
10. Click **System Administrator**.
11. In the Page Layout to Use drop-down list, select Account Mobile Layout, and then click **Save**.
12. Now when you access the Account object, you'll do so through the mobile-optimized layout. Try it now by going to Salesforce1 and tapping **Accounts** in the navigation menu.

    Since you just accessed the Burlington Textiles Corp of America account from the full site, you should see that in the Recent Accounts list.

13. Tap that account.

Tell Me More....

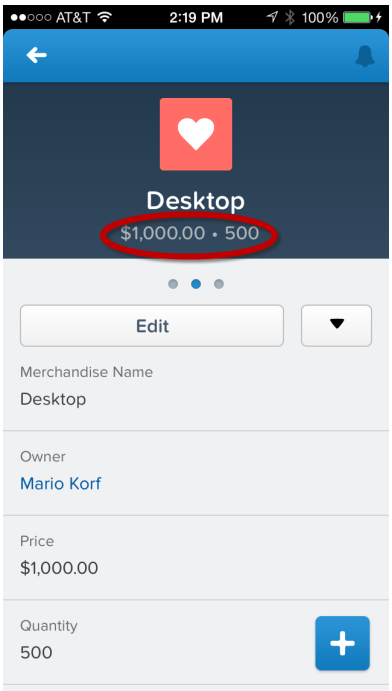
Normally, after creating a page layout for mobile users, you'd add it to a mobile user's profile. To keep things simple (so that you don't have to log out and switch users), you simply added the page layout to your own profile instead.

Step 2: Display Key Fields Using Compact Layouts

In the previous tutorial you learned how standard page layouts can be used to optimize a layout for mobile users. However, page layouts aren't the only thing used to customize how your data appears in a mobile environment. Salesforce1 uses *compact layouts* to display a record's key fields at a glance.

In this tutorial, you create a custom compact layout and then set it as the primary compact layout for the Merchandise object.

1. From Setup, enter `Objects` in the `Quick Find` box, then select **Objects**, then click the Merchandise object.
2. Scroll down to the Compact Layouts related list and click **New**.
3. In the Label field, enter `Merchandise Compact Layout`.
4. Move `Merchandise Name`, `Price`, and `Quantity` to the Selected Fields list, and then click **Save**.
5. Now you need to set the compact layout as the primary. Click **Compact Layout Assignment**.
6. Click **Edit Assignment**, select the compact layout you just created, and then click **Save**.



Tell Me More....

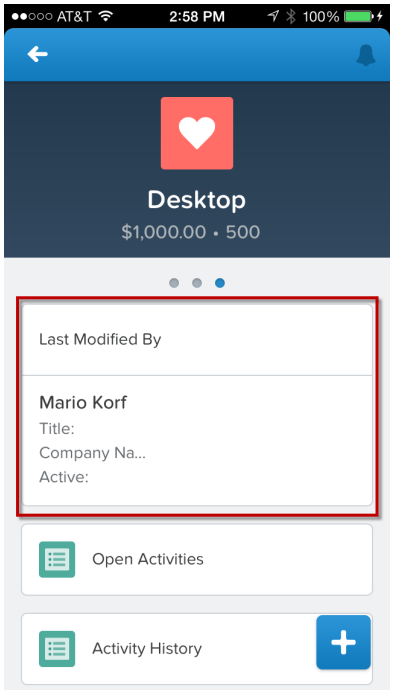
- In this exercise you only used three fields, but the first four fields you assign to your compact layout populate the record highlights section at the top of each record view.
- You don't need to create compact layouts for Salesforce1. If you don't create them, records display using a read-only, default compact layout. After you create a custom compact layout you can replace the default with your new layout.
- Compact layouts aren't just for mobile. When accessing Salesforce from a desktop browser, compact layouts determine which fields appear when a feed item is created.

Step 3: Add Mobile Cards to the Related Information Page

You've already seen the related information page in Step 3: Explore the Mobile App on page 10; this is the page that shows Activities by default. You navigate to the related information page by swiping left on the detail page for a record. Using *mobile cards*, you can add *related lookup cards* and *Visualforce page cards* to this record's related information page.

In this step, you add a related lookup card to the Merchandise object. Merchandise already has a lookup field that's automatically generated, Last Modified By, so you can use that.

1. Open the page layout for Merchandise from Setup by entering `Objects` in the `Quick Find` box, selecting **Objects**, and then selecting **Merchandise**.
2. Scroll down to the Page Layouts section and click the **Edit** link next to Merchandise Layout.
3. In the palette, click the **Expanded Lookups** category.
4. Drag `Last Modified By` to the Mobile Cards section, and then click **Save**.
5. To test it out, go back to your mobile device and look at a piece of merchandise.
6. Swipe left to get to get to the related information page and you'll see the mobile card you added.



Tell Me More....

- You don't have any Visualforce pages yet, but once you've enabled one for mobile, you can add those pages to the Mobile Cards section like you just did.
- You can also use the Mobile Cards section to add elements from the Components category. That category doesn't appear in this tutorial, because no components are available on custom objects.
- Unlike compact layouts, mobile cards only appear in Salesforce1.

Enable Social Collaboration

Level: Beginner; **Duration:** 5–10 minutes

Users can *follow* merchandise records and collaborate on them using Chatter. When you are following a record, the platform automatically pushes notifications about updates to you. The feed for the record becomes a running log where users can collaborate on the data record by posting comments, files, links, and more.

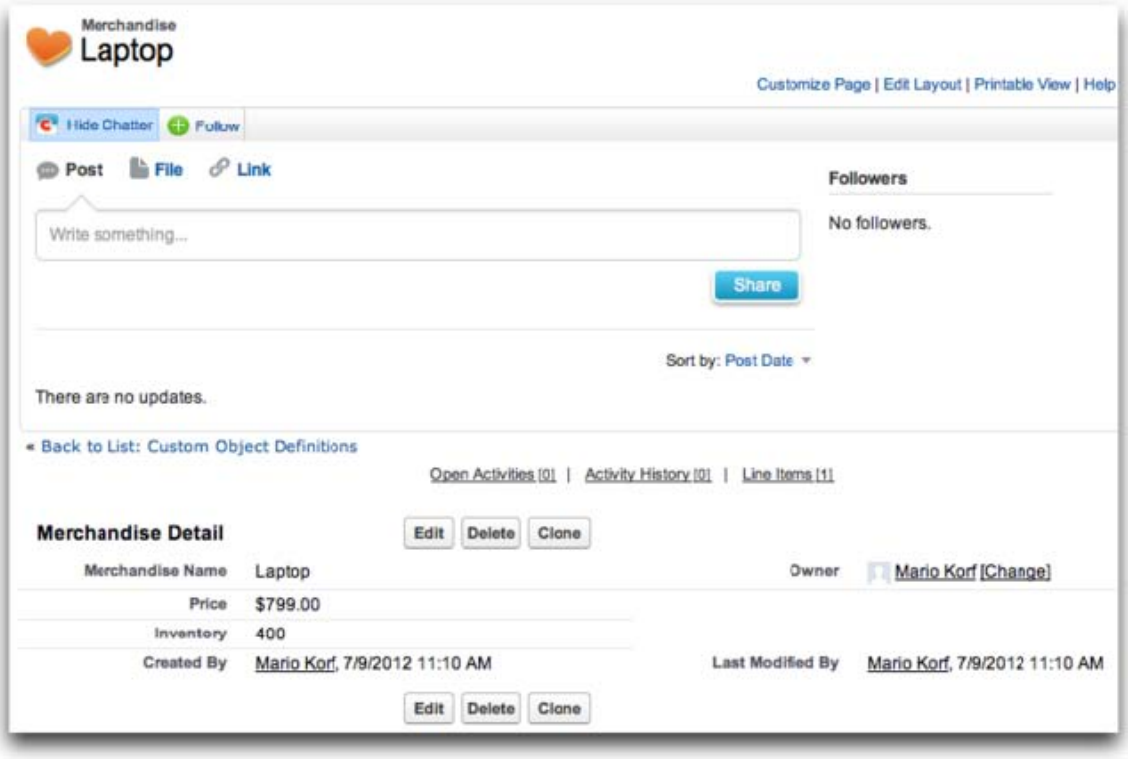
If you look at the default Invoice page layout in the Warehouse app, social collaboration isn't available. Why not? When you created the Warehouse app, the app wizard automatically enabled feed tracking on the original object—in this case, Merchandise. However, for new custom objects, the platform doesn't enable feed tracking by default. But it's easy to enable this functionality yourself in just a minute or two.

Once you've enabled feed tracking, you can also receive notifications on your mobile device, so that you'll know when someone comments on your post or otherwise interacts with you. At the end of this tutorial you enable push notifications, which will send alerts to your mobile device, even when you're not using the Salesforce1 downloadable app.

Step 1: Examine the Merchandise Page Layout

Take a look at how Merchandise already has a feed.

1. Click the Merchandise tab.
2. Click into any piece of merchandise and review the Merchandise page layout. Notice the top half of the page is dedicated to social collaboration. You can follow a piece of merchandise, attach files, and post useful links. You want that functionality for invoices too.

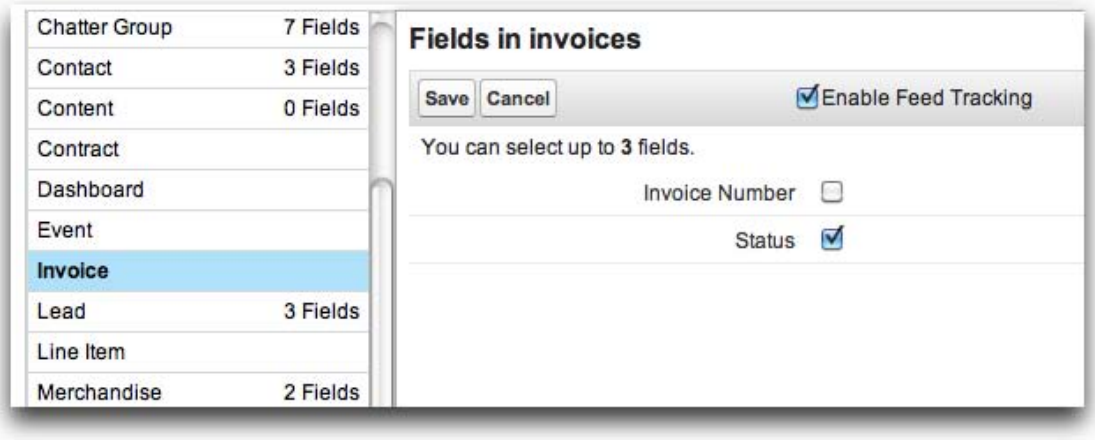


Step 2: Enable Collaboration on Invoices

To enabled feed tracking:

1. From Setup, enter *feed* in the Quick Find box.
2. Click **Feed Tracking**.

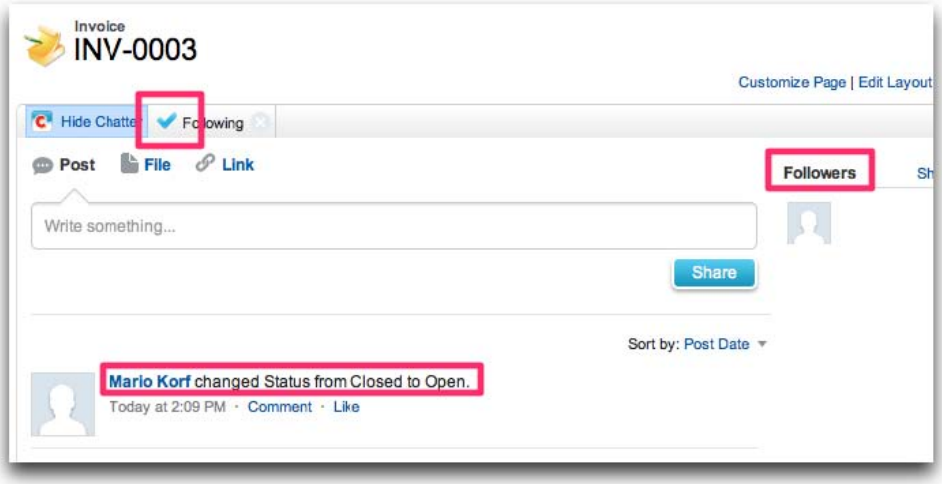
3. Notice two fields are being tracked for Merchandise. Take a look at your Invoice object, and notice no fields are being tracked.
4. To enable feed tracking for Invoice, click **Invoice**, select `Enable Feed Tracking`, select `Status`, and click **Save**.



Step 3: Try Out the App

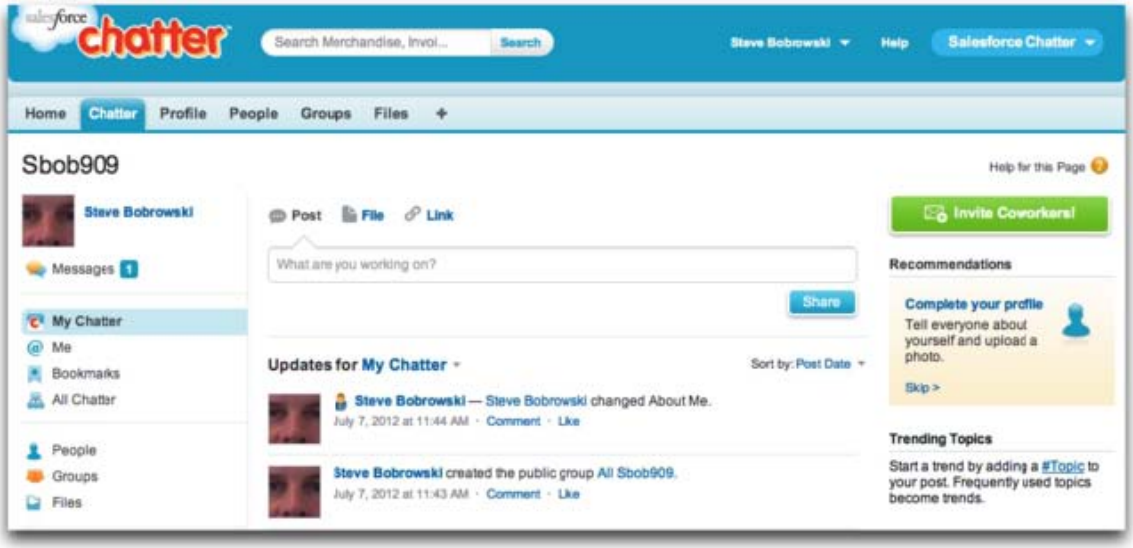
Now that you've finished modifying the Invoice page layout, have a look around.

1. Click the Invoices tab, click into the detail page for an Invoice, and notice that the Chatter feed for an Invoice is now available.
2. You can collaborate on this invoice by clicking **Follow**. Now if you update an invoice (to change it from Closed to Open for example), anything that happens to the invoice status will automatically appear in your Chatter feed, and the feed of anyone else who follows this invoice.



Tell Me More....

This tutorial only touches on Chatter, focusing on the feeds for a custom object. From the app menu in the upper right, select Salesforce Chatter to see a feed-centric view of data in your organization. The Chatter app lets you securely collaborate with other users in your organization—kind of like a private, secure Facebook just for you and your coworkers.



Step 4: Enable Notifications for Mobile

Once you've enabled a feed, you will see those updates in Chatter, but you can also receive updates on your mobile device, even when your app isn't running! To receive these updates, you need to enable notifications.

1. From Setup, enter *Salesforce1 Notifications* in the Quick Find box, then select **Salesforce1 Notifications**
2. Select the notifications you want your Salesforce1 users to receive.
3. If you're authorized to do so for your company, select Include full content in push notifications.

4. Click **Save**. If you checked the box to include full content in push notifications, a pop-up appears displaying terms and conditions. Click **OK** or **Cancel**. By enabling this option, you're agreeing to the terms and conditions on behalf of your company. For details, see Salesforce1 Mobile App Notifications in the Salesforce help.

Now when someone mentions you in a post or comments on a post you created, you'll get a notification on your device, even when your Salesforce1 downloadable app isn't running! You can't see any notifications yet, because you need to create another user to make some updates. You'll do that in a later lesson.

ADD APP LOGIC WITH CLICKS, NOT CODE

Level: Beginner; **Duration:** 30–40 minutes

At this point you already know how to create a basic app and do things like create and relate objects, and customize the standard user interface. In this set of tutorials, you'll go a step further by learning advanced point-and-click development to further enhance the underlying database and improve the UI.

Automate a Field Update Using Workflow

Level: Beginner; **Duration:** 10–15 minutes

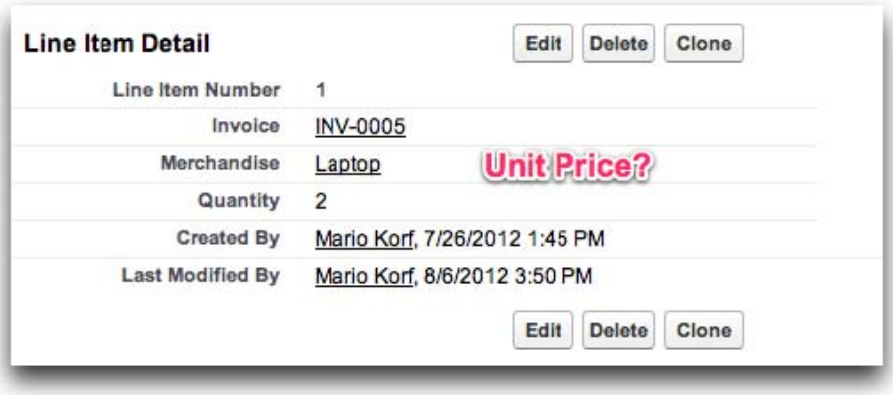
Your company can operate more efficiently with standardized internal procedures and automated business processes. In Salesforce, you can use workflow rules to automate your procedures and processes. Workflow rules can trigger actions (such as email alerts, tasks, field updates, and outbound messages) based on time triggers, criteria, and formulas.

Automatically populating a field with a default value is a common business rule. Recall that you did something similar already using a lookup field on two related objects. A Line Item can "look up" merchandise and give the user a choice of which item they want. But what if, rather than having a user choose, populating the field was done automatically? That's when you need a workflow rule, so that depending on different conditions, Salesforce can automatically populate a field with the appropriate value, and without user intervention.

Step 1: Examine the Line Item Detail Page

To get started, quickly review the Invoice and Line Item objects from earlier tutorials.

1. Select the Warehouse app from the app picker, then click the Invoices tab.
2. Open any invoice, and then open the detail page for a line item. Notice there's no price field for the line item.

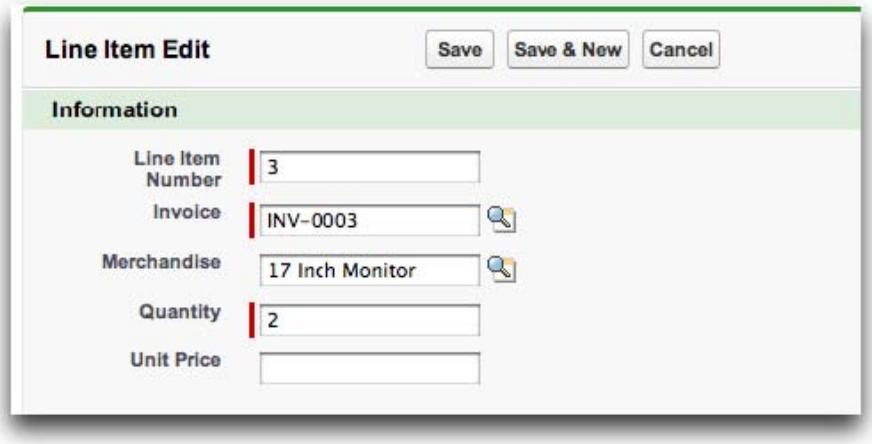


In this tutorial, you create a new field for the Line Item object called `Unit Price`. You don't want users creating their own price, and since the price is already stored in the Merchandise object, you can populate this field automatically using a neat feature called a *workflow rule*.

Step 2: Create a Unit Price Field

The steps for creating the new `Unit Price` field are essentially the same as when you created the `Price` field on the Merchandise object except this time name the field *`Unit Price`*.

1. From the Line Item tab or record, click the Quick Access menu (the tab that pops out from the right side of the window), hover over View Fields and click **New**. (If you aren't on the Line Item object already, in Setup, enter *`Objects`* in the `Quick Find` box, then select **Objects**. Then click **Line Item**, and in the Custom Fields and Relationships section, click **New**.)
2. For the data type, select `Currency` and then click **Next**.
3. Fill in the custom field details as follows.
   - `Field Label:` *`Unit Price`*
   - `Length:` *`16`*
   - `Decimal Places:` *`2`*
4. Leave the defaults for the remaining fields by clicking **Next** on subsequent screens until you can **Save**.
5. Now go back to an existing Invoice and add a new Line Item. Notice there's a new field for `Unit Price`, but you have to populate that field manually. You want this field to populate automatically, so click **Cancel**, and add this new functionality.



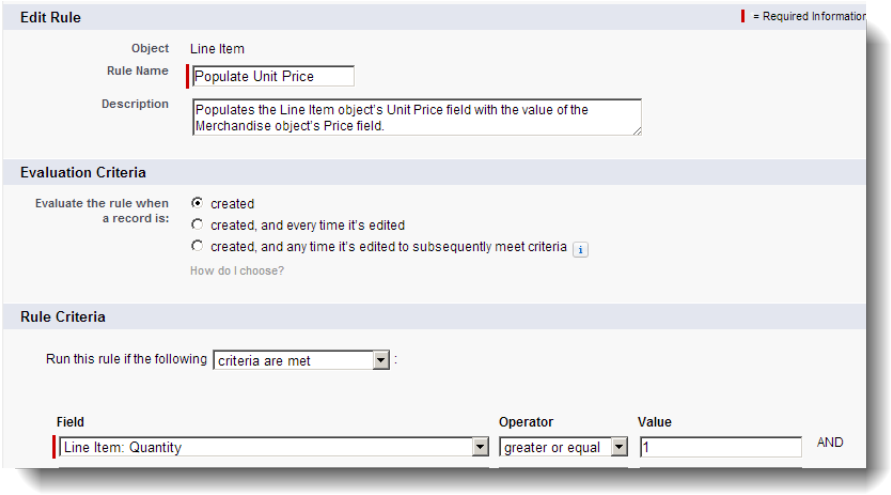
Step 3: Automatically Populate the Unit Price Field

To automatically populate the new `Unit Price` field, create a workflow rule.

1. From Setup, enter *`Workflow Rules`* in the `Quick Find` box, then select **Workflow Rules**.
2. Optionally, read the brief introduction, click **Continue**, and then click **New Rule**.
3. Select the Line Item object, and click **Next**.
4. For the rule name, enter *`Populate Unit Price`*, and for the description enter something like *`Populates the Line Item object's Unit Price field with the value of the Merchandise object's Price field`*.
5. For evaluation criteria, select `created`.

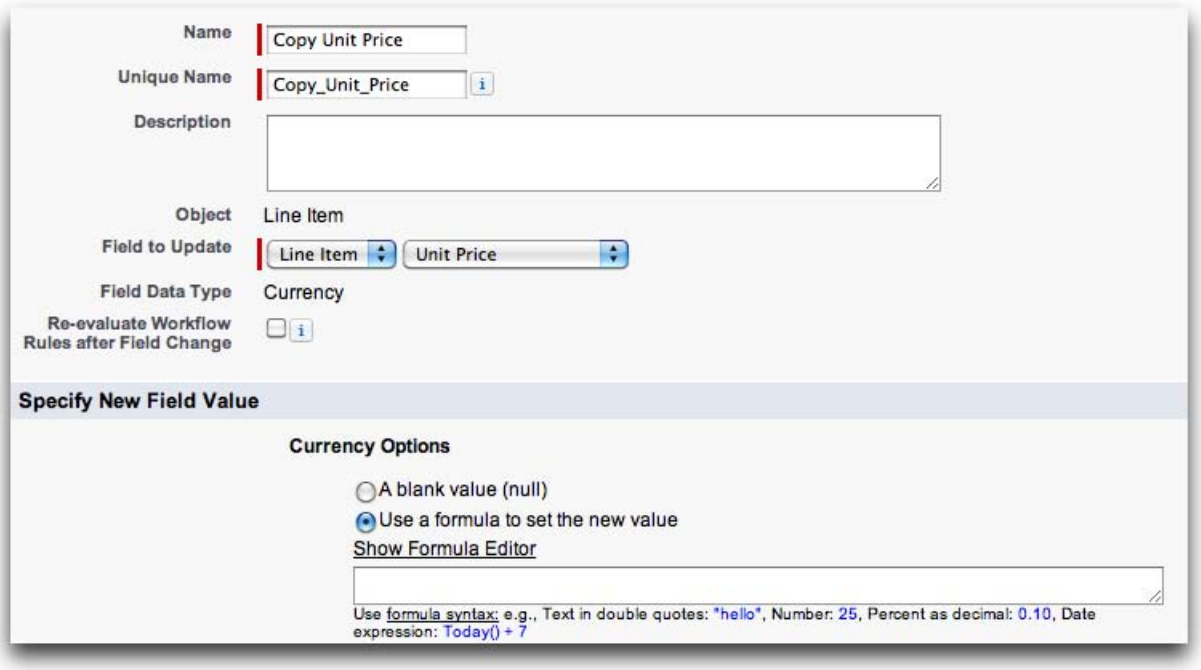
6. In the first rule criteria row, for the field select `Line Item: Quantity`, for the operator select `greater or equal`, and for the value enter `1`.
7. Click **Save & Next**.

Note: It makes sense to fire this workflow rule only for *new* line item records because you are effectively assigning a default field value when creating a new record. Later on, users might need to adjust the price of merchandise in each line item (for example, to offer discounts).

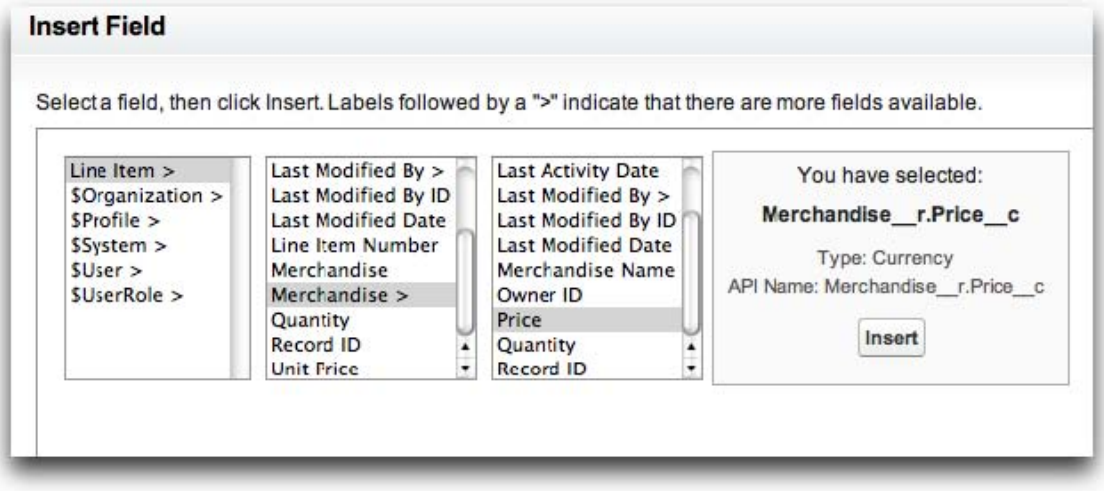


Continuing on, the next step is to assign an action to the workflow rule to update the `Unit Price` field automatically.

1. Click the drop-down list that reads Add Workflow Action and choose **New Field Update**.
2. In the `Name` field, enter `Copy Unit Price`.
3. In the Field to Update list, choose Line Item and then Unit Price.
4. Select the option to use a formula to set the new value. Before continuing, confirm that your screen matches the following.



5. Click **Show Formula Editor**, and then click **Insert Field**.
6. In the first column choose **Line Item >**, in the second column choose **Merchandise >**, and in the third column choose **Price**.
7. Confirm that your screen matches the following, and then click **Insert**.



8. Click **Save**, and then click **Done** to return to the detail page of the new workflow rule.

Tell Me More...

In the formula, notice some new syntax, namely "`Merchandise__r`". You've seen `__c` used already, so what's with the `__r`? That's the platform's object notation for a field that's related to another object. You can use related fields to traverse object relationships and access related fields. In this case, the formula uses the relationship between the Line Item record and Merchandise object to get the corresponding Merchandise record's value for `Price`.

Step 4: Update Total Inventory When an Order is Placed

The inventory of merchandise should be automatically maintained as orders are placed. When you create a new invoice ("Open" status), every new line item needs to decrease the total inventory by the number of units sold. Similarly, updates to an existing line item need to update the total inventory by the difference in units sold.

There are a few different ways you can make this update. You could do this in Apex code, or by creating a Flow, or by creating another workflow rule. For simplicity, you'll stick with workflow for now, but there is one minor problem to fix first, which is that the workflow field update won't work with a lookup relationship. So the first step is to change the lookup to a master-detail. Fortunately, the platform makes such changes very easy.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**, and click **Line Item**.
2. Scroll down to Custom Fields and Relationships, and next to Merchandise click **Edit**.
3. Click **Change Field Type**, and then select Master-Detail Relationship.
4. Click **Next**, and then **Save**.

Now you can create the workflow rules.

1. From Setup, enter *Workflow Rules* in the Quick Find box, then select **Workflow Rules**
2. On the All Workflow Rules page, click **New Rule**.
3. Select Line Item as the object, and click **Next**.
4. In the Rule Name field, enter *Line Item Updated*.
5. For Evaluate the rule when a record is: select created, and every time it's edited.
6. In the Rule Criteria field, leave *criteria are met* selected.
7. In the Field drop-down list, select *Invoice: Status*. In Operator, select *equals*. For Value, click the lookup icon and choose Open, and click **Insert Selected**.
8. Click **Save & Next**.
9. Click **Add Workflow Action** and choose **New Field Update**. The New Field Update wizard opens.
10. In the Name field, enter *Update Stock Inventory*.
11. In the first Field to Update drop-down list, select Merchandise. In the second, select Quantity.
12. Select Use a formula to set the new value.
13. Click **Show Formula Editor**.
14. Click **Insert Field** and choose **Line Item** > **Merchandise** > **Quantity**. Click **Insert** to add the field to the editor.
15. Click **Insert Operator** and choose **– Subtract**.
16. Click **Insert Field** and choose **Line Item** > **Quantity**. Click **Insert** to add the field to the editor.

    The completed formula should be Merchandise__r.Quantity__c - Quantity__c.
17. Click **Check Syntax**, and make corrections if necessary.
18. Click **Save** to close the New Field Update wizard and return to Step 3 of the Workflow wizard.
19. Click **Done**.

Step 5: Activate the Workflow Rule

This is a tiny step, but it's an important one. By default, workflow rules are not active.

1. In Setup, enter *Workflow Rules* in the Quick Find box, then select **Workflow Rules** to get to the All Workflow Rules page.
2. Next to Line Item Updated and Populate Unit Price, you'll see an **Activate** link. Click the link next to each workflow rule.

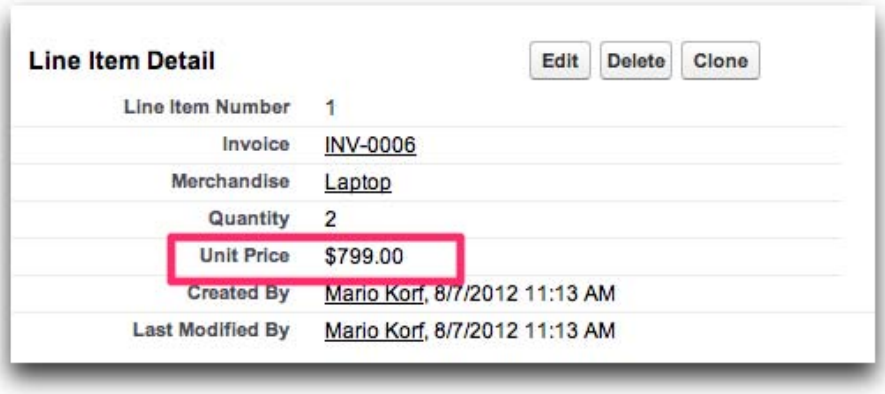
Tell Me More...

Workflow rules are not activated by default because you might turn off workflow rules when running bulk processes. For example, you might want to update a whole bunch of records at the same time, and firing the workflow rule each time wouldn't invalidate your processes. Workflow rules can also do things like send email updates, and you might not want to send thousands of emails when you're doing a simple price change.

Step 6: Try Out the App

Now try out the revised app and see how the new workflow rule implements your business logic.

1. Click the Invoices tab and either create a new Invoice or edit an existing Invoice.
2. Add a **New Line Item** and after you've chosen the Merchandise, click **Save**.
3. Click back into the detail page for the new Line Item and notice how the first workflow rule you created automatically populated the Unit Price field by looking up the Price of the Merchandise that you selected. Sweet.



Add a Formula Field

Level: Beginner; **Duration:** 5–10 minutes

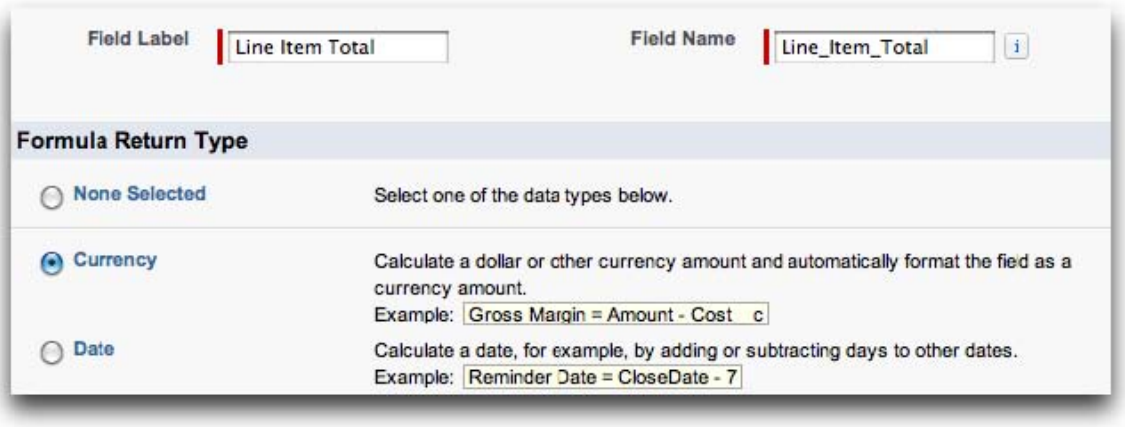
Another thing that's missing from the Line Item object is a Line Item Total field that displays the product of each Line Item's Quantity and Unit Price. In this tutorial, you implement this common business logic by creating a new formula field in the Line Item object, again, without writing any code.

Step 1: Calculate a Value for Each Line Item

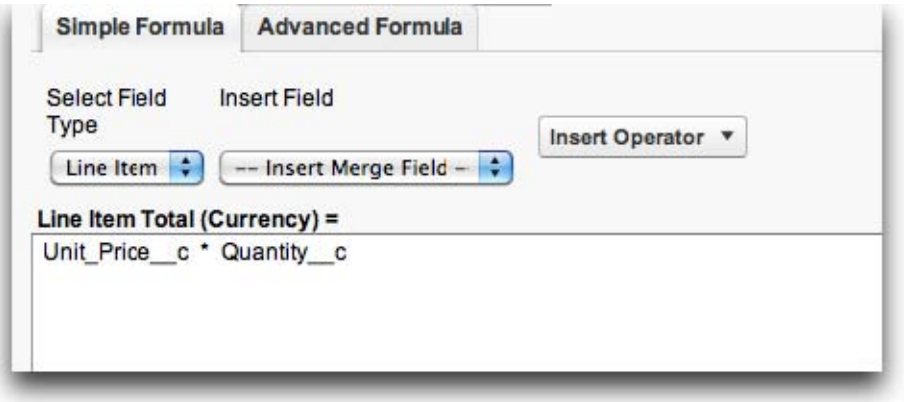
In the first step of this tutorial, you'll add a new calculated field called Line Item Total to the line item. This field multiplies the number of items with the price and acts as a total for each line item.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**.
2. Click the **Line Item** object. Then, in the Custom Fields & Relationships related list, click **New**.

3. Choose `Formula`, and click **Next**.
4. For `Field Label`, enter *`Line Item Total`*.
5. For Formula Return Type, choose `Currency` and click **Next**.



6. Click the Insert Merge Field drop-down list, and choose **Unit Price**. You should now see `Unit_Price__c` in the text box.
7. Click the Insert Operator drop-down list and choose **Multiply**.
8. In the **Insert Merge Field** drop-down list, select **Quantity**. You should now see `Unit_Price__c * Quantity__c` in the text box.



9. Click **Next**, **Next**, and then **Save**.

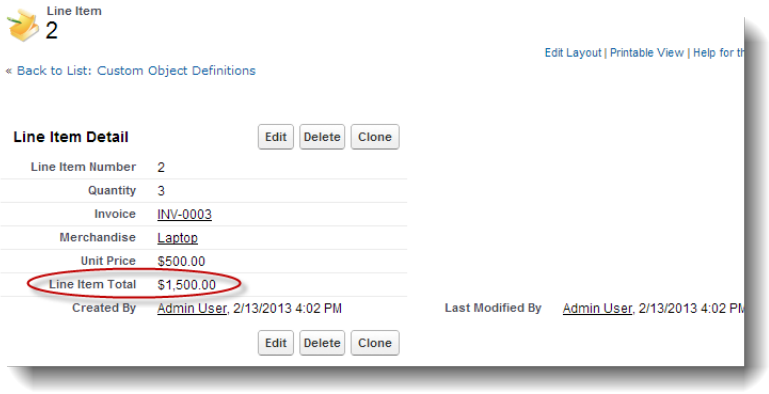
Tell Me More....

The Formula field type is great for automatically deriving field values from other values, as you have done here. The formula you entered was quite straightforward: a simple multiplication of two field values on the same record. There's also an Advanced Formula tab, which allows you to do much more with these formulas.

Step 2: Try Out the App

To see the new Line Item Total formula field in action, you'll need to create a new line item.

1. Click the Invoices tab and then click an existing invoice.
2. Add a new line item, select a piece of merchandise, and enter a quantity.
3. Save the line item and you can see the formula field in action.



Add a Roll-Up Summary Field

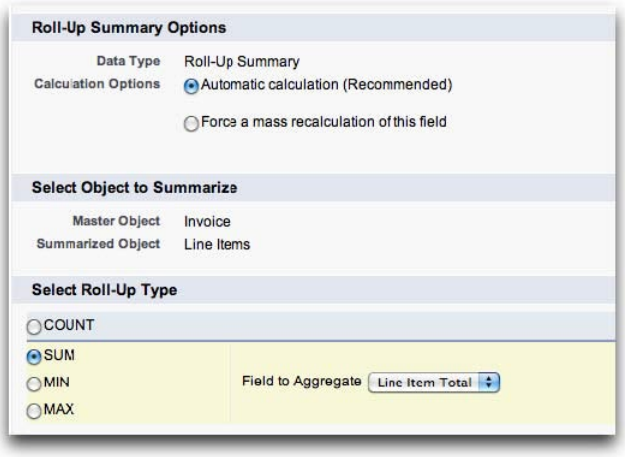
Level: Beginner; **Duration:** 5–10 minutes

Another thing that's missing from the Invoice is a field that aggregates all of the line items into one big invoice total. This is easy to do if the objects are in a master-detail relationship, because you can use a *roll-up summary field*.

Step 1: Calculate a Total With a Roll-Up Summary Field

Now that you have the total for each line item, it makes sense to add them all to get the invoice total. Because the line items have a master-detail relationship with the invoice, you can use a roll-up summary field to calculate this value. Roll-up summary is a special type of field that lets you aggregate information about related detail (child) objects. In this case, you want to sum the value of each line item.

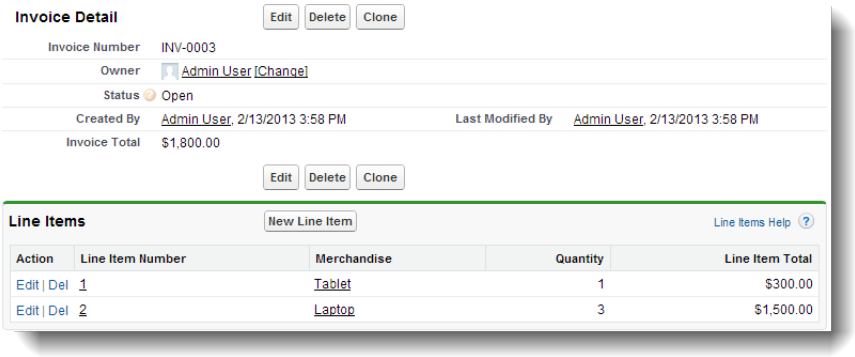
1. Navigate back to the Invoice custom object page from Setup by entering `Objects` in the `Quick Find` box, then selecting **Objects** and then clicking **Invoice**.
2. In the Custom Fields & Relationships related list click **New**.
3. Choose `Roll-Up Summary` as the data type, and click **Next**.
4. For the `Field Label` field, enter `Invoice Total`, and click **Next**.
5. In the Summarized Object list choose Line Items.
6. For Roll Up Type, select `Sum`.
7. In the Field to Aggregate list choose Line Item Total.
8. Verify that your screen looks like this. Then click **Next**, **Next** and **Save**.



Step 2: Try Out the App

To see the new Invoice Total formula field in action, you only need to examine an invoice.

1. Click the Invoices tab and then click an existing invoice.
2. Notice the new `Invoice Total` field that "rolls up" all the values from the detail object's Line Item Totals.
3. To get the Line Item Total field to appear on the detail page, you'll have to edit the page layout. (If you haven't done that yet, see Modify a Page Layout on page 32). When you do, it should look like the following image.



Enforce a Business Rule

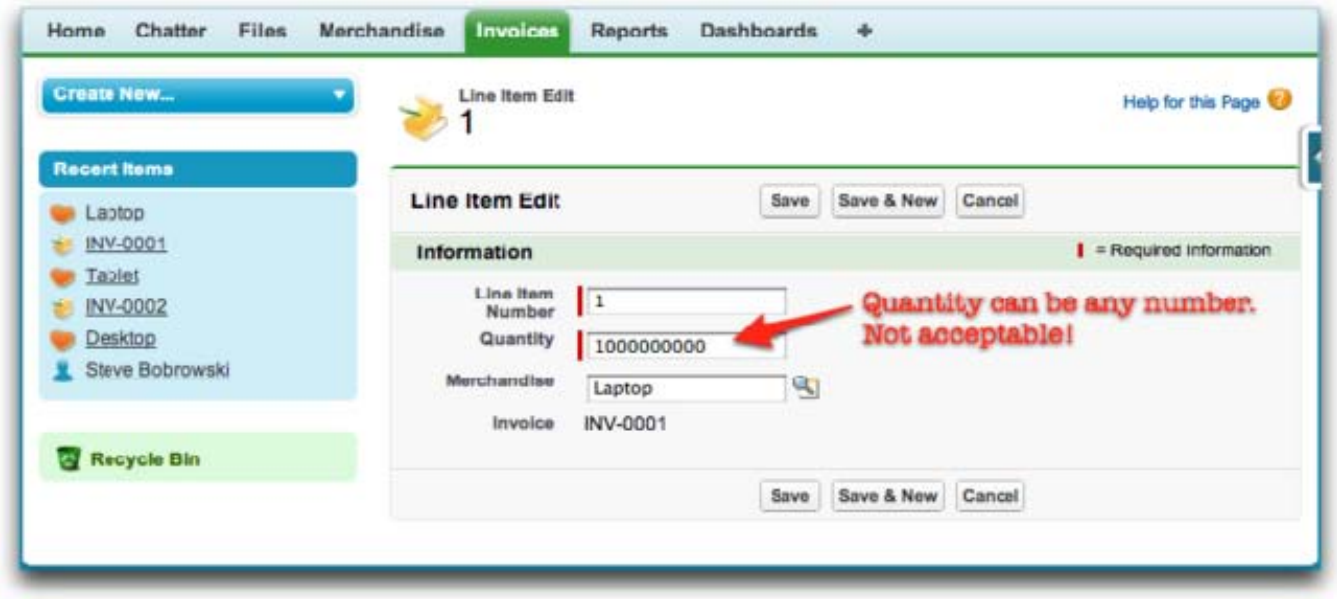
Level: Beginner; **Duration:** 5–10 minutes

Typically, every business app enforces rules that prevent bad data from getting into the system. Without such rules, things can get really messy, really fast because users might not adhere to these rules on their own. In this tutorial, you learn how to enforce a basic business rule for the Warehouse app—you can't order zero or negative items. To do this, you create and test a validation rule, all in just a couple of minutes without any coding.

Step 1: Understand the Business Rule

Before you begin, make sure you have a clear understanding of this particular business rule.

1. Select the Warehouse app.
2. Click the Invoices tab, select an invoice, and look at a specific line item.
3. Play around with the quantity field for the line item. Notice that a value is required, but that you can set the value to any number: 0, -10, 3.14159. You don't want users entering bad data (such as negative numbers), so this situation isn't acceptable.



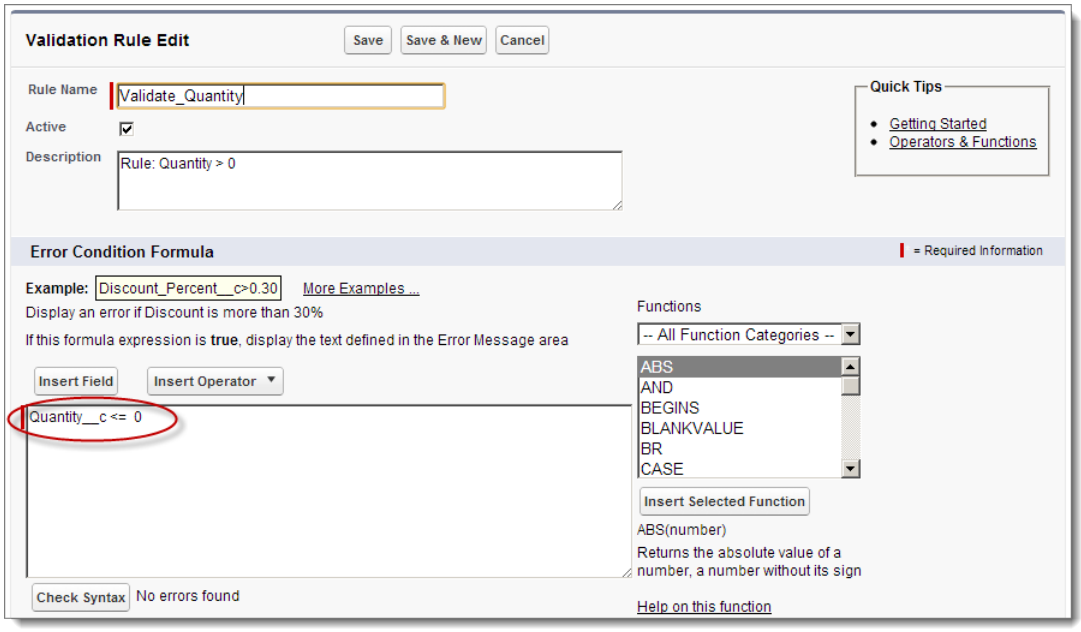
Step 2: Create a Validation Rule

Enforcing basic business rules is easy and doesn't require any coding.

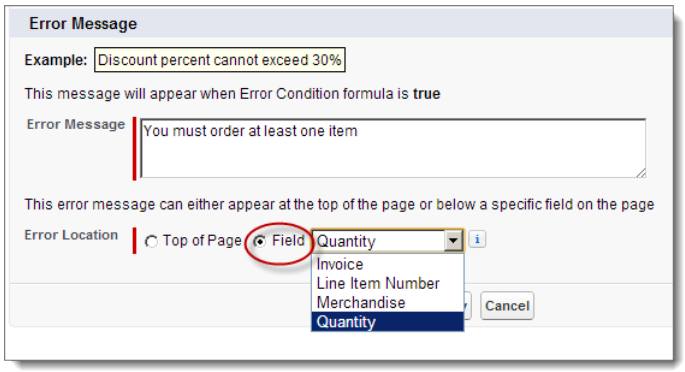
1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**, click **Line Item**, scroll down to the Validation Rules related list, and click **New**.
2. For Rule Name type *Validate_Quantity*.
3. Optionally fill out the Description field. It's a good practice to document business logic so that other developers can easily understand the purpose of the rule. Use the documentation links if you need extra help on this page.
4. In the Error Condition Formula area, you build a validation rule's error condition formula to identify when the error condition evaluates to TRUE.
   a. Click **Insert Field** to open the Insert Field popup window.
   b. Select Line Item > in the first column and Quantity in the second column.
   c. Click **Insert**.
   d. Type the less-than-or-equal-to symbol (<=) and the numeral 0, so the formula looks like:

   ```
   Quantity__c <= 0
   ```

5. Click **Check Syntax** to make sure there are no errors. If you do find errors, fix them before proceeding.



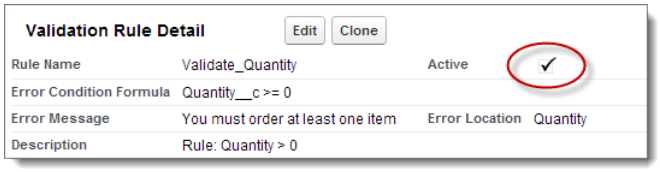
6. In the `Error Message` field, type *`You must order at least one item`*.
7. For the `Error Location`, select `Field`, then choose Quantity from the drop-down list.



8. Click **Save**.

Tell Me More....

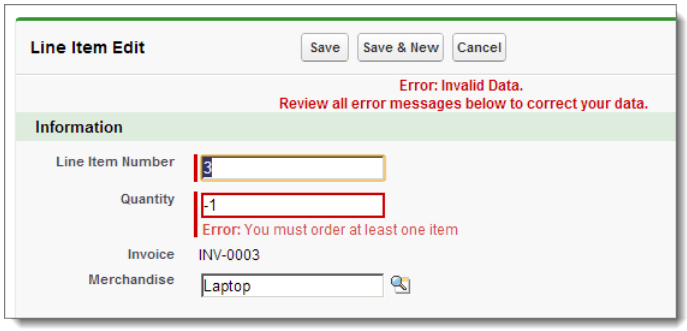
Take a quick look at the Validation Rule Detail page. Notice that the new validation rule is "Active" meaning that the platform is currently enforcing the rule. Validation rules, unlike workflow rules, default to active. In certain situations, you might want to deactivate the rule temporarily (for example, before loading a bunch of data). This is easy to do by simply deselecting the `Active` box (but don't do this now).



Step 3: Try Out the App

Now that the rule is in place and active, it's time to give it a try.

1. Click the Invoices tab and select an existing invoice.
2. Click **New Line Item**
3. Enter a line item number and a quantity of *-1*.
4. Once you choose a merchandise item and click **Save**, you'll see the error message that you set up for the rule.



5. Fix the error by entering a valid quantity and then **Save**.

Tell Me More....

- If you didn't see the error message, check the validation formula again. You need to make the rule fire when the condition evaluates to TRUE.
- The formula in this tutorial is rather simple, but don't let that fool you. The platform's formula syntax empowers you to enforce a wide range of business rules that not only includes one object, but pulls in other related objects as well.

Step 4: Modify the Validation Rule

Modify the existing validation rule to check how many items are in stock.

1. From Setup, enter *Objects* in the `Quick Find` box, select **Objects**, select **Line Item**, scroll down to the Validation Rules related list, and edit the **Validate_Quantity** rule.
2. Edit the `Description` field to explain that it won't allow users to order more items than are in stock.
3. In the Error Condition Formula area, start by putting some parentheses around the first rule, insert the logical `OR` operator, and then add another set of parentheses so that the error condition looks like this:

```
(Quantity__c <= 0) || ()
```

4. Click between the second set of parentheses, then click **Insert Field** to open the Insert Field popup window.
5. Leave `Line Item >` selected in the first column, select `Quantity` in the second column, and then click **Insert**.
6. Type or insert the greater-than symbol (>).
7. Click **Insert Field** and select `Line Item >` in the first column, `Merchandise >` in the second column, and `Quantity` in the third column.

8. Click **Insert** and verify the code looks like the following:

```
(Quantity__c <= 0) || (Quantity__c > Merchandise__r.Quantity__c)
```

9. Click **Check Syntax** to make sure there are no errors.
10. Finally, edit the `Error Message` field to add *`You can't order more items than are in stock`*, and then **Save**.

Tell Me More....

Take a look at the formula you created.

- `Mechandise__r`—Because the Merchandise object is related to the Line Item object, the platform lets you navigate from a line item record to a merchandise record; that's what the `Mechandise__r` is doing.
- `Quantity__c`—This is the field you created to track the total amount of stock on a merchandise record.
- `Merchandise__r.Quantity__c`—This tells the system to retrieve the value of `Quantity` field on the related merchandise record.
- `Quantity__c`—This refers to the `Quantity` field on the current (line item) record.

Putting it all together, the formula checks that the total inventory on the related merchandise record is less than the number of units being sold. As indicated on the Error Condition Formula page, you need to provide a formula that is true if an error should be displayed, and this is just what you want: it will only be true when the total inventory is less than the units sold.

Step 5: Try Out the New Rule

Now that the modified rule is in place, test it.

1. Click the Invoices tab and select an existing invoice.
2. Create a **New Line Item** and type a quantity of *`6000`*.
3. Choose a merchandise item, and click **Save**. You see the error message that you set up for the rule.
4. Fix the error by entering a valid quantity, and then click **Save**.

Tell Me More....

- Validation rules and formulas combine to create really powerful business logic, with very little development effort.
- For a list of sample validation rules, make sure to read "Examples of Validation Rules" in the Salesforce Help: `https://help.salesforce.com/HTViewHelpDoc?id=fields_useful_field_validation_formulas.htm`.

Create an Approval Process

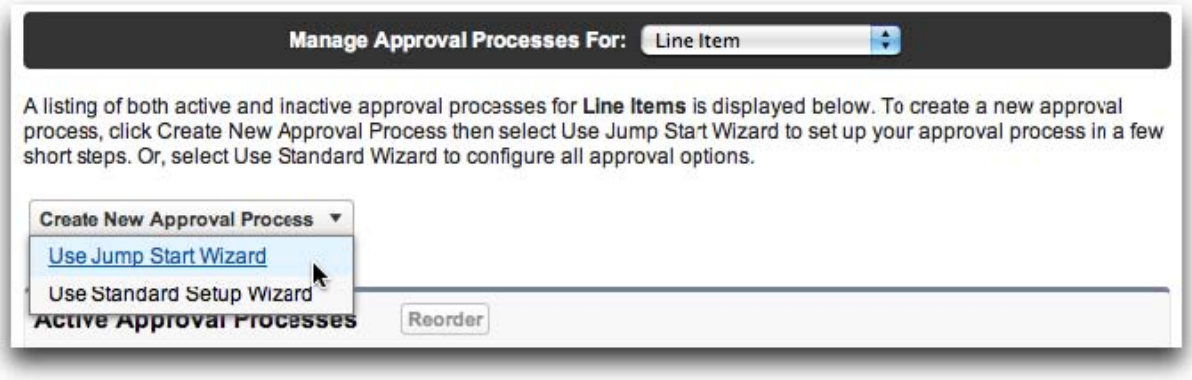
Level: Beginner; **Duration:** 10–15 minutes

An approval process specifies the steps necessary for a record to be approved and who must approve it at each step. A step can apply to all records included in the process or just records that have certain attributes. An approval process also specifies the actions to take when a record is first submitted for approval and that record is approved, rejected, or recalled.

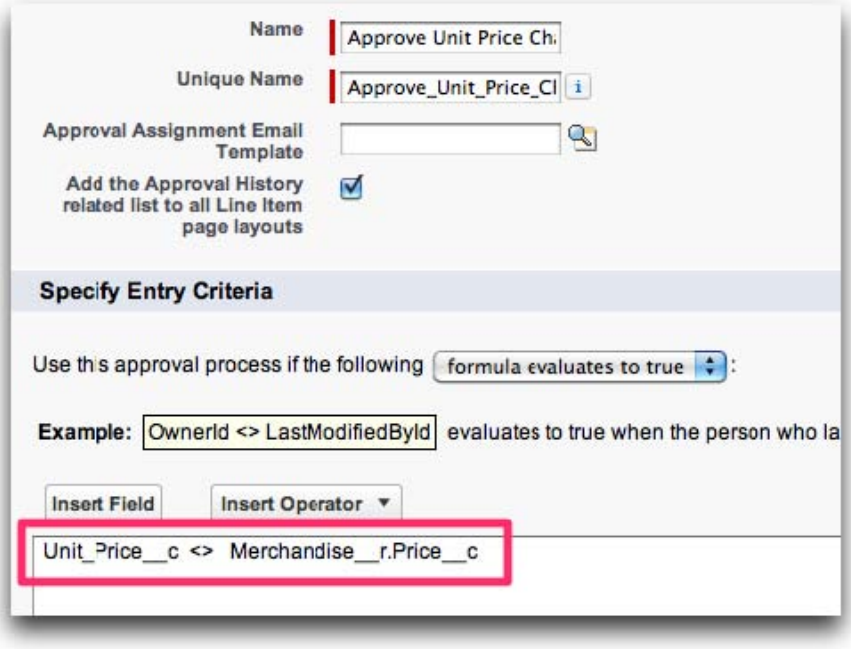
Step 1: Create an Approval Process

To create an approval process, you start in Setup.

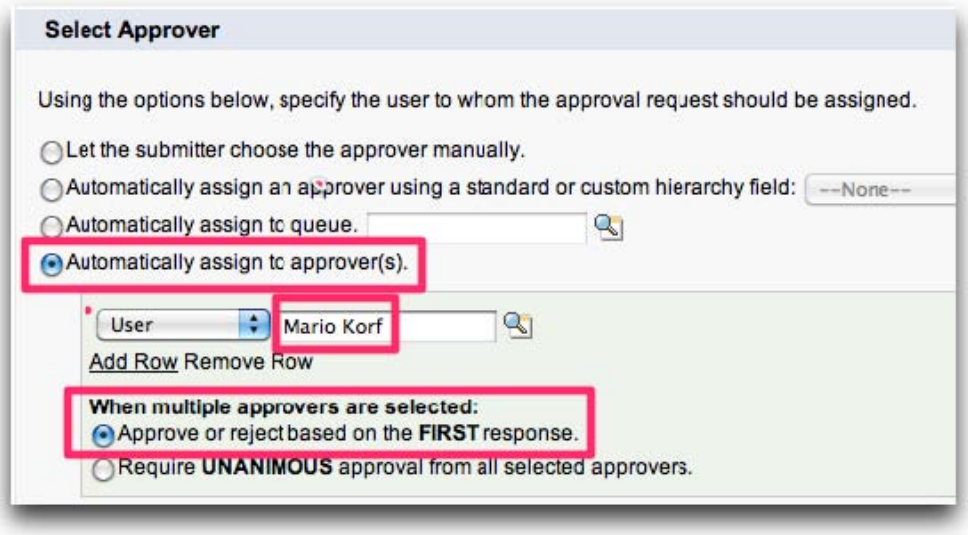
1. From Setup, enter *Approval Processes* in the Quick Find box, then select **Approval Processes**.
2. In the Manage Approval Process For drop-down list, choose Line Item.
3. Click **Create New Approval Process** and then **Use Jump Start Wizard**.



4. In the Name field, enter *Approve Unit Price Change*
5. Click the drop-down list next to Use this approval process if the following, and choose *Formula evaluates to true*.
6. In the formula field, click **Insert Field**, select Line Item > and then select Unit Price. Click **Insert**.
7. Click **Insert Operator** and select **<> Not Equal**.
8. Click **Insert Field**, select Line Item >, select Merchandise > and then Price. Click **Insert**. Before moving on, make sure your screen looks like:



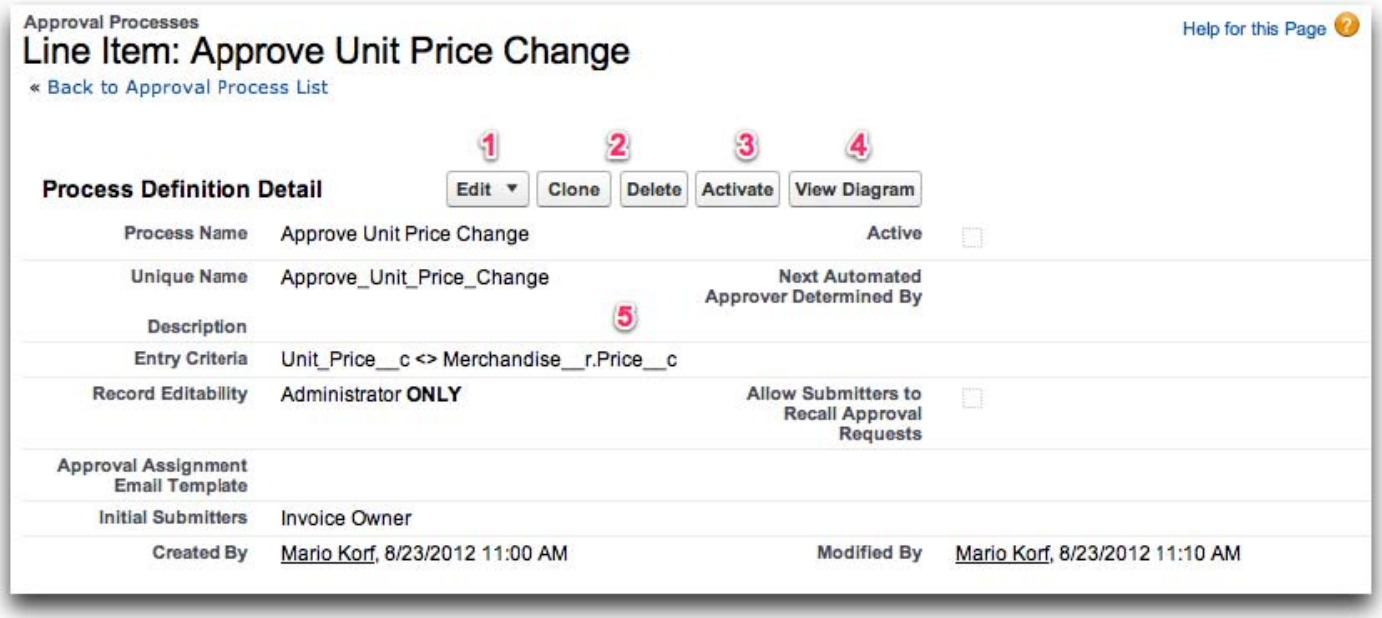
9. Now you need to assign the approval to someone. For large companies where multiple people could have the ability to grant approval, you might assign this to a queue. In DE orgs there are only two users, so click the option for `Automatically assign to approvers` and choose Admin User. (If you've edited your profile, this will be your name, note that you may need to click the lookup icon if you don't's see your name listed.)
10. Make sure your screen looks like this and then **Save** your work.



11. Click **OK** in the pop-up, and then click **View Approval Process Detail Page**.

Step 2: Examine the Approval Process Detail Page

The detail page of the approval process has a lot going on, and it's worth a minute to explore the user interface.



1. Edit every step of an approval process.
2. Clone or delete an approval process.
3. Activate and deactivate an approval process.
4. View an approval process diagram as a flow chart.
5. View general details of the approval process.

In addition, you can add new steps and actions (email alerts, field updates, and outbound messages) wherever you want.

Step 3: Modify Approval Process Actions

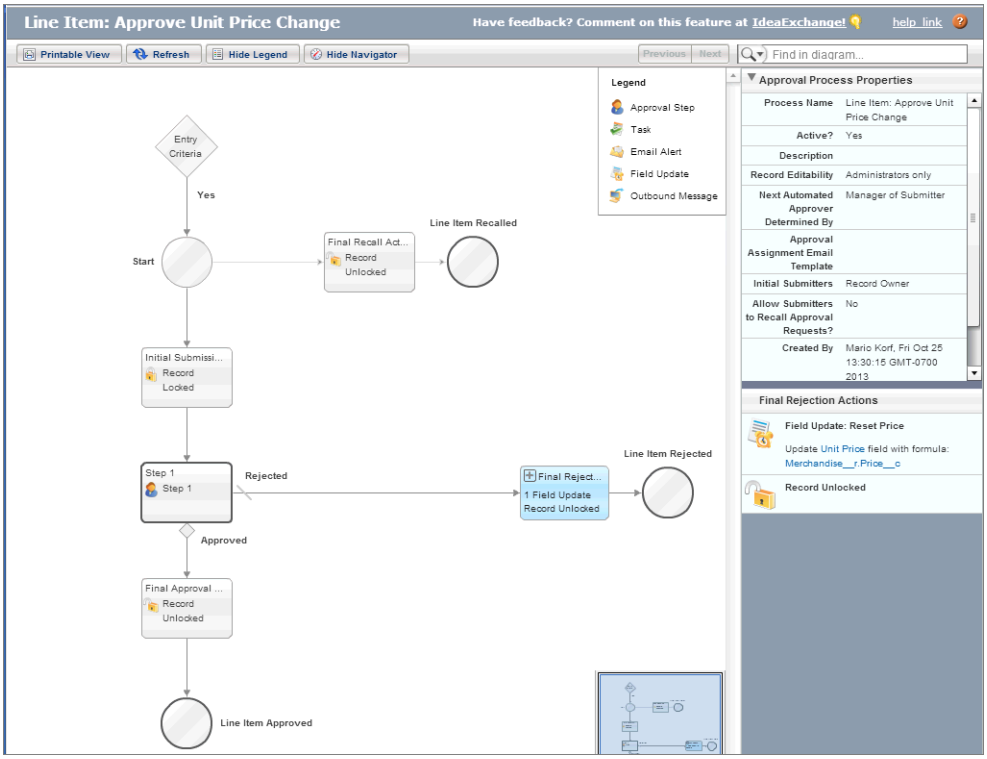
In this step you modify the approval process so that if the price change is rejected, the price reverts back.

1. In the Final Approval Actions section, click **Edit** next to Record Lock.
2. Choose **Unlock the record for editing**, and then click **Save**.
3. In the Final Rejection Actions section, click **Add New** and choose **Field Update**.
4. In the `Name` field, enter *`Reset Price`*.
5. In Field to Update, choose Unit Price.
6. Select `Use a formula to set the new value`.
7. Click **Show Formula Editor**.
8. Use the Formula Editor to select `Line Item >`, then `Merchandise >`, then `Price`.
9. Click **Insert**, and then click **Save**.

Step 4: Activate the Approval Process

Just like with workflow rules, you must activate an approval process before you can use it. This might seem like an unnecessary step until you think about situations where you might not want an approval process to run. For example, let's say you want to run a special promotion and decrease the price of a certain laptop in all open invoices. This would fire off the approval process for every open invoice, creating a bottleneck to getting orders out the door. In a case like this, you'd want to deactivate the approval process before running the batch update. When you're finished, you'd activate the approval process again.

1. Click **Activate** and then click **OK** in the pop-up.
2. While you're on the detail page, click **View Diagram** to get a visual representation of your approval process. You can click any of the nodes to get more information.



Before you can see how the approval process works, you need to make sure that your users will be able to submit the relevant records for approval. Otherwise, the approval process will never start! In this step, you add the Submit for Approval button to the Line Item page layout.

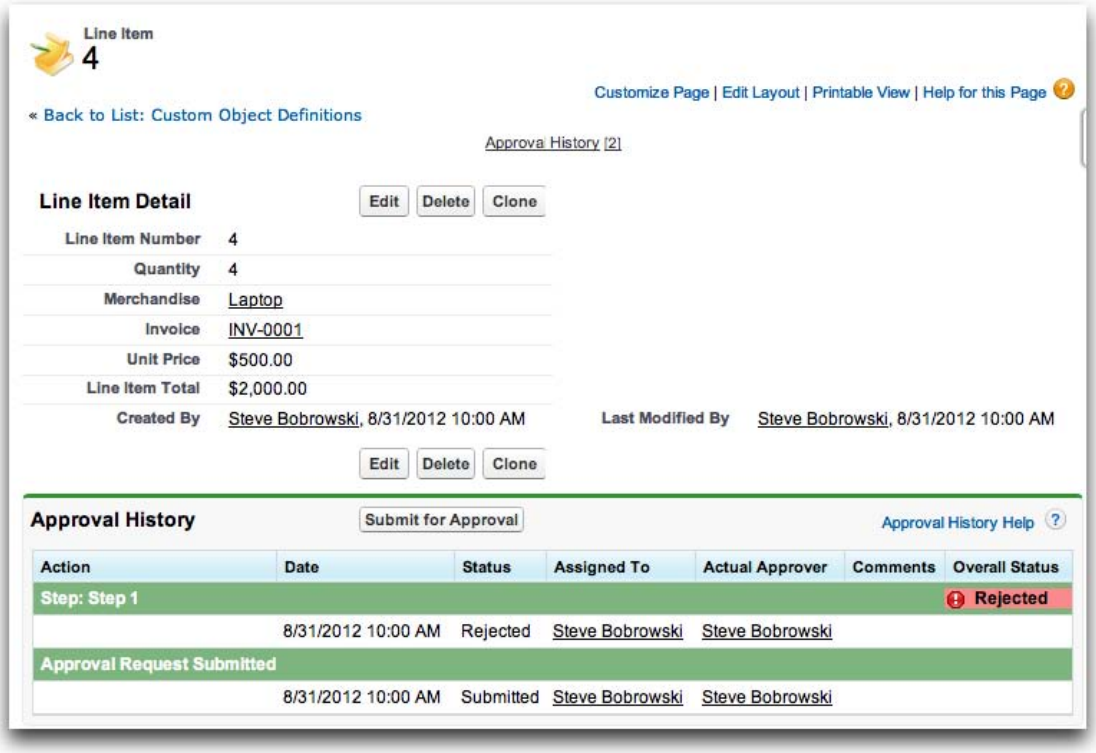
1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**, and click **Line Item**.
2. In the Page Layouts related list, click **Edit** next to Line Item Layout.
3. From the Buttons category in the palette, drag the **Submit for Approval** button to the Standard Buttons area.
4. Click **Save**.

Now all users assigned to this page layout will be able to submit line items for approval.

Step 5: Try Out the App

Now it's time to try out the new approval process and simulate the workflow as both the submitter and approver of a change.

1. Click the Invoices tab, and choose an existing invoice.
2. Add a new item to the invoice.
3. Click **Edit** next to the new line item, reduce the value for Unit Price, and then click **Save**.
4. Click **Submit for Approval** and **OK**.
5. Notice that the record is locked, you get a default email, and in the Approval History related list the overall status is Pending.
6. Click the link you received in your email, add a comment, and then click **Approve**. Notice the record is unlocked and the overall status is Approved.
7. Repeat steps 1-6, but this time reject the price change. Notice that Unit Price reverts to the default merchandise price.



Tell Me More....

Approval processes are automatically included on your Home tab. If you click the Home tab, you can see the items you need to approve and reject right there.

Step 6: Configure Approvals for Chatter and Salesforce1

Approval processes have built-in support for Chatter posts, which means they can also show up on your mobile device.

1. In Setup, enter "Chatter Settings" in the `Quick Find` box, then select **Chatter Settings**.
2. Click **Edit**.
3. Select `Allow Approvals`, and then click **Save**.

Approval feed items will now show up on your users' Chatter feed on the full site and in Salesforce1.

Create a Flow

Note: Visual Workflow isn't supported in Salesforce1.

Level: Beginner; **Duration:** 15–20 minutes

Visual Workflow enables you to build applications, known as *flows*, to guide users through screens for collecting and updating data. You can visually build flows using the drag-and-drop user interface of the Cloud Flow Designer. No coding required!

In this tutorial, we'll create a simple flow that does the following each time it runs:

- Prompt the user for the line item information.

- Create the line item record for the invoice.
- Reduce the quantity of merchandise in stock by the quantity ordered in the line item.

Step 1: Add Flow Variables

You can use flow variables to store data that can be used throughout the flow and referenced as values for updating record fields. In this tutorial, we'll create two flow variables.

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

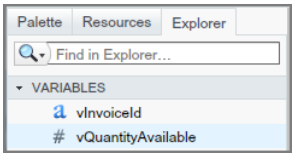
   If prompted, activate the Adobe® Flash® plug-in.
3. Create the first variable.
   a. From the Resources tab, double-click **Variable**.
   b. Configure the variable as follows.

Field	Value
Unique Name	*vQuantityAvailable*
Description	*Quantity of merchandise in stock.*
Data Type	*Number*
Scale	*0*

   c. Click **OK**.
4. Create the second variable.
   a. From the Resources tab, double-click **Variable**.
   b. Configure the variable as follows.

Field	Value
Unique Name	*vInvoiceId*
Description	*ID of the invoice to which the flow adds the new line item. A custom button on the Invoice detail page launches the flow and passes the invoice ID into this variable.*
Data Type	*Text*
Input and Output	*Input Only* This setting enables the variable to be set when the flow is launched by a custom button.

   c. Click **OK**.
5. Click **Explorer** to verify the variables are saved.



6. Save the flow.
   a. Click **Save**.
   b. For the `Name`, enter **Add Line Item from Invoice and Update Stock Quantity**.
      The `Unique Name` is automatically populated based on this entry.
   c. Click **OK**.
      Ignore any activation warnings for now.

Step 2: Add a Form Screen

A screen can use form-style fields to gather data—in this case, line item information—from the flow user.

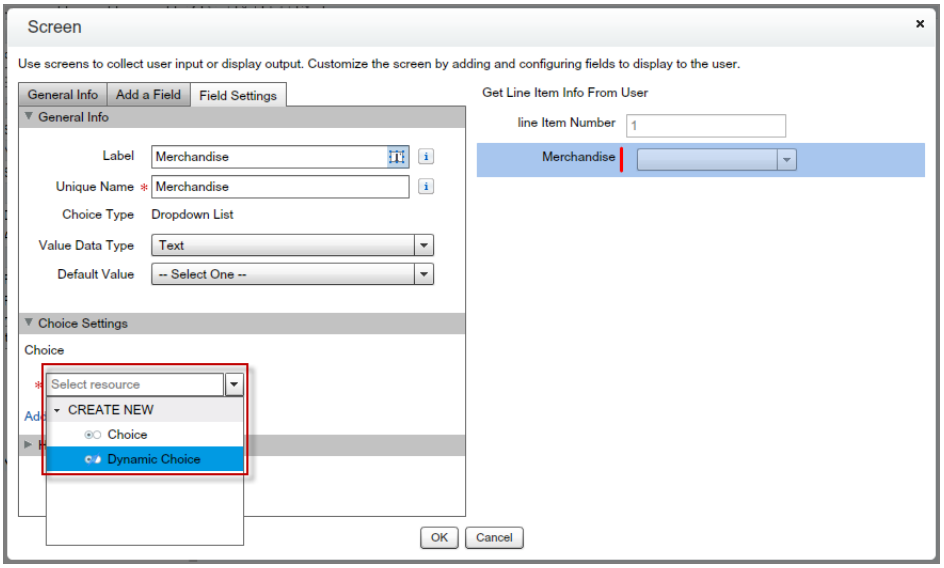
1. From the Palette tab, drag the Screen onto the canvas.
   The Screen overlay opens with the General Info tab selected.
2. For the `Name`, enter **Get Line Item Info From User**.
   The `Unique Name` is automatically populated based on this entry.
3. Add a field for the Line Item Number.
   a. From the Add a Field tab, double-click **Textbox**.
      A textbox field appears in the preview pane on the right side of the Screen overlay.
   b. Click **[Textbox]** in the preview pane.
   c. On the Field Settings tab, configure the field as follows.

Field	Value
`Label`	*`Line Item Number`* The `Unique Name` is automatically populated based on this entry.
`Default Value`	*`1`*

4. Add a field for the Merchandise.
   a. From the Add a Field tab, double-click **Dropdown List**.
      A drop-down list field appears in the preview pane.
   b. Click **[Dropdown List]** in the preview pane.
   c. On the Field Settings tab, configure the field as follows.

Field	Value
Label	*Merchandise* The Unique Name is automatically populated based on this entry.
Value Data Type	*Text*

d. In the Choice Settings section, click the drop-down arrow then **CREATE NEW** > **Dynamic Record Choice**.



The Dynamic Record Choice overlay appears.

5. Create the dynamic record choice resource, which at runtime dynamically populates the Merchandise field with choice options, each of which represents a Merchandise record in the database.

a. Enter the following values.

Field	Value
Unique Name	*dcMerchandise*
Value Data Type	*Text*
Create a choice for each	*Merchandise__c*

b. Set the filter criteria so that the dynamic record choice returns only the merchandise that have items in stock.

Field	Value
Field	*Quantity__c*
Operator	*greater than*

Field	Value
Value	*0*

c. Set the following fields so that the choices are displayed using the Name in each Merchandise record, sorted in alphabetical order. Also, we want the choice to store the ID of the user-selected Merchandise record.

Field	Value
Choice Label	*Name*
Choice Stored Value	*Id*
Sort Results by	*Name* *Ascending*

d. Save the quantity in stock from the user-selected merchandise record to the flow variable we already created.

Field	Value
Field	*Quantity__c*
Variable	*{!vQuantityAvailable}*

e. Click **OK**.

The Dynamic Record Choice overlay closes, and the Screen overlay appears.

6. Add a field to capture the quantity ordered in the line item.

a. From the Add a Field tab, double-click **Number**.

A number field appears in the preview pane on the right side of the Screen overlay.

b. Click **[Number]** in the preview pane.

c. On the Field Settings tab, configure the field as follows.

Field	Value
Label	*Quantity Ordered* The `Unique Name` is automatically populated based on this entry.
Scale	*0*

7. Add a field to capture the unit price of the merchandise.

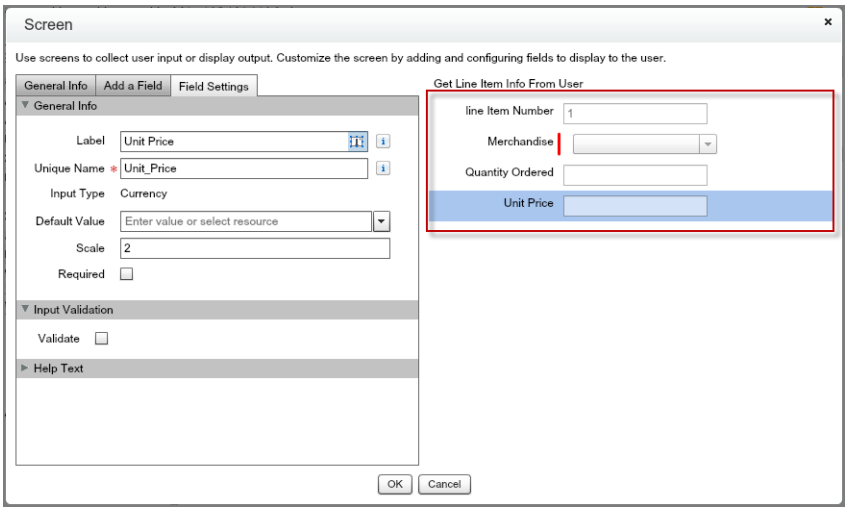
a. From the Add a Field tab, double-click **Currency**.

b. Click **[Currency]** in the preview pane.

c. On the Field Settings tab, configure the field as follows.

Field	Value
`Label`	*`Unit Price`* The `Unique Name` is automatically populated based on this entry.
`Scale`	*`2`*

The preview pane on the Screen overlay should now include four fields.



8. Click **OK**.
9. Save the flow.
   a. Click **Save**.
   b. Ignore the activation warnings for now.
   c. Click **OK**.

Step 3: Add a Record Create Element

Now that the flow can contain all the data required to create a Line Item record, let's add a Record Create element to do just that.

1. From the Palette tab, drag the Record Create onto the canvas.
2. For the `Name`, enter **Create Line Item**.
   The `Unique Name` is automatically populated based on this entry.
3. For the `Create` field, enter **Line_Item__c**.
4. Set the fields in the record using values from flow variables and screen fields.
   a. Click **Add Row** until you have five assignment rows.
   b. Set the fields and values as follows.

Field	Value
Invoice__c	*{!vInvoiceId}*
Merchandise__c	*{!Merchandise}*
Name	*{!Line_Item_Number}*
Quantity__c	*{!Quantity_Ordered}*
Unit_Price__c	*{!Unit_Price}*

5. Click **OK**.

   The Screen and Record Create elements now appear on the canvas.

6. Connect the two elements by dragging the node at the bottom of the Screen element onto the Record Create element.

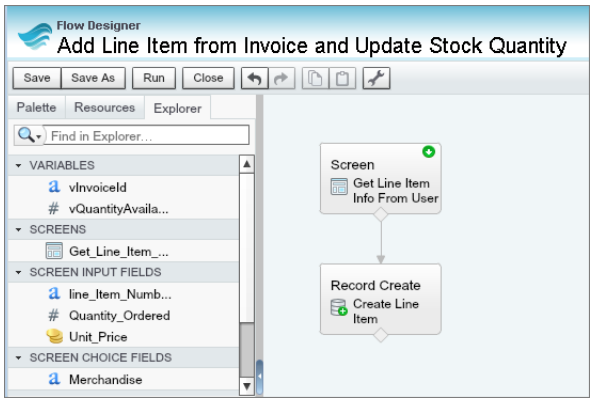
   The node doesn't move, but a connector line appears as you drag from one node to another element.

7. Hover over the Screen element and click .

   This identifies which element to execute first when the flow runs.

8. You can drag the elements to position them as you wish.

   What really matters is that the connectors link the elements together so that the flow executes them in the correct order, starting with the identified start element.



9. Click **Save**.

   Notice that the activation warnings no longer appear because we set the start element and linked the elements together.

Step 4: Add a Record Update Element

Now let's add a Record Update element to update the relevant Merchandise record, reducing the quantity available by the quantity ordered in the line item.

1. From the Palette tab, drag the Record Update onto the canvas.
2. For the `Name`, enter **Decrement Available Stock**.

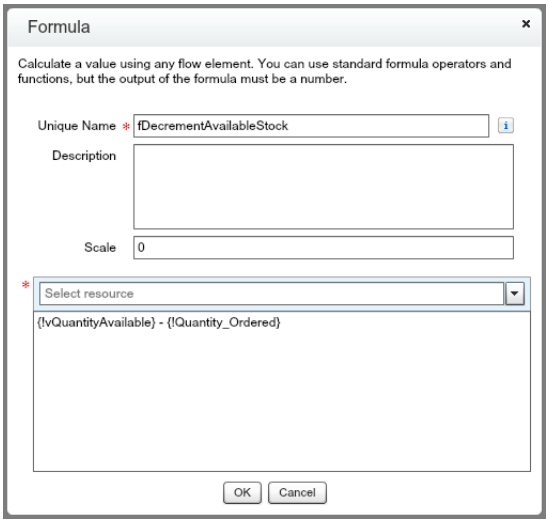
   The `Unique Name` is automatically populated based on this entry.

3. For the `Update` field, enter **Merchandise__c**.
4. Set the filter criteria so that the flow updates only the Merchandise record associated with the line item.

Field	Value
`Field`	*`Id`*
`Operator`	*`equals`*
`Value`	*`{!Merchandise}`*

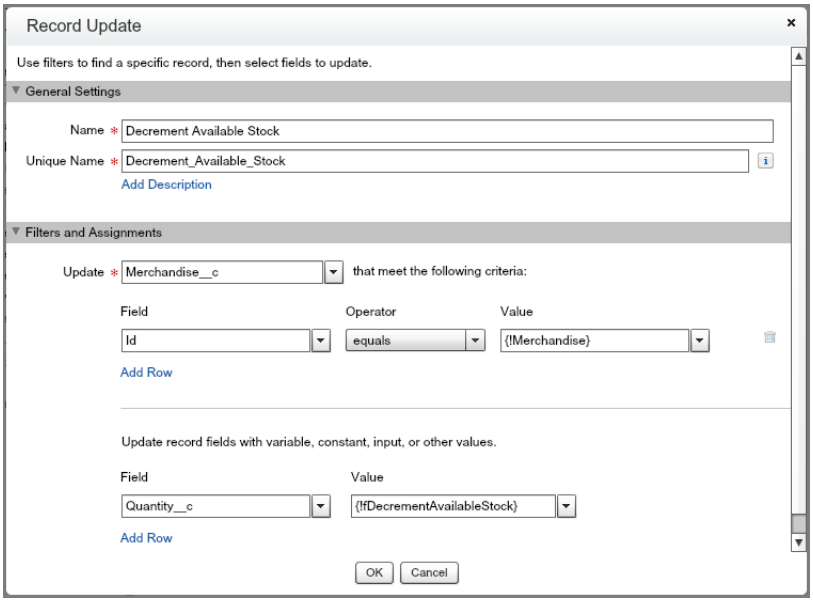
5. In the `Field` column for updating record fields, enter *`Quantity__c`*.
6. In the `Value` column, click the arrow and select **CREATE NEW** > **Formula**.
7. Configure the formula to subtract the quantity ordered in the line item from the quantity of Merchandise.

Field	Value
`Unique Name`	*`fDecrementAvailableStock`*
`Value Data Type`	Number
`Scale`	*`0`*
Formula text box	*`{!vQuantityAvailable} - {!Quantity_Ordered}`*



8. Click **OK**.

   The Formula overlay closes, and the Record Update overlay appears.



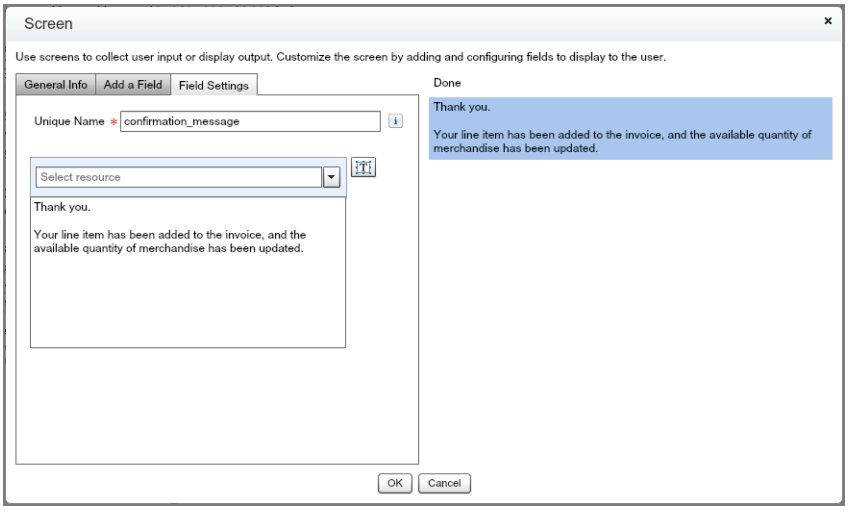
9. Click **OK**.
10. Drag the node from the bottom of the Record Create element onto the Record Update element to connect them.
11. Click **Save**.

Step 5: Add a Confirmation Screen

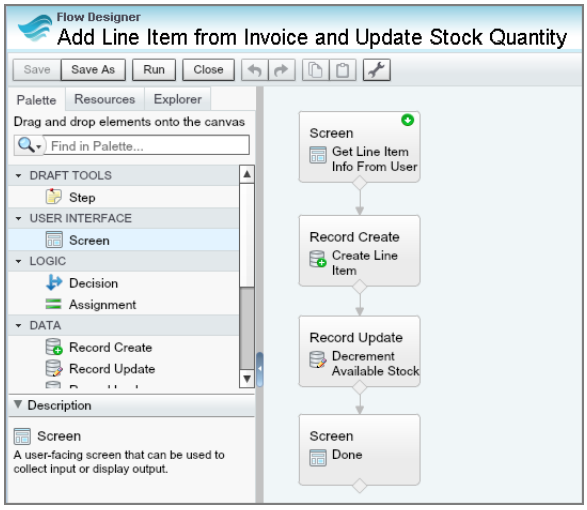
The flow is complete enough to do the job, but let's add a screen to let the flow user know that the flow has finished.

1. From the Palette tab, drag the Screen onto the canvas.
   The Screen overlay opens with the General Info tab selected.
2. For the `Name`, enter **Done**.
   The `Unique Name` is automatically populated based on this entry.
3. From the Add a Field tab, double-click **Display Text**.
4. Click **[Display Text]** in the preview pane.
5. On the Field Settings tab, configure the field as follows.

Field	Value
`Unique Name`	*`confirmation_message`*
text box	*`Thank you.`* *`Your line item has been added to the invoice, and the available quantity of merchandise has been updated.`*

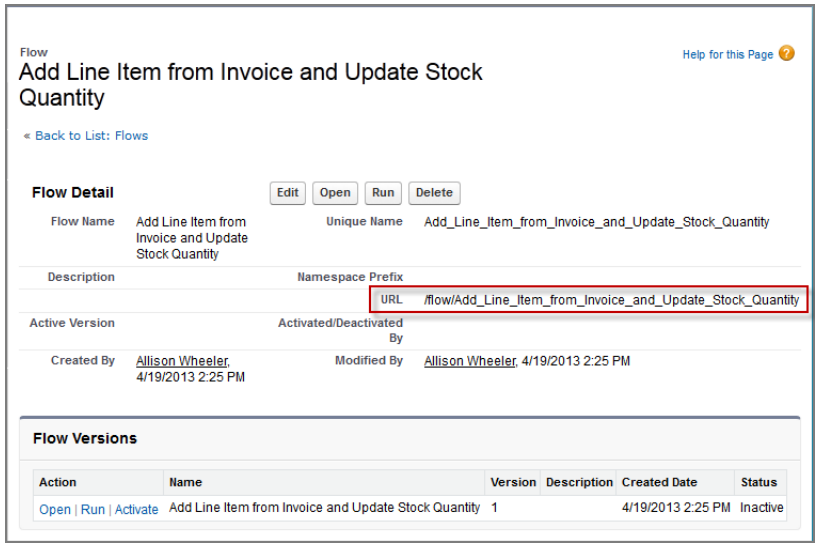


6. Click **OK**.
7. Drag the node from the bottom of the Record Update element onto the new Screen element to connect them.
8. Click **Save**.



9. Click **Close**.

   The flow detail page displays the flow URL. You'll need this later when you create the custom button for launching this flow.



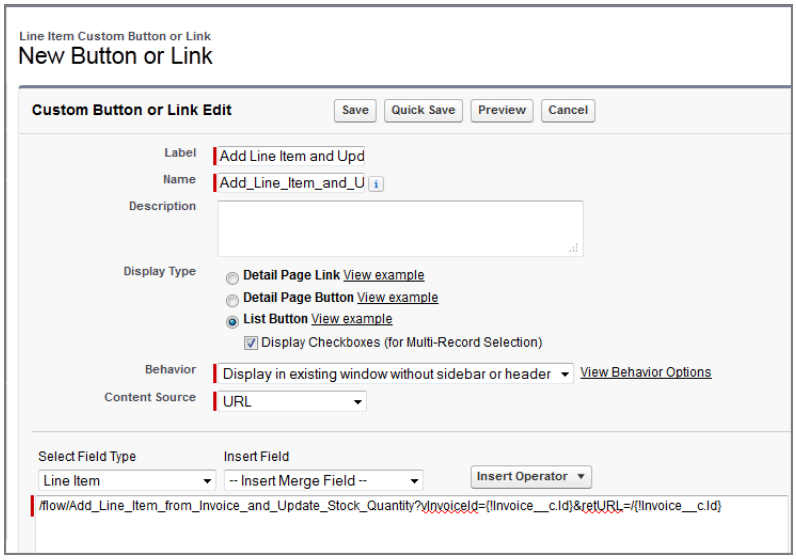
Step 6: Add a Custom Button

Now that we have a flow, let's add a custom button so that users can launch the flow from the invoice detail page. The button will:

- Launch the flow specified by the flow URL. You can find the flow URL on the flow detail page.
- Pass the relevant invoice ID into a flow variable.
- Set the flow finish behavior so that when the flow user clicks **Finish**, the browser returns the user to the relevant invoice detail page.

1. Create the custom button for the Line Item custom object.
   a. From Setup, enter *Objects* in the Quick Find box, select **Objects**, then select **Line Item**.
   b. In the Buttons, Links, and Actions related list, click **New Button or Link**.
   c. Define the custom button.

Field	Value
Label	*Add Line Item and Update Stock Qty* The Name is automatically populated based on this entry.
Display Type	*List Button*
Behavior	*Display in existing window without sidebar or header*
Content Source	*URL*
URL text box	*/flow/Add_Line_Item_from_Invoice_and_Update_Stock_Quantity ?vInvoiceId={!Invoice__c.Id}&retURL=/{!Invoice__c.Id}*



d. Click **Save**.

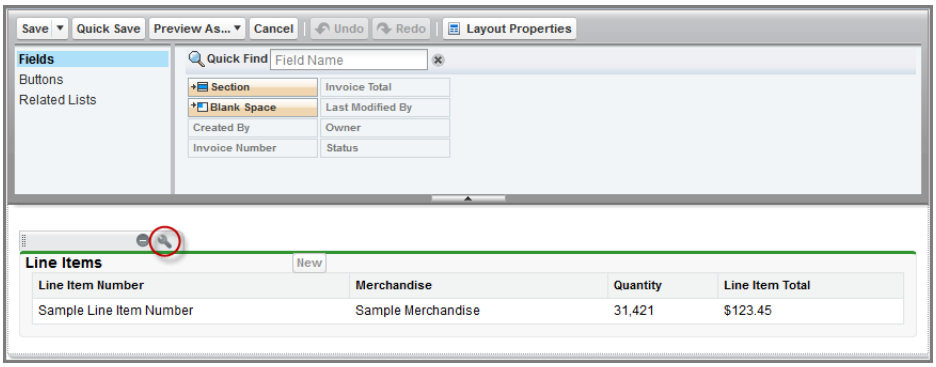
e. Click **OK**.

2. Add the custom button to the Invoice page layout.

a. From Setup, enter *Objects* in the Quick Find box, select **Objects**, then select **Invoice**.

b. In the Page Layouts related list, click **Edit** for the Invoice Layout.

c. Click for the Line Items related list.

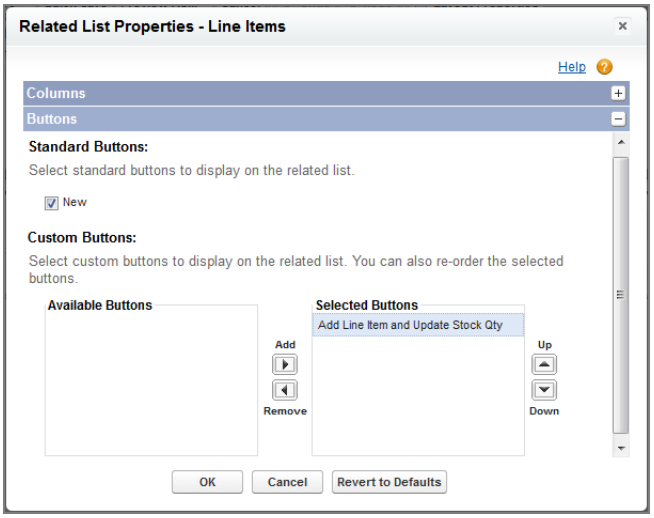


d. Expand the Buttons section.

e. Under Available Buttons, select **Add Line Item and Update Stock Qty**.

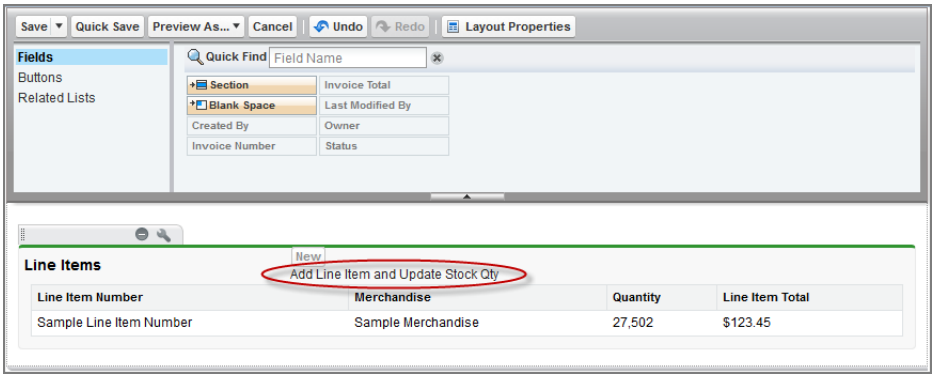
f. Click .

The button name now appears under Selected Buttons.



g. Click **OK**.

3. Verify that the button appears in the preview area for the Line Items related list.



4. Click **Save**.

Step 7: Try Out the App

Now try out the revised app and see the flow in action.

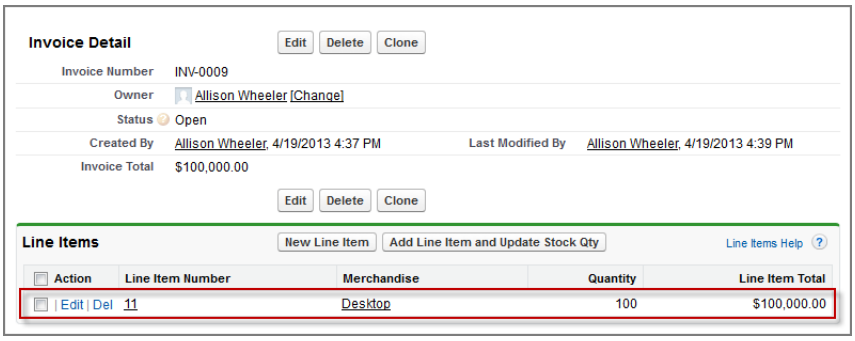
1. To make sure you get real results when you try the app, configure an existing Merchandise record to have a known starting quantity.
   a. Click the **Merchandise** tab.
   b. Click **Desktop**.
   c. Click **Edit**.
   d. Set the `Quantity` to *`1000`*.
   e. Click **Save**.

   Now when we try out the flow, we can easily verify that the merchandise quantity is updated correctly.
2. Click the **Invoices** tab and either create a new invoice or edit an existing one.
3. Click the **Add Line Item and Update Stock Qty** custom button you just created.

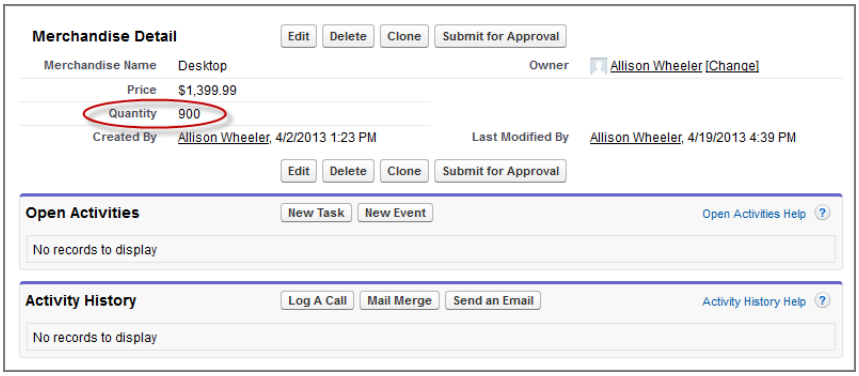
4. To see results on the invoice detail page, enter the following values for the line item.

Field	Value
`Line Item Number`	*11*
`Merchandise`	*Desktop*
`Quantity Ordered`	*100*
`Unit Price`	*1000*

5. Click **Next** and then **Finish**.
6. Verify that the line item correctly appears on the invoice detail page.



7. Click the **Merchandise** tab.
8. Click **Desktop**.
9. Verify that the `Quantity` changed from 1000 to 900.



Congratulations! You've successfully updated your inventory. But what happens if an error occurs? The standard behavior is for the flow to email the organization administrator a generic message, but you can modify the flow to also immediately notify the user. This is covered in the next, optional, tutorial.

Step 8: Add a Fault Screen

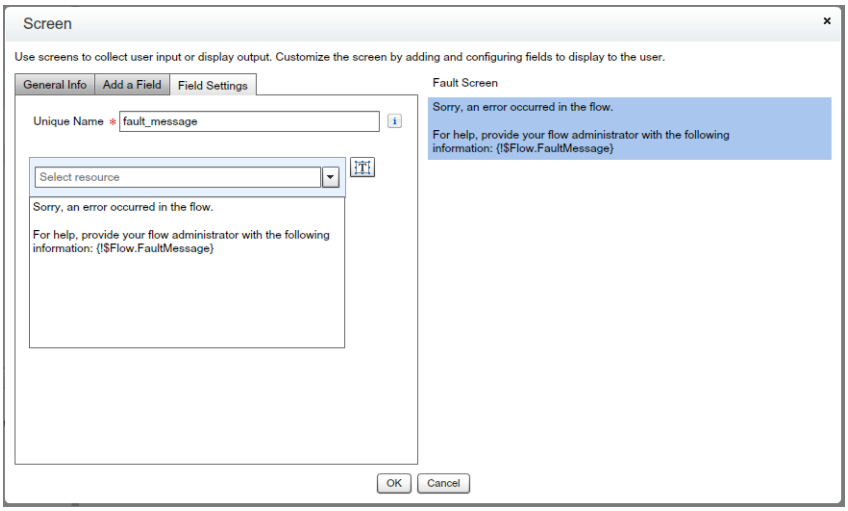
If an error occurs while the flow is interacting with the database, the flow displays a generic unhandled fault message. The system sends the organization administrator an email with information to help identify the issue. You can also set up fault paths to a screen that displays this information to the flow user. Instead of waiting for a system message to reach your email account, you can view the information in the flow and immediately fix the problem.

1. Reopen the flow if necessary.
2. From the Palette tab, drag the Screen onto the canvas.

   The Screen overlay opens with the General Info tab selected.
3. For the `Name`, enter **Fault Screen**.

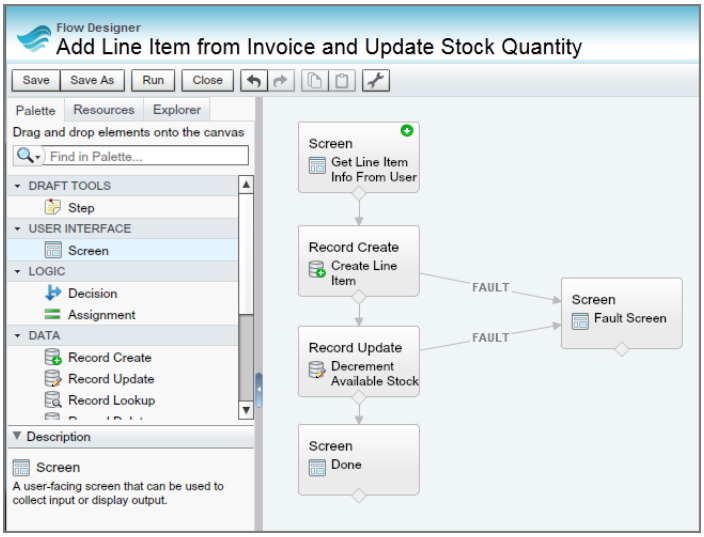
   The `Unique Name` is automatically populated based on this entry.
4. From the Add a Field tab, double-click **Display Text**.
5. Click **[Display Text]** in the preview pane.
6. On the Field Settings tab, configure the field as follows.

Field	Value
`Unique Name`	*`fault_message`*
text box	*`Sorry, an error occurred in the flow.`* *`For help, provide your flow administrator with the following information: {!$Flow.FaultMessage}`*



7. Click **OK**.
8. Connect both the Record Create and Record Update elements to the fault screen.
   a. Drag the node from the bottom of the Record Create element onto the new Screen element.
   b. Drag the node from the bottom of the Record Update element onto the new Screen element.

   Notice that these new connectors have "FAULT" labels.



9. Click **Save**.

10. Click **Close**.

Now, if the flow encounters a validation rule or error, the flow user sees a meaningful error message instead of the generic unhandled fault message.

If your flow is already working fine, you can still test the fault screen by entering a value that would fail the validation rule you created in an earlier tutorial. Specifically, while running the flow, enter a `Quantity Ordered` value that's obviously greater than the quantity available for the merchandise.

ANALYZE DATA WITH REPORTS AND DASHBOARDS

Level: Beginner; **Duration:** 30–40 minutes

How great would it be to get a report in your inbox every morning that tells you how much stock you have for each item in your warehouse? Or perhaps you'd like to see that information displayed as a graphical chart whenever you access the app on your phone?

This series of tutorials introduces you to reports and dashboards, or what we refer to as *Salesforce Reports and Dashboards*. Once you've defined your reports, you can place them on a dashboard, so you can see all your key metrics at a glance. Salesforce Reports and Dashboards lets you see what's important to you, exactly how and where you want to see it.

Create a Report

Level: Beginner; **Duration:** 15 minutes

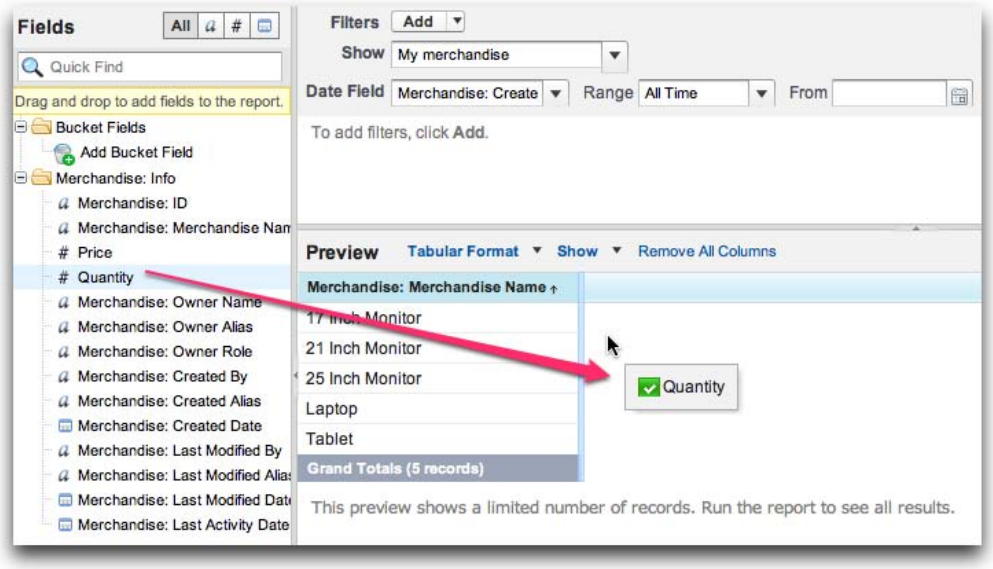
The Warehouse app you created with the App Quick Start wizard includes a Reports tab, where you can create, edit, run, and schedule reports. Start by creating a simple report that tells you how much stock you have for each item in your warehouse. Then you'll use groupings and filters to get the most out of the data in your report.

Try out buckets for on-the-fly grouping, and experiment with showing your report data graphically as a chart. And once you've got charts mastered, take a look at how you can provide users with valuable context by embedding charts in record detail pages.

Step 1: Create a Simple Report

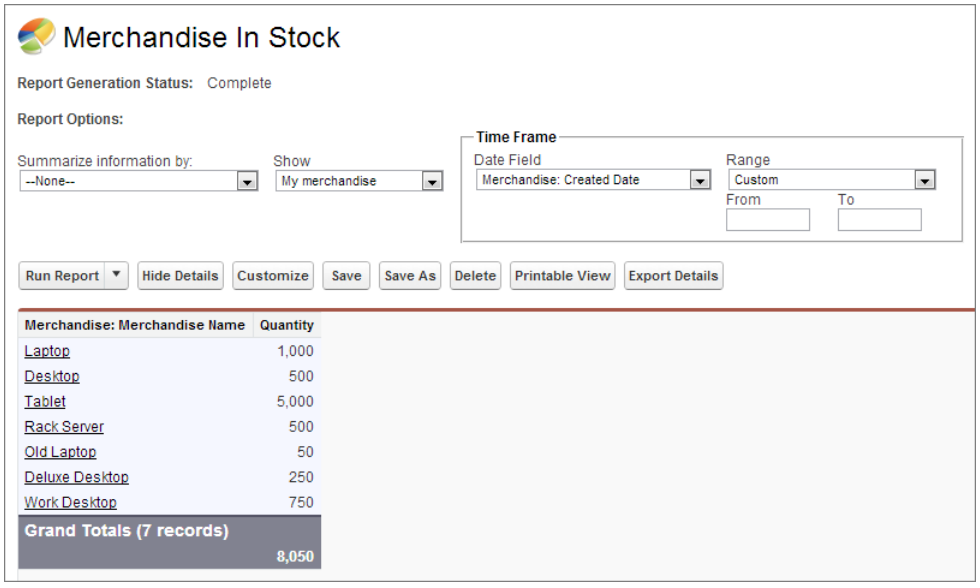
In this step, you create a simple tabular report that shows the merchandise in your warehouse and how many pieces of each are in stock. Tabular reports present data in simple rows and columns, much like a spreadsheet. They can be used to show column summaries, like sum, average, maximum, and minimum.

1. From the Reports tab, click **New Report**.
2. In the Quick Find box, enter `Merchandise`, and in the **Other Reports** folder, choose **Merchandise**.
3. Click **Create**.
4. In the report builder, notice that the `Merchandise Name` field is already there. You only need one more field: the quantity of each item. From the Fields pane, drag `Quantity` onto the preview.



5. Click **Save**, and give your report a meaningful name, such as `Merchandise in Stock`.
6. In the Report Folder drop-down list, select Unfiled Public Reports, so everyone can access it. (If you didn't want this report to be accessible to everyone, you'd create a folder and give different people different levels of access to it. More on that later.)
7. Click **Save and Run Report**.

That's it. Your new report is ready to go!



You can get fancy with reports, but that's all you need from this one. And as you'll soon see, even this simple report gives you a lot of functionality.

- Use the Summarize Information by drop-down list to summarize the report based on any field on the Merchandise object. For example, you could summarize on `Owner Name` to see who entered each piece of merchandise, as well as the count.
- Use the Show drop-down list specify whether you want to see just your merchandise, your team's merchandise, or all merchandise.

- In the Time Frame section, you can choose to run this report based on the created, modified, or last activity date, as well as choose the date range for the data you want to see.
- Click **Run Report**, and choose to run the report now or on some future date. If you choose the latter, it takes only a few more clicks to have that report in your inbox every day—or however often you want it.
- If you'd rather see a summary than a bunch of details, click **Hide Details**.
- Click **Customize** to make changes to the report, and you'll return to the familiar drag-and-drop interface you used to create the report.
- And finally, you can export the report as a printed document, spreadsheet, or CSV file by clicking **Export Details**.

Tell Me More....

- Click the column headers to toggle between ascending and descending order. The Grand Totals indicates the record count as well as the summaries you chose. Click **Customize** to make additional changes to this report.
- You can click through to the data records that are being reported on, a characteristic found in all reports on Salesforce. For example, click the name of any merchandise record listed in the report to view its detail page.
- A report folder's sharing settings determine who can do what with reports in that folder. Click next to the folder in the Reports tab and click **Share**. You can give people three levels of access: Viewer, Editor, or Manager. For more information, see "Share a Report or Dashboard Folder" in the Salesforce Help.

Step 2: Get More Information Out of Your Report

The report builder gives you a lot of ways to view your data. Viewing data in groups usually helps make sense of what you're looking at. In this case, grouping by item, price, or total units sold can be helpful.

First we'll turn our simple tabular report into a slightly fancier summary report, and then we'll give it a grouping.

1. Click **Customize**.
2. The default format is tabular, but we want a summary report. Click **Tabular Format** and choose **Summary** instead.
3. Find and drag the `Price` field to your report.
4. Click next to `Price`, click **Summarize this Field**, select `Average`, and then click **Apply**.
5. Click next to `Quantity`, click **Summarize this Field**, select `Sum`, and then click **Apply**.
6. Select the `Merchandise Name` field (either from Fields or Preview panel) and drag it to the area labeled **Drop a field here to create a grouping**. This aggregates data by the unique merchandise item.

The report is now grouped by merchandise, and it includes the sum of quantity and the average price for each level.

Tell Me More...

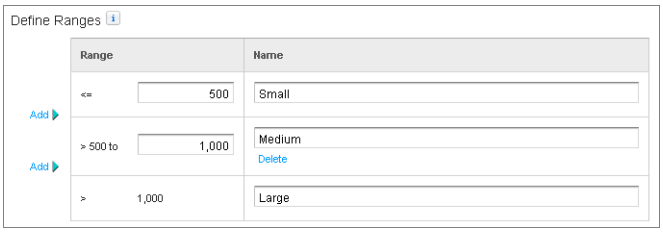
Try adding a *cross filter* from the Add drop-down list in the Filters pane. A cross filter lets you filter on the report's child objects using a simple *`with`* or *`without`* condition. To learn more about cross filters, watch Using Cross Filters in Reports.

Step 3: Add Buckets to Your Report

Bucketing lets you quickly group report records without creating a formula or a custom field. For example, say you also want to group by quantity into ranges. To do this, create a bucket field on `Quantity` and define the ranges.

First, create a bucket field based on `Quantity` with ranges for small, medium, and large. You'll use the bucket field to create the grouping.

1. Click ▼ on `Quantity` and click **Bucket this Field**.
2. Enter a bucket field name, *`Quantity Range`*.
3. Define ranges as *`Small`* (500), *`Medium`* (between 500–1000), and *`Large`* (greater than 1000).



4. Click **OK**.
5. Grab the Quantity Range bucket field that's already on the report and make it the first-level grouping by dragging it onto the drop zone above `Merchandise Name`.

Now the report shows data grouped in two levels—first, by quantity range (small, medium or large), and second, by merchandise name.

Quantity	Price
Quantity Range: Small **(4 records)**	
1,300	avg $1,497.99
Merchandise: Merchandise Name: Deluxe Desktop **(1 record)**	
250	avg $1,399.99
250	$1,399.99
Merchandise: Merchandise Name: Desktop **(1 record)**	
500	avg $1,000.00
500	$1,000.00
Merchandise: Merchandise Name: Old Laptop **(1 record)**	
50	avg $345.99
50	$345.99
Merchandise: Merchandise Name: Rack Server **(1 record)**	
500	avg $3,245.99
500	$3,245.99
Quantity Range: Medium **(2 records)**	
1,750	avg $823.00
Merchandise: Merchandise Name: Laptop **(1 record)**	
1,000	avg $500.00
1,000	$500.00
Merchandise: Merchandise Name: Work Desktop **(1 record)**	
750	avg $1,145.99
750	$1,145.99
Quantity Range: Large **(1 record)**	
5,000	avg $300.00
Merchandise: Merchandise Name: Tablet **(1 record)**	
5,000	avg $300.00
5,000	$300.00
Grand Totals (7 records)	
8,050	**avg $1,133.99**

Tell Me More....

You can filter a bucket field just like other fields in the report. For example, set a filter for `Quantity Range not equal to Small` to see only merchandise with quantities in the medium or large range.

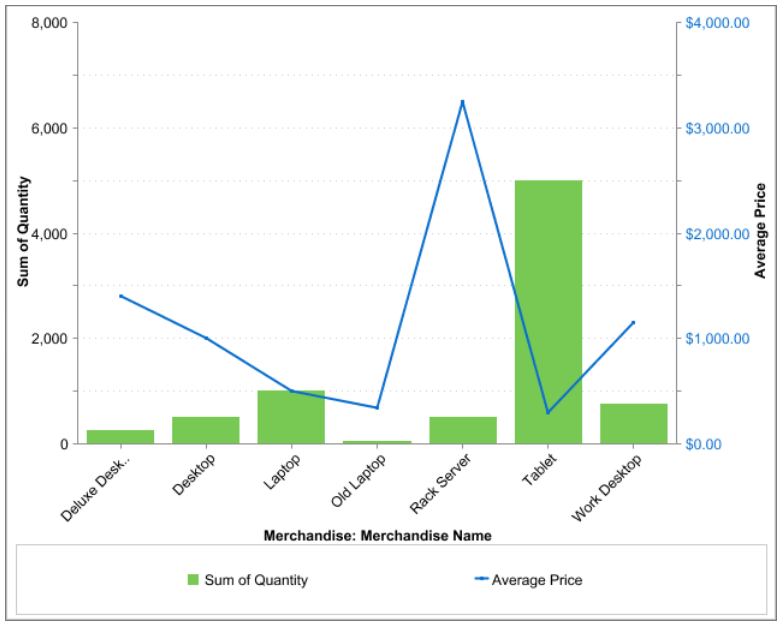
To learn more about bucket fields, watch Getting Started with Buckets.

Step 4: Show Your Report Data as a Chart

It's often a good idea to give users a visual way to understand the data in your report. Let's add a combination chart to our report now.

1. In the Preview pane, click **Add Chart** to create a chart to represent your data. In the Chart Editor that appears, click the vertical bar chart.
2. In the Y-Axis drop-down list, leave Sum of Quantity selected.
3. In the X-Axis drop-down list, select Merchandise: Merchandise Name. Notice the bucket field, Quantity Range, is also available, as there are two groupings.
4. Select `Plot additional values`.
5. In the Display drop-down list, select Line.
6. Select `Use second axis`.
7. In the Value drop-down list, select Average of Price.
8. Click **OK**, then **Save**.

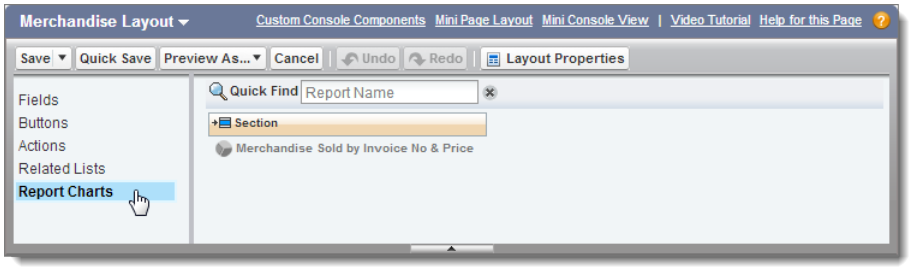
The combination chart shows merchandise in stock (bars) against average price (line).



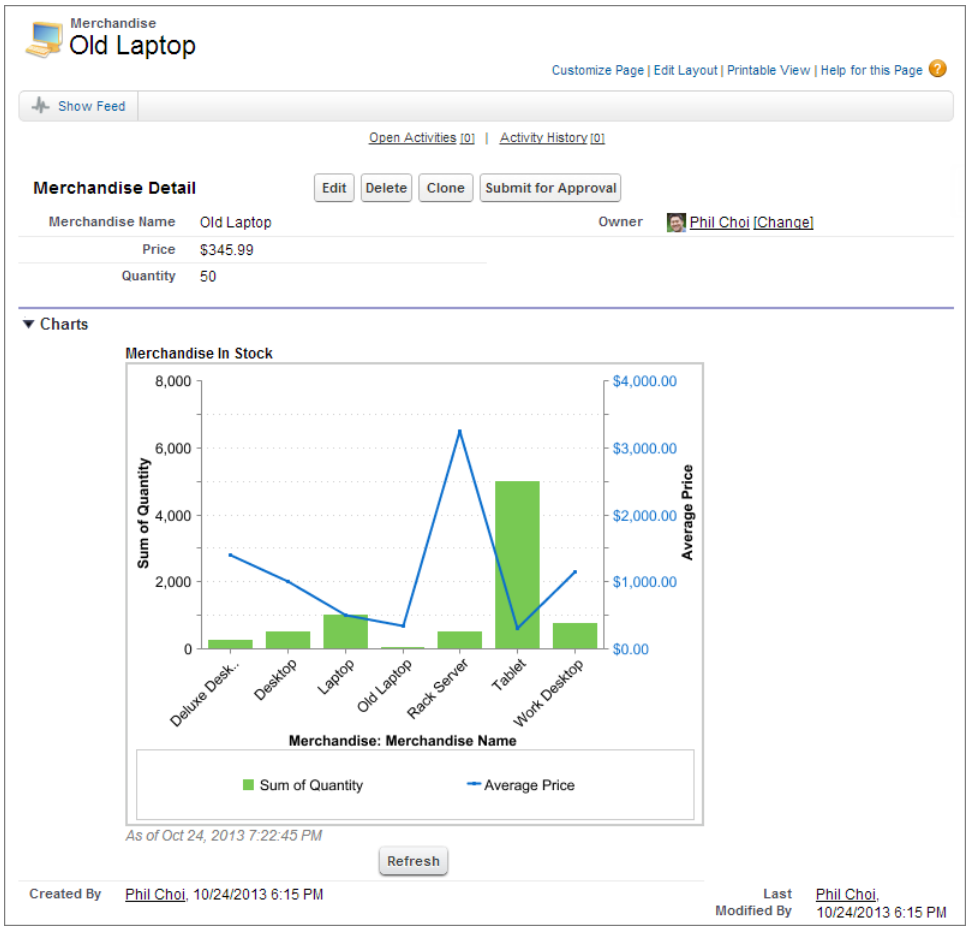
Step 5: Embed the Report Chart in a Record Page

There are many ways to share reports once you've created them. One of the best is to embed the report's chart on a record detail page, where users can see it as they do their work: no need to jump over to the Reports tab. In Modify a Page Layout on page 32, you learned how to customize what's on the detail page for a particular type of record. Now we'll do that for merchandise records.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects**, then choose **Merchandise**.
2. Under the Page Layouts related list, click **Edit** next to Merchandise Layout.
3. Click **Report Charts** in the palette.



4. Drag the **Section** element onto the preview pane and place it above the Mobile Cards area. Enter *Charts* for the section name, and select 1-column for the layout.
5. In the Quick Find box, type the name of the report and click to find and select the report chart. (You can add two if you want.) You can browse up to 200 recently viewed reports. But you only see reports that already have charts.
6. Drag the **Merchandise In Stock** report chart onto the layout.
7. Click **Save** and go look at a merchandise record. It will look something like:



Now users can quickly see how much merchandise is in stock, without leaving their record detail page! Notice that, by default, the chart is automatically filtered to show data that's relevant to the particular record type you're looking at. You can set different filters back on the page layout. Just click on the chart to customize it.

To learn more about embedding report charts on record pages, watch Embedding Charts Anywhere.

Tell Me More....

Salesforce provides the Reports and Dashboards REST API that lets you access your data remotely and build your own apps and visualizations. There's an API resource for almost anything you can do with reports through the standard web interface. For example, say you've used Visualforce to build a custom app, and you want to give that app a Reports tab. Or your users need a special kind of chart that isn't one of the out-of-the-box report builder options.

For a quick start on using the Reports and Dashboards REST API, see the *Salesforce Reports and Dashboards REST API Developer Guide*.

Create a Dashboard

Level: Beginner; **Duration:** 15 minutes

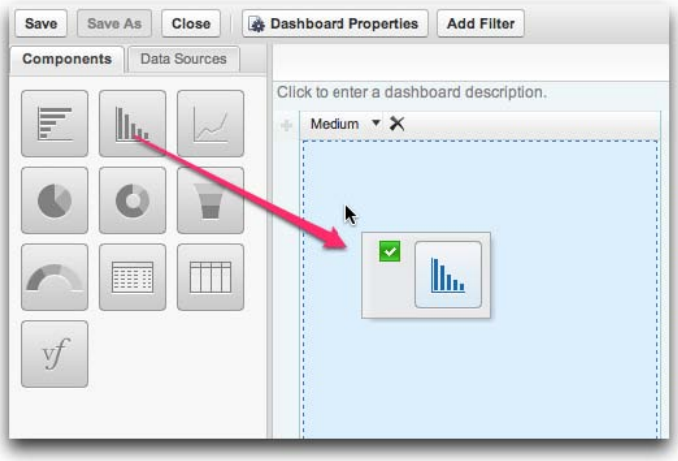
Dashboards in Salesforce are like a dashboard in your car, showing you important information at a glance. Dashboards can show data in charts, gauges, tables, metrics, or Visualforce pages. Naturally, you can customize dashboards to show exactly what you want.

In this tutorial, you create a new dashboard that's powered by the report you created in the previous tutorial.

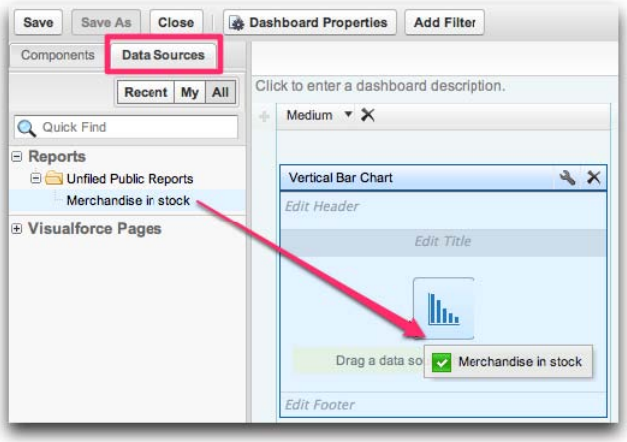
Step 1: Create a New Dashboard

Create a new dashboard for the Warehouse app that's powered by the Merchandise in Stock report that you've created.

1. Click the Reports tab and then **New Dashboard**.
2. Click the editor's Components tab, then drag the Vertical Bar Chart component and drop it in the first column of the new dashboard.



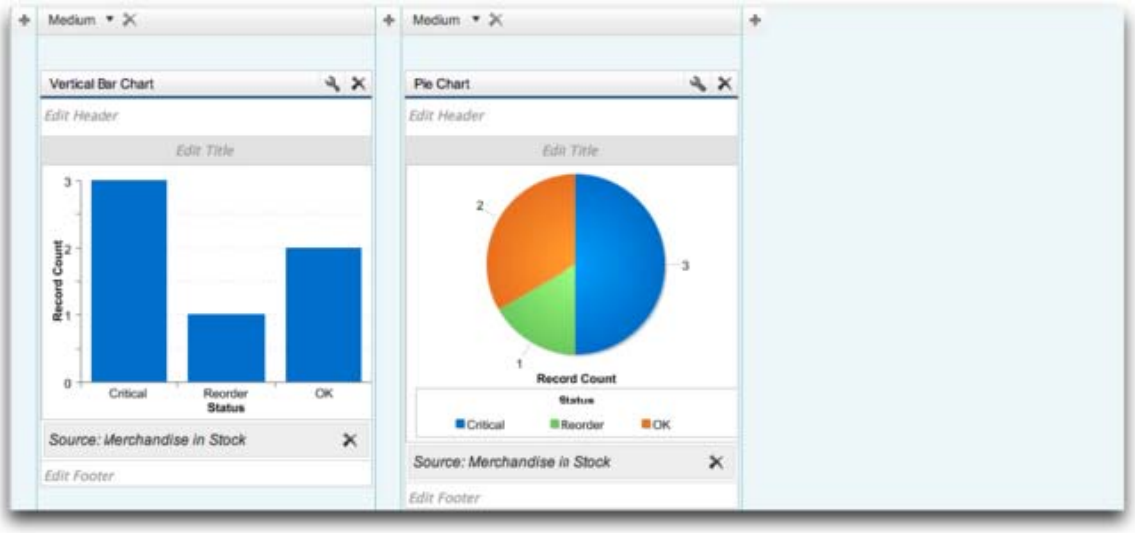
3. Now click the editor's Data Sources tab, and under **Reports** > **Unfiled Public Reports**, drag your report and drop it on top of the new Vertical Bar Chart component that's in the dashboard.



Step 2: Add a Pie Chart Component

That was so easy. Why not play around with adding a different type of dashboard component, just for fun?

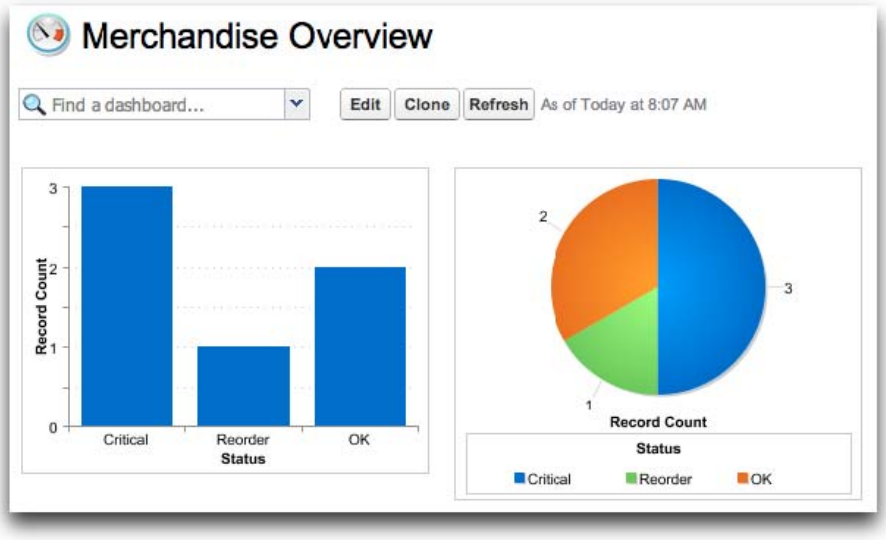
1. Repeat the previous steps, but this time use a Pie Chart component in the second column.
2. Then click Remove this column ( ) in the header of the third column to remove the unused third column from the layout. When you are finished, the dashboard preview should look similar to the following.



3. Click **Save**, name the dashboard `Merchandise Overview`, and click **Save**.

Step 3: Try Out the App

1. **Close** the editor, and in the pop-up dialog, choose **Save and Close**. The dashboard then runs automatically when you leave the editor. Your dashboard should look similar to the following image.



2. To access the dashboard at any time, click the Reports or Dashboard tab in the Warehouse app.

Tell Me More....

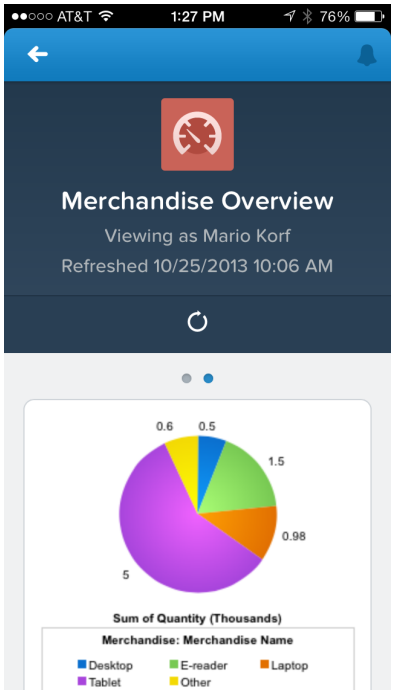
- When you set a running user for a dashboard, it runs using the security settings of that single, specific user. All users with access to the dashboard see the same data, regardless of their own personal security settings. To set the running user, click ▼ next to the `View dashboard as` field.

- Dashboards can be updated either manually or on a schedule, and they can be delivered through email and mobile.
- A dashboard won't automatically refresh unless it is set to do so. Each time you view a dashboard, it indicates in the upper-right corner when it was last refreshed. To update the data in the dashboard, click **Refresh**.
- Try adding a filter when editing the dashboard by clicking **Add Filter**. A filter lets you see different views of dashboard data based on filter conditions. You can add up to three filters per dashboard with up to ten conditions on a filter. Instead of filtering at the report level, you directly manipulate dashboard data.

Step 4: Access Dashboards from Your Mobile App

Mobile dashboards give you the fastest and clearest way to see what's important to you at a glance, on the go.

1. In Salesforce1, tap to open the navigation menu.
2. Tap **Dashboards**, and from the list of recent dashboards, tap **Merchandise Overview**.
3. Dashboards look and navigate a bit different on a mobile device. To switch columns, swipe left and right.



4. Tap a component to see the details of a component.
5. In the component view, tap data points to see their values highlighted.

Unleash Your Reports with the Salesforce Reports and Dashboards REST API

Level: Intermediate; **Duration:** 40 minutes

You've learned how to do some pretty sophisticated things with reports in this workbook so far. But what if you need more? What if your users can't live without a custom app specially tailored to their own unique business requirements?

No problem! If you can code a little, or have access to someone who can, Salesforce provides an API that can handle almost anything you can do with reports through the standard web interface.

For example, say you've used Visualforce to build a custom app, and you want that app to display report data. Or your users need a special kind of chart that isn't one of the out-of-the-box options when they build a dashboard. In this tutorial, we'll take a look at some ways you might give your users what they want.

Note: This is just a brief overview to show you some of the cool kinds of things you can do with the Salesforce Reports and Dashboards REST API. For full instructions and a detailed reference, check out the *Salesforce Reports and Dashboards REST API Developer Guide*.

To use the API, you have your app send a request to a URL that's based on the instance where your Salesforce organization is running. For example, if your organization is hosted on `na1.salesforce.com`, you could get a list of all the reports you have by sending a request to `https://na1.salesforce.com/analytics/reports`.

Here are the basic operations you can undertake with the Salesforce Reports and Dashboards REST API. We'll be using some of these in the next few steps.

Action	URL	Method	Request Body
List all recently used, supported reports.	`/analytics/reports`	`GET`	N/A
Retrieve report, report type, and related metadata for the specified report.	`/analytics/reports/<reportId>/describe`	`GET`	N/A
Run the specified report.	`/analytics/reports/<reportId>`	`GET`	N/A
Run the specified report with dynamic filters.	`/analytics/reports/<reportId>`	`POST`	Report Metadata
Run the specified report asynchronously.	`/analytics/reports/<reportId>/instances`	`POST`	N/A
Run the specified report asynchronously with filters.	`/analytics/reports/<reportId>/instances`	`POST`	Report Metadata
List the 200 most recent run instances of the specified report.	`/analytics/reports/<reportId>/instances`	`GET`	N/A
Fetch the specified run instance of the specified report.	`/analytics/reports/<reportId>/instances/<instanceId>`	`GET`	N/A
Get a list of recently used dashboards.	`/analytics/dashboards`	`GET`	N/A
Retrieve metadata, data, and status for the specified dashboard.	`/analytics/dashboards/<dashboardID>`	`GET`	N/A
Trigger a dashboard refresh.	`/analytics/dashboards/<dashboardID>`	`PUT`	N/A
Get the status for the specified dashboard.	`/analytics/dashboards/<dashboardID>/status`	`GET`	N/A

All Salesforce Reports and Dashboards REST API resources are accessed using:

- A base URI for your company (for example, `https://na1.salesforce.com`)

- Version information (for example `/services/data/v29.0/analytics`)
- A named resource (for example, `/reports`)

Put together, an example of the full URL to the resource is:

```
https://na1.salesforce.com/services/data/v29.0/analytics/reports/
```

Step 1: Run a Report Synchronously

If speed is what you need, synchronous execution is the way to go. Your report runs afresh every time the user looks at it, and feeds it right back to your app. If your users need to track hour-by-hour changes, you may want to run your report synchronously.

Let's get acquainted with the Salesforce Reports and Dashboards REST API by running our Merchandise report. We'll run it synchronously this time, and without any filters.

- Kick off the report by sending a `GET` command with the ID of the report you want to run.

  The command will look like this:

  ```
  curl -s -H 'Authorization: OAuth token ...'
  https://na1.salesforce.com/services/data/v29.0/analytics/reports/00OD0000001ZbP7MAK
  ```

  Note: We're using NA1 as the instance for this example. Substitute the instance where your Salesforce organization is hosted.

You've just run your first report via the API! Don't worry about reading the results yet. You'll get to that in the next few pages.

Step 2: Run a Report Asynchronously

Running a report asynchronously means sending the request, then getting the results back at some later time. There are a few advantages to running reports asynchronously through the API.

- When you run asynchronously, the results are kept around in a cache that you can use any time during the next 24 hours. And the API commands for reusing cached results don't count against the 1200-requests-per-hour limit. (General API request limits still count, though.)
- Asynchronous reports have a longer time-out interval. So if you know your report is looking at a very large data set and you don't want to risk timing out, you might want to run asynchronously.
- You can run up to 1200 asynchronous reports per hour, which is over twice the limit for synchronous reports. So if you expect a lot of users to be looking at your app, asynchronous runs might be for you.

1. Kick off your asynchronous report by sending a `POST` command to `https://<instance>//analytics/reports/<reportId>/instances`.

   The command will look like:

   ```
   curl -s -H 'Authorization: OAuth token ...'
   https://na1.salesforce.com/services/data/v29.0/analytics/reports/00OD0000001ZbP7MAK/instances
    -X POST -d ''
   ```

2. To get the results of your asynchronous run, poll the report run instance with `GET`.

A specific asynchronous run of a report is called an instance. Each instance has an ID. To get the data set that an instance contains, you send a request to the system, identifying the instance you want by its ID. This is called polling the instance. If the report has finished running, the response to your poll request is the data set you asked for. (If it's not finished, you get an "in progress" response.)

```
curl -s -H 'Authorization: OAuth token ...'
https://na1.salesforce.com/services/data/v29.0/analytics/reports/00OD0000001ZbP7MAK/instances/instance_id
```

Now we've run a report synchronously and asynchronously. Next, we'll make our data more useful by narrowing down our results.

Step 3: Filter Report Data

A report is most useful when you use filters to narrow down the data it returns.

You learned how to set filters on the fly, using the standard web interface, in Create a Report on page 79. You can filter a report via the API as well. The API has commands to add filters, edit them, or remove them.

For example, say you've just run a saved report that is filtered to show only items that you have more than a dozen of. Now you want to filter for smaller quantities, without changing the saved report. To do this, send back the report metadata object with edited filters.

1. Here's some typical metadata that your report run might have returned:

```
'{reportMetadata":{"name":"MerchandiseReport","id":"00OD0000001ZbP7MAK","developerName":"MerchandiseReport",
"reportType":{"type":"MerchandiseList","label":"Merchandise"},"reportFormat":"MATRIX",
"reportBooleanFilter":null,"reportFilters":[{"column": "QUANTITY",
"operator":"greaterThan",
"value":"12"}],"detailColumns":["MERCHANDISE.NAME","CREATED_DATE","QUANTITY"],
"currency":null,"aggregates":["RowCount"],"groupingsDown":[{"name":"CONTACT2.COUNTRY_CODE",
"sortOrder":"Asc","dateGranularity":"None"}],"groupingsAcross":[{"name":
"OWNER","sortOrder":"Asc","dateGranularity":"None"}]}}'
```

2. Change the filter and run the report. It will look something like this, with the edited filter shown in bold type. (This example is synchronous, but an asynchronous run works the same way.)

```
curl -s -H 'Authorization: OAuth token ...'
https://na1.salesforce.com/services/data/v29.0/analytics/reports/00OD0000001ZbP7MAK -X
 POST -d '{reportMetadata":{"name":"MerchandiseReport","id":"00OD0000001ZbP7MAK",
"developerName":"MerchandiseReport","reportType":{"type":"CaseList","label":"Cases"},
"reportFormat":"MATRIX","reportBooleanFilter":null,"reportFilters":[{"column": "QUANTITY",
 "operator":"lessThan", "value":"12"}],
"detailColumns":["MERCHANDISE.NAME","CREATED_DATE","QUANTITY"],
"currency":null,"aggregates":["RowCount"],"groupingsDown":[{"name":"CONTACT2.COUNTRY_CODE",
"sortOrder":"Asc","dateGranularity":"None"}],"groupingsAcross":[{"name":"OWNER",
"sortOrder":"Asc","dateGranularity":"None"}]}}'
```

You've just run a filtered report and retrieved the data. You're ready to do some cool tricks with it! For some ideas, along with full instructions and detailed reference information, check out the *Salesforce Reports and Dashboards REST API Developer Guide*.

Step 4: Find, Show, and Refresh Dashboards

Many users interact with reports mainly through dashboards. You can use the Reports and Dashboards REST API to access and refresh dashboards just as easily as you can with reports.

For example, suppose your users are tired of paging through screens in search of the dashboards they need. You can use the Reports and Dashboards REST API to let them choose from among the dashboards they've looked at recently.

1. To help your users find their dashboards easily, use a GET request on the Dashboard List resource to retrieve a list of recently used dashboards.

```
/services/data/v31.0/analytics/dashboards
```

For each dashboard, the Dashboard List resource sends you back something like this. The URL handle stores the status or results for the dashboard. The list is sorted by the date when the dashboard was last refreshed.

```
[ {
  "id" : "01ZD00000007QeuMAE",
  "name" : "Adoption Dashboard",
  "statusUrl" : "/services/data/v31.0/analytics/dashboards/01ZD00000007QeuMAE/status",
  "url" : "/services/data/v31.0/analytics/dashboards/01ZD00000007QeuMAE"
}]
```

2. You may want to show users their dashboard data in different ways, depending on the platform or device where they're using your app. You can pull the data from the dashboard with a GET request to the Dashboard Results resource.

```
/services/data/v31.0/analytics/dashboards/01ZD00000007S89MAE
```

What you get back is the actual data in the dashboard, plus its metadata (the dashboard ID, name, component metadata, and any filters) and its refresh status. The result will look like this:

```
{
{
  "componentData" : [ {
    "componentId" : "01aD0000000a36LIAQ",
    "reportResult" : {
      // Report result data omitted for brevity.
    },
    "status" : {
      "dataStatus" : "DATA",
      "errorCode" : null,
      "errorMessage" : null,
      "errorSeverity" : null,
      "refreshDate" : "2014-04-10T20:37:43.000+0000",
      "refreshStatus" : "IDLE"
    }
  } ],
  "dashboardMetadata" : {
    "attributes" : {
      "dashboardId" : "01ZD00000007S89MAE",
      "dashboardName" : "Simple Dashboard",
     "statusUrl" : "/services/data/v31.0/analytics/dashboards/01ZD00000007S89MAE/status",

      "type" : "Dashboard"
    },
    "canChangeRunningUser" : false,
    "components" : [ {
      "componentData" : 0,
      "footer" : null,
      "header" : null,
      "id" : "01aD0000000a36LIAQ",
      "properties" : {
        "aggregateName" : "s!AMOUNT",
        "maxRows" : null,
```

```
        "sort" : {
          "column" : "TYPE",
          "sortOrder" : "asc"
        },
        "visualizationProperties" : { },
        "visualizationType" : "Bar"
      },
      "reportId" : "00OD0000001g2nWMAQ",
      "title" : null,
      "type" : "Report"
    } ],
    "description" : null,
    "developerName" : "Simple_Dashboard",
    "filters" : [ {
      "name" : "Amount",
      "options" : [ {
        "alias" : null,
        "endValue" : null,
        "id" : "0ICD00000004CBiOAM",
        "operation" : "greaterThan",
        "startValue" : null,
        "value" : "USD 2000000"
      } ],
      "selectedOption" : null
    } ],
    "id" : "01ZD00000007S89MAE",
    "layout" : {
      "columns" : [ {
        "components" : [ 0 ]
      } ]
    },
    "name" : "Simple Dashboard",
    "runningUser" : {
      "displayName" : "Allison Wheeler",
      "id" : "005D00000016V2qIAE"
    }
  }
}
```


3. If you're concerned that users might not be seeing the latest data, you can refresh a dashboard remotely by sending a PUT Dashboard Results request, specifying the dashboard you want to refresh by its ID.

```
/services/data/v31.0/analytics/dashboards/01ZD00000007S89MAE
```

The response contains the status URL for the refreshed dashboard:


```
{
  "statusUrl" : "/services/data/v31.0/analytics/dashboards/01ZD00000007S89MAE/status"
}
```

ENHANCE THE MOBILE EXPERIENCE WITH ACTIONS

Level: Beginner; **Duration:** 20–25 minutes

You've already seen how the functionality in your app is available from a mobile device. Indeed, you could say that every Salesforce developer is a mobile developer! But so far, you've only exposed some data and customized the layout. What's really awesome is when you can provide users with custom mobile functionality that allows them to be highly productive on the go.

In this tutorial you create *quick actions*. Quick actions are split into two categories, *global actions*, and *object-specific actions*. Global actions are used when you want to create something quickly from pretty much anywhere in the app. Object-specific actions are used when you want to automatically associate what you're doing with something else.

Quickly Create Records Using Global Actions

Global actions are for when you want to create something quickly from pretty much anywhere in the app. For example, imagine one of your users works at a trade show and meets new people all day long. She needs a way to quickly add someone as a contact without navigating to a record or associating this person with any other information. That's what a global action is for: creating quick records that they can follow up with later.

You can include global actions on page layouts for any supported object, and on global publisher layouts, too. In effect, this means you can use a global action from anywhere.

The overall steps for creating a global action are:

1. Create the global action.
2. Choose which fields users see, and if possible predefine required field values.
3. Add the action to the global page layout.

Step 1: Create a Global Action

A global action can appear anywhere with a global publisher layout, so it's useful for things that need to be done quickly, but not necessarily completely.

1. In Setup, enter `Actions` in the `Quick Find` box, then select **Global Actions**.

   Notice there are already a number of actions to choose from. You've seen some of these already in Salesforce1.

2. Click **New Action**.
3. For Action Type, leave Create a Record selected.
4. For Target Object, select Merchandise.
5. For `Label`, enter `New Merch`.
6. Click **Save**.

After saving, the action layout editor opens. Typically at this point you'd customize the fields that show up here, but there aren't many fields on this object, so it's not necessary yet. Click **Save**.

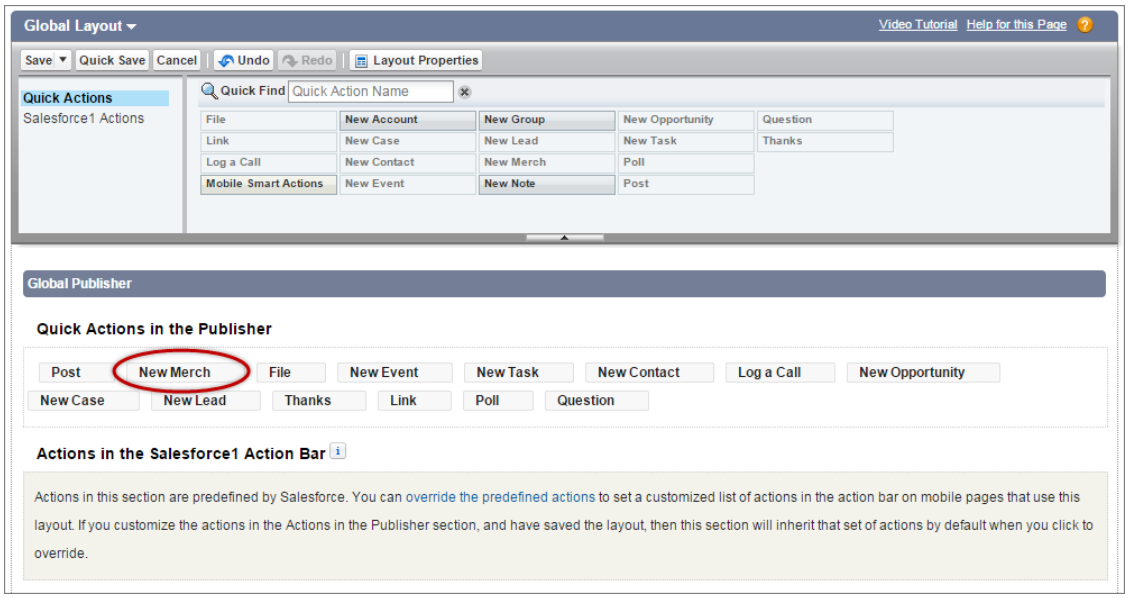
Tell Me More....

- You created a custom label called New Merch, but you can also choose one of the generated labels by choosing from the Standard Label Type drop-down list.
- At the bottom of the global action detail page, there's a section for predefined values. If you predefine a required field, you don't need to display that field on the page. Predefining fields is also a great way to customize the mobile experience, and you'll learn about that in just a bit.

Step 2: Customize the Global Layout

Before the global action will show up in either the full Salesforce site or Salesforce1, you need to add it to the global publisher layout.

1. In Setup, enter *Publisher Layouts* in the Quick Find box, then select **Publisher Layouts**.
2. Next to Global Layout, click **Edit**.
3. In the editor, notice a number of items are in the Quick Actions in the Salesforce Classic Publisher section, such as Post, File, and New Task. Drag the New Merch action into the left side of the Quick Actions in the Salesforce Classic Publisher section, between Post and File.



4. Click **Quick Save**.
5. In the Salesforce1 and Lightning Experience Actions section, click **override the predefined actions**.
6. Click the **Salesforce1 Actions** category in the palette, and then drag **New Merch** into the Salesforce1 and Lightning Experience Actions section so that it's the second item in the list.
7. Click **Save**.
8. Try it out by opening Salesforce1. You see the **New Merch** action in the action bar.

Tell Me More....

- In Salesforce1, global actions appear in the action bar on pages to which the global publisher layout applies, such as the feed, groups, and layouts that haven't been overridden by another publisher layout.
- If you had placed the New Merch action first in the list, anyone using the full Salesforce site would see the expanded list of fields for that action every time they opened Chatter. That could be an annoying use of space! So it's better to locate actions that require a lot of fields somewhere further down the line.
- Just as you can with regular page layouts, you can assign global publisher layouts to different user profiles. This lets different types of users have different global actions.

Create Related Records with Object-specific Actions

Object-specific actions let users create records that are automatically associated with related records. The Warehouse app currently doesn't have a good use case for an object-specific action, so this example uses the Account and Case standard objects, which come in every Developer Edition organization.

In this example, a mobile technician might want a way to create a new case while still on site with a customer. If you add a record create action to the Account object with Case as the target object, the technicians can browse to the customer account record on their mobile device, and log cases directly from there.

The overall steps for creating an object-specific action are:

1. Create the object-specific action.
2. Choose which fields users see. Predefine required field values where possible.
3. Add the action to one or more of that object's page layout.

Step 1: Define an Object-Specific Action

For this scenario, you create an invoice that's associated with an existing account.

1. From Setup, enter *Accounts* in the Quick Find box, then select **Buttons, Links, and Actions**.
2. Click **New Action**.
3. For Action Type, leave Create a Record selected.
4. For Target Object, select Case.
5. For Label, enter *Create a Case*, and then click **Save**.

   The action layout editor opens, which is where you can customize the fields assigned to the action.
6. Remove the Status field from the layout by dragging it into the palette, and then click **Save**.
7. You get a warning message about a required field. Click **Yes**, because you'll fix that next.

Tell Me More....

You just dragged a required field off the page layout. The platform gives you a warning message, and as well it should, users won't be able to create a case from the mobile action! The reason for removing that field will become clear in the next step, when you predefine that field's value.

Step 2: Choose Fields and Predefine Field Values

Objects can have many fields, and so when a user creates a record for that object, it can result in a long list that takes up the screen space and time that mobile users don't have. So it's important to choose which fields show up on the action layout. Additionally, you can predefined field values, and then remove them from the action layout.

For this example, mobile technicians are already on site logging the case. Rather than require them to choose a status every time they create an case, you can predefine the field value. Then you can remove the required field from the action layout. Whenever the Create a Case action is used, the status will automatically be set.

1. From Setup, enter *Accounts* in the Quick Find box, then select **Buttons, Fields, and Actions**.
2. Click the **Create a Case** action you just created.
3. In the Predefined Values related list, click **New**.
4. From the Field Name drop-down list, select Status.
5. Set its specific value to Working, and then click **Save**.

Tell Me More....

Note that predefined values override default values. In the previous example, imagine that cases created on the full Salesforce site are typically new, and so whenever a case is created there, the default value is set to "Open". But when a new case is created from a mobile device, it's because there's a mobile technician on site, and they are actually working on that case. New cases logged from the mobile device overrides the default value and predefines it as "Working". As you can see, not only do predefined field values free up screen space, they can also be used to optimize for what people do when they are mobile.

Step 3: Customize an Object-Specific Layout

Before the action will show up in either the full Salesforce site or Salesforce1, it needs to be added to a page layout.

1. From Setup, enter *Accounts* in the Quick Find box, then select **Page Layouts**.
2. Next to Account Mobile Layout, click **Edit**.

   This layout is the one you created earlier. Notice that the Salesforce1 and Lightning Experience Actions section is empty, and a message is telling you that actions on this layout are predefined by Salesforce. You don't want that. You want to customize the actions on this layout to be pertinent to the work the mobile users need to do.

3. In the Salesforce1 and Lightning Experience Actions section, click **override the predefined actions**.
4. Click the **Salesforce1 Actions** category in the palette, and then drag **Create a Case** so that it's the second item in the list.

   A **New Case** item is also in the palette. The **New Case** item is a default action assigned to the Account object, but it's not editable. You don't want this default action, because you created a custom Create a Case action.

5. Click **Save**. The new Create a Case action now shows up in the action bar on the Account record pages in Salesforce1 for all mobile technicians.
6. Test it on your mobile device by navigating to an account.
7. On the detail page for an account, tap the **Create a Case** action.

You don't see the required Status field for the case, but it's there, and so is the association to this particular account.

Tell Me More....

When you create object-specific layouts, keep in mind that only the first four actions in the list appear on the action bar in Salesforce1. The rest of the actions are accessible from the action menu when users tap  in the action bar.

SECURE YOUR SYSTEM

Level: Beginner; **Duration:** 35–40 minutes

The platform makes it easy for you to implement a security policy of least privilege for all types of users. Effectively, each user should only have the privileges they need to get the job done. Every organization is locked down tightly when you first provision it. These tutorials teach you how to use various features such as users, profiles, permission sets, and roles to progressively open up access so that just the right users have access to just the right data at just the right time.

Here's a preview of how it's done.

1. **Create profiles and permission sets** — Identify the different types of users you need for your application, based on the different functions each type needs to access. Create a base level profile for each type of user such that each profile has only the permissions required for that type of user to perform these functions. Then create permission sets to handle exceptions—situations in which a user may need a few more permissions.
2. **Create users** — For each person who needs app and database access, create a user, assigning the user to the appropriate profile and permission sets.
3. **Set sharing models** — For each object, set the organization-wide default record sharing model to determine whether the records that each user owns are public or private.
4. **Share private records** — Use roles, groups, record sharing rules, and other means to share private records with other users.

Prerequisites

Browser Switching

This tutorial requires you to switch between users. To do that, it's easier to leave one browser open as your admin/developer (the login you've been using so far), and use a different browser for other users. For example, if you are using Safari for your admin/developer login, use a different browser such as Mozilla Firefox for the Manager and Salesperson users. That way, you can simply switch between different browsers to configure security and then test record access without having to log out and log in repeatedly. If you're using Google Chrome, you can also use Chrome *incognito* to log in as multiple users in the same browser.

Create a Profile and Permission Set

Duration: 5–10 minutes

Before creating users, it's best to create one or more profiles and permission sets. Profiles and permissions sets are collections of functional permissions and settings that control what a user can do. For example, profiles and permission sets control:

- System-level access, including time- and IP-based login restrictions as well as permissions that apply to different functions within an organization such as the ability to manage users
- Object-level access, including permissions to create, update, and delete records for each object in the database

- Field-level access, including the ability to read or edit fields in objects
- Access to invoke Apex classes and custom logic

So what's the difference between a profile and a permission set? Users can have exactly one profile, but could have a number of permission sets. Here's how that might work; suppose you need to implement a security policy that has many types of users with similar yet varying privilege requirements. Rather than building and managing many profiles that differ only slightly, it's more efficient to build one profile to manage the common permissions and then use permission sets to manage other specific sets of permissions.

Note: Before you get started creating profiles and permission sets, be aware that the available permissions you can configure for a profile or permission set depend on the user license you associate with it.

Step 1: Create a Profile

Complete the following steps to create a base profile for the Warehouse app:

1. From Setup, enter *Profiles* in the Quick Find box, then select **Profiles**.
2. Next to **Standard Platform User** click **Clone**.
3. Name the new profile *Warehouse App User*, then click **Save**.

Tell Me More....

The profile you clone is important to consider because it determines what type of user license to use. For example, in a DE org, you have three Salesforce Platform User licenses that these tutorials intend to use.

Step 2: Edit a Profile

Now edit the new profile so that it delivers the common permissions that all types of Warehouse app users need to do their work. Specifically, every Warehouse app user needs to be able to:

- Switch to the Warehouse app
- See the Invoices tab, but not necessarily the Merchandise tab
- Read Merchandise records because Merchandise is a lookup object

Complete the following steps to create the baseline profile for Warehouse app users:

1. On the Warehouse App User detail page, click **Edit**.
2. In the Custom App Settings section, select Visible and Default for the Warehouse app.
3. In the Tab Settings section, set Invoices to Default On and Merchandise to Tab Hidden.
4. In the Custom Object Permissions section, enable Read for the Merchandise object (see the following image) and then click **Save**.

Step 3: Create the Manager Permission Set

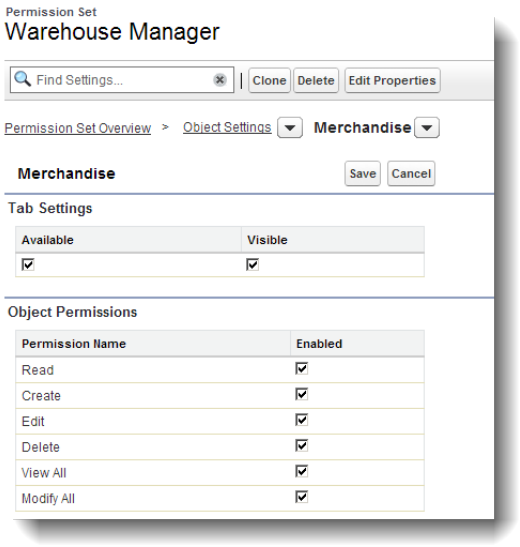
In this step, you are going to configure security for two different types of Warehouse app users: managers and sales people. Both types of users can use the Warehouse App profile for their base permissions, but need different privileges thereafter. To handle this requirement, create two different permission sets.

Use the following steps to create the Warehouse Manager permission set:

1. From Setup, enter `Permission Sets` in the `Quick Find` box, then select **Permission Sets** and then **New**.
2. For `Label`, enter `Warehouse Manager`.
3. For User License, select Salesforce Platform and click **Save**.

Now modify the new permission set so that it provides access to create, update, and delete Merchandise object records, and view the Merchandise tab.

1. From the Warehouse Manager permission set detail page, click **Object Settings**.
2. Click **Merchandise**.
3. Click **Edit**.
4. In Tab Settings, enable `Available` and `Visible`.
5. In Object Permissions, enable all permissions.
6. Click **Save**.



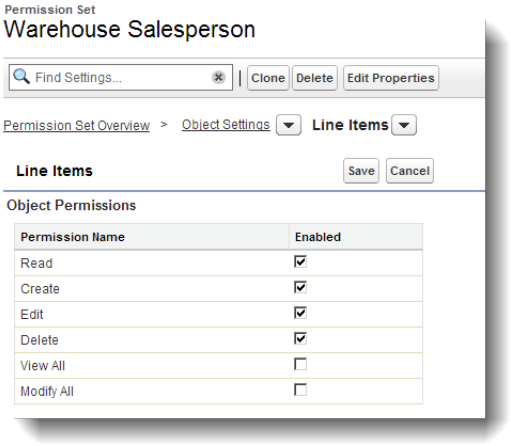
Step 4: Create the Salesperson Permission Set

Use these steps to create the Warehouse Salesperson permission set:

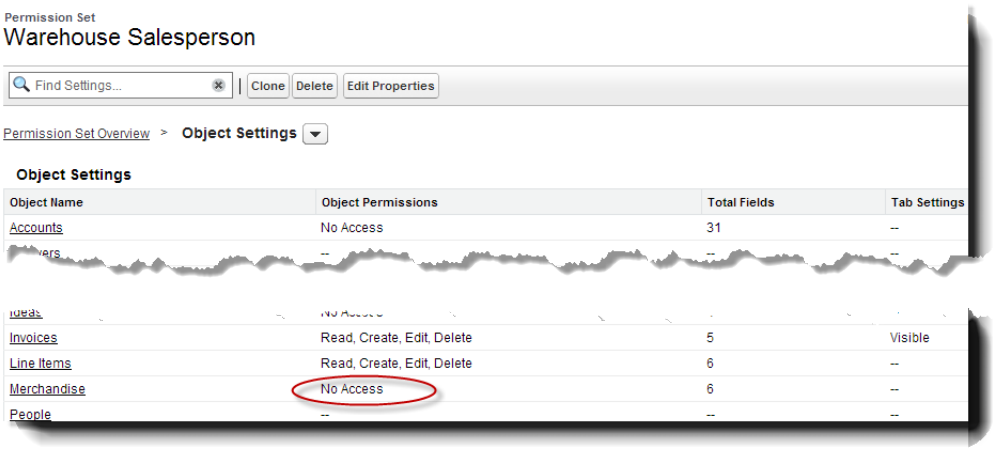
1. From Setup, enter `Permission Sets` in the `Quick Find` box, then select **Permission Sets** and then **New**.
2. For `Label`, enter `Warehouse Salesperson`.
3. For User License, select Salesforce Platform and click **Save**.

Now modify the new permission set so that it provides access to create, update, and delete Invoice and Line Item object records, and view the Invoices tab.

1. From the Warehouse Salesperson permission set detail page, click **Object Settings**.
2. Click **Invoices** and then **Edit**.
3. In Tab Settings, enable `Available` and `Visible`.
4. In Object Permissions, enable these permissions: `Read`, `Create`, `Edit`, and `Delete`.
5. Click **Save**.
6. In the breadcrumb menu, click **Object Settings**.
7. Click **Line Items**.
8. Click **Edit**.
9. In Object Permissions, enable the following permissions: `Read`, `Create`, `Edit`, and `Delete`.
10. Click **Save**.



The Warehouse Salesperson permission set doesn't give access to Merchandise, only Invoices and Line Items. You can see this on the Object Settings page for the permission set.



Create New Users

Duration: 5–10 minutes

Once you have profiles and permission sets in place, you can turn your attention to users. Every new org starts with a super-user administrator that can access and customize everything in the organization, including profiles, permission sets, and other users. You happen to be logged in as that super-user right now. Because you don't want everyone to have that kind of power and access, you'll want to restrict what people can do.

In this tutorial you create two new users that represent people that work in the warehouse: a manager and a salesperson. Yes, these are the same names for the profiles and permission sets you created earlier, but now you'll assign them to people.

Step 1: Create New Users

Use the following steps to create a new user that serves a "sales manager." In the following steps, make sure to use an email address that you can access immediately:

1. In Setup, enter *Users* in the Quick Find box, then select **Users**.
2. Click **New User**.
3. Fill out the form as follows:
   - First Name: *Sales*
   - Last Name: *Manager*
   - Email: *enter your email address*
   - Username: *your username.manager@your domain*
   - Nickname: *your username.manager*
   - Role: Leave this field blank for now, you'll assign roles later.
   - User License: *Salesforce Platform*
   - Profile: *Warehouse App User*
   - At the very bottom, clear the checkboxes for the newsletters, but make sure Generate new password and notify user immediately: is checked.
4. Click **Save**.

Repeat the process to create a new user that serves as a "salesperson," with the following exceptions:

- First Name: *Sales*
- Last Name: *Person*
- Email: *enter your email address*
- Username: *your username.sales@your domain*
- Nickname: *your username.sales*
- Role: Leave this field blank for now, you'll assign roles later.
- User License: *Salesforce Platform*
- Profile: *Warehouse App User*
- At the very bottom, clear the checkboxes for the newsletters, but make sure Generate new password and notify user immediately: is checked.

You now have two users, both using the Warehouse App User profile. Also, you should have two emails in your email inbox: activation emails for each new user.

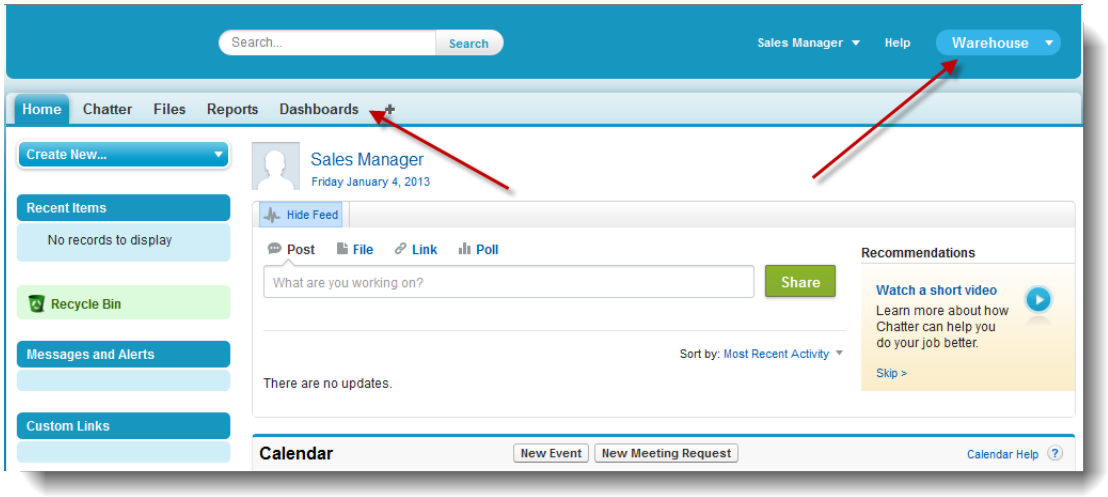
Step 2: Test Record Access

The Warehouse App User profile is assigned to both of these new users, so while they can log into the DE org and start the Warehouse app, they can't do much more. First, you'll log in as the Sales Manager.

1. You should have an email in your inbox, click the link to log in as the Sales Manager.

   Note: This is a good time to switch between browsers, as noted in the Prerequisites.

2. Change your password and then you should see the Home tab.
3. Notice that the default app is Warehouse (if you don't see Warehouse that's OK, you just missed that setting in the profile, either edit the profile or select the Warehouse app now), but that you can't see the Merchandise or Invoices tabs. Why not? Because the user's profile doesn't provide the permissions necessary to access to the underlying objects that power the app.



Step 3: Assign Permission Sets to Users

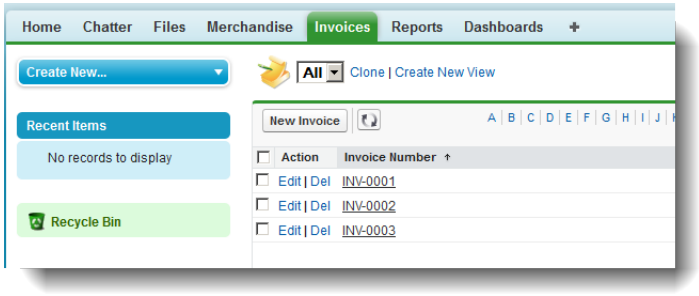
To give the Sales Manager and Sales Person users access to the permissions they require, simply update each user with the appropriate permission set.

1. Switch back to the browser with your administrator login.
2. In Setup, enter *Users* in the `Quick Find` box, then select **Users**.
3. Click **Manager, Sales** to go to this user's detail page.
4. In the Permission Set Assignments section, click **Edit Assignments**.
5. Add both the Warehouse Merchandise Manager and Warehouse Sales Person permission sets to the user's list of Enabled Permission Sets and click **Save**.
6. Repeat the previous steps for the **Person, Sales** user, but this time, add *only* the Warehouse Sales Person permission set to the user's list of permission sets.

Step 4: Test Record Access

Now it's time to see the effects of adding the permission sets to the two users.

1. Switch back to your other browser that's already logged into the DE org as the Sales Manager, refresh the page, and notice that the Merchandise and Invoices tabs are now available.
2. Click on the Merchandise tab.
3. Click **Go!** next to View: All to display all records.
4. Click on the Invoices tab and check those out, too.



5. Now log out and, using the activation link in your email, log in as the Salesperson (and change your password).
6. Confirm that the Salesperson can see the Invoices tab, but not the Merchandise tab, as governed by the user's permission sets.

Tell Me More

As you can see, it's pretty easy to create profiles and permission sets and then assign them to different users.

Configure Org-Wide Defaults

Duration: 5–10 minutes

Inherent in the design of the platform's security model is the notion of *record ownership*, which helps to simplify the implementation of row-level least-privilege data security policies. The creator of a record *owns* the record after creation and has full access — the owner can read, update, delete, and transfer ownership for the record.

Various data access controls determine whether org users can access records they don't own. These controls include an object's sharing model, role hierarchies, groups, and sharing rules.

To begin, each object has a sharing model, also known as an organization-wide default (OWD), which governs the default org-wide access levels users have to each other's records in the object.

- With an object that uses a private sharing model, the record owner can read and manage a record, assuming that the user's profile provides object-level access. Other users can work with records they don't own only by other record sharing means.
- With an object that uses a public read-only sharing model, any user can read all records in the object, assuming that the user's profile provides the Read permission and field-level access for the object.
- With an object that uses a public read/write sharing model, any user can read and write all records in the object, as permitted by the object- and field-level permissions in each user's profile.

An object can have different sharing requirements based on the user context, so it's very important to consider this fact when setting its OWD. A good rule of thumb is to set each object's OWD to be as strict as necessary for the most strict user requirement, and then use sharing rules to open up access, as required.

So why can the Salesperson user see all Invoices and Line Items? To answer this, investigate the OWDs for these objects.

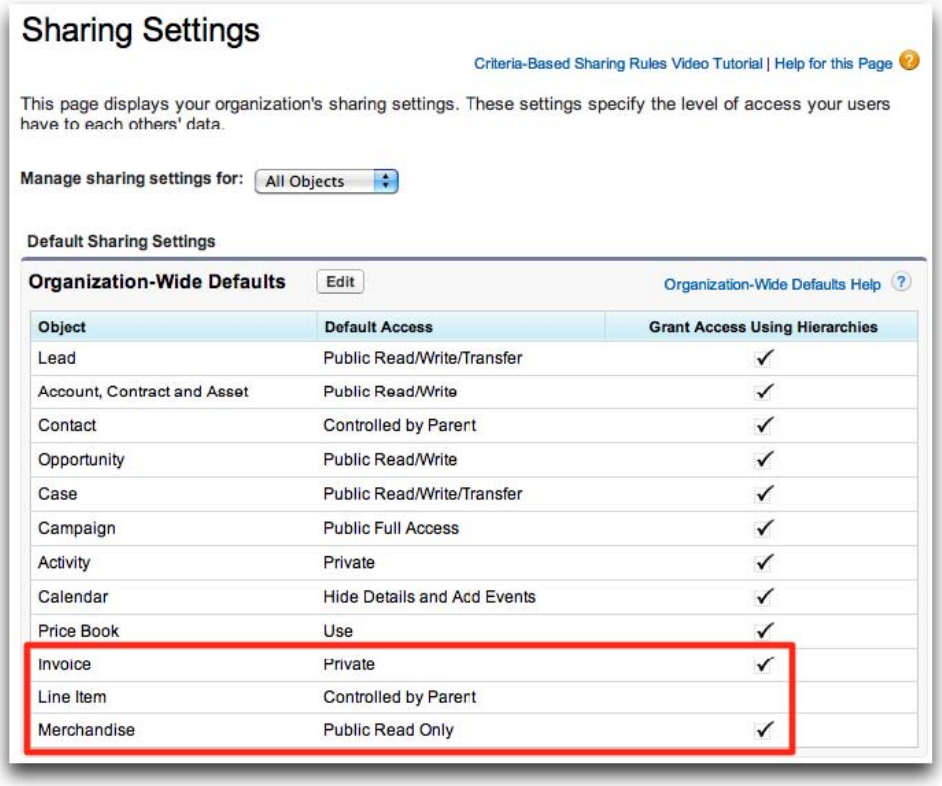
Step 1: Modify the OWD for Invoices

Complete the following steps to view the OWD for Invoices.

1. Switch back to the browser with your super-user admin login.
2. From Setup, enter *Sharing Settings* in the Quick Find box, then select **Sharing Settings**.

Before continuing, notice that the OWD for both Invoice and Merchandise is set to Public Read/Write, which allows every logged–in user to read, create, update, and delete any record in these objects no matter who owns the record. Now change the OWD for both objects:

1. In the Organization-Wide Defaults section, click **Edit**.
2. For Invoice, select Private, and select Grant Access Using Hierarchies.
3. For Merchandise, select Public Read Only, and select Grant Access Using Hierarchies.
4. Click **Save**.



Tell Me More....

What about the Line Items object? Before you leave this page, notice that the Line Item object's OWD is "Controlled by Parent", which means it inherits the OWD of the parent Invoice object. This happened automatically because you of the master-detail relationship between the two objects. Neat, huh?

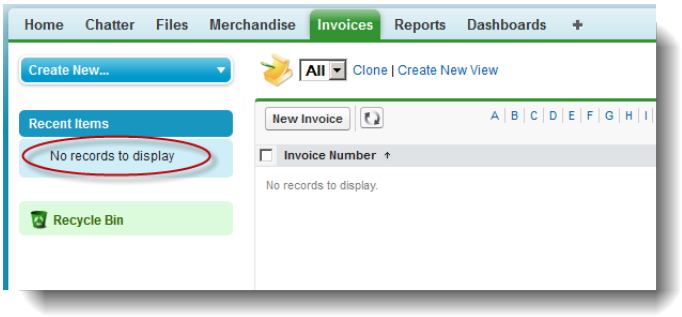
Step 2: Test Record Access

To see the effects of changing the OWD for Invoice, complete the following steps as a Sales Person:

1. Switch back to the browser that's logged into the DE org as the Sales Person, then click the Invoices tab.
2. Click **Go!** next to View: All.
3. Notice that the list of available invoices is empty.

Why did this change? Although Sales Person has a permission set that lets the user CRUD Invoice and Line Item records, this permission only provides the ability to CRUD records that the user *owns*. Considering that the OWD for these objects is set to Private, and the only records created were by the admin super-user, Sales Person can't see that owner's records. To prove that Sales Person can access only records that the user owns, complete the following steps to create a new Invoice that Sales Person owns:

1. On the Invoice detail page, click **New** , then click **Save**.
2. Click **New Line Item** and add choose some Merchandise. Click **Save**.
3. Click the Invoices tab and you'll see there's now an invoice there.
4. Now log out, and log back in as the Sales Manager user.
5. Repeat the steps above to prove that you cannot access Invoices and Line Items in the system that the Sales Manager user does not own due to the Private setting for these objects.



Share Records Using a Role Hierarchy

Duration: 5–10 minutes

In the last tutorial you saw that private record access can get in the way of managers seeing what their employees are up to. You need to open up that record access to managers, but not necessarily *all* managers. Ideally managers should be able to see all invoices owned by salespeople that they manage. In this tutorial, you learn how to set up and use a role hierarchy to automatically open up private records in an organization's org chart.

Step 1: Create a Role Hierarchy

To create a role hierarchy:

1. Switch back to the browser with your administrator login.
2. From Setup, enter *Roles* in the Quick Find box, then select **Roles**.

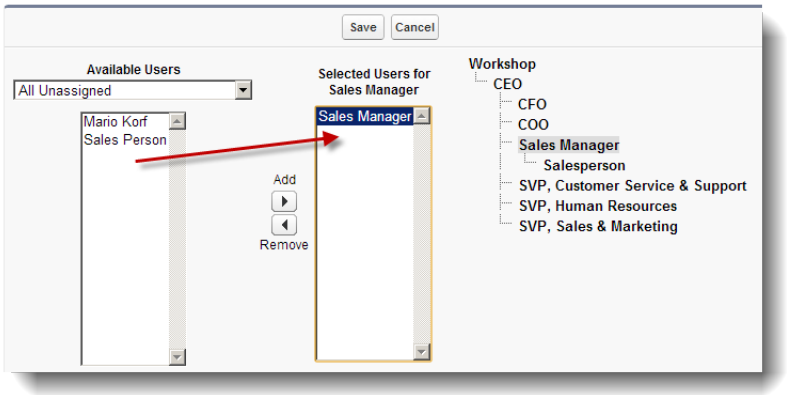
   Notice there's a drop-down list of sample role hierarchies you can choose. Click through the options and notice the differences.
3. Chose Territory-based Sample and click **Set Up Roles**.
4. Under CEO, click **Add Role**.
5. For Label, enter *Sales Manager* and click **Save & New**.
6. For Label, enter *Salesperson*.
7. For This role reports to, use the lookup to select **Sales Manager**.
8. Click **Save**.
9. Now go back to the Creating the Role Hierarchy page. Expand the node for Sales Manager, and you can see the subordinate Salesperson role.

Tell Me More....

There are a lot of extra roles defined based on the sample template you started with. You can delete them if you want, it won't make any difference for this set of tutorials. Note that DE orgs come only with two users, so unfortunately you can't continue to add users.

Step 2: Assign Users to Roles

1. If you're not on the Creating the Role Hierarchy page, from Setup, enter *Roles* in the Quick Find box, select **Roles**, and then click **Set Up Roles**.
2. Next to Sales Manager role, click **Assign**.
3. Add Sales Manager to the Selected list, then click **Save**.



4. Repeat the process to assign the Sales Person user to the Salesperson role.

Step 3: Test Record Access

Again, it's time to test the effects of your most recent security configuration changes.

1. Switch back to the browser that's logged in as the Sales Manager, then click the Invoices tab.
2. Click **Go!** next to View: All.
3. Notice that the Sales Manager user can now work with the invoice owned by the Sales Person user. That's because the role hierarchy shares private records up the role hierarchy.

Tell Me More....

A role hierarchy is just one option for sharing access to private records. For example, organizations often need to share sets of private records that are related by ownership or other criteria with particular users. For such requirements, you can use *groups*. All that you need to do is create a group and your *sharing rules* using a few more clicks.

CODE CUSTOM APP LOGIC

To quickly build apps that are easy to maintain, use the platform's built-in, point-and-click options for business logic whenever possible. Sometimes though, features such as workflow rules, formula fields, rollup summary fields, and approvals can't meet all of your needs — that's when you should consider coding app logic.

In this series of tutorials, you learn how to use Apex and code custom app logic that meets unique requirements for app logic. *Apex* is the platform's programming language that you can use to build things like stored procedures and database triggers that are common in traditional database-driven application development platforms. Along the way, you'll learn how to use several tools to develop Apex classes, methods, database triggers, and unit tests.

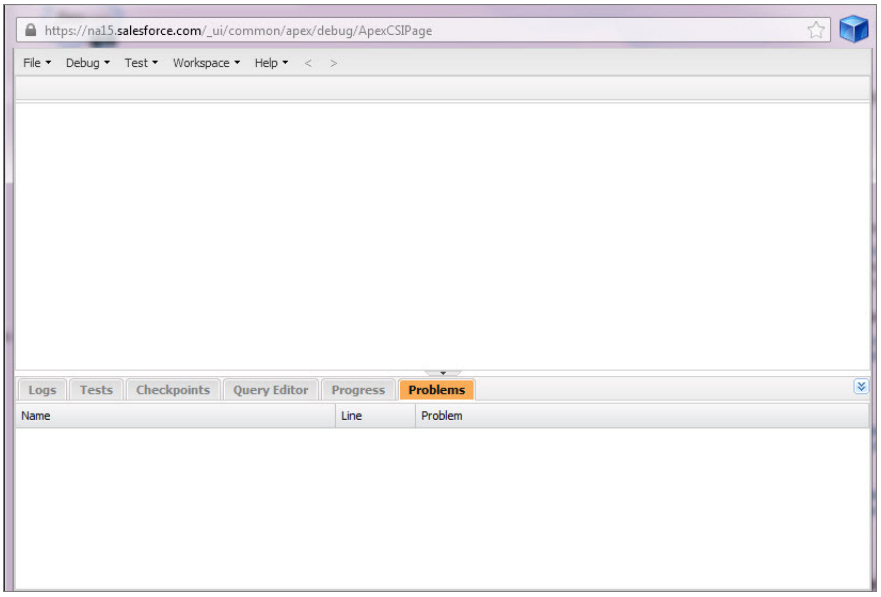
Explore the Developer Console and Apex

In this tutorial, you get a first look at Apex using the Developer Console.

Step 1: Start the Developer Console

There are several tools that you can use for code. This tutorial gets you started with Apex language fundamentals using one such tool, the Developer Console, which is part of the browser-based development environment.

Log into your DE org open the Developer Console under `Your Name` or the quick access menu ( ).



Tell Me More....

Notice the **Help** link click at the top of the of the console? If you click the link you'll find a bunch of really great resources. If you don't have time now, check it out later.

Step 2: Execute Basic Apex Code

Now it's time for you to dive into Apex. Use the console to execute a few lines of Apex code.

1. Click **Debug** > **Open Execute Anonymous Window**.
2. In the Enter Apex Code window, enter the following code.

```
for (Integer i=1; i<=10; i++) {
   System.debug('Hi ' + i); // output "Hi" to the debug log
}
```

3. Select the checkbox for `Open Log` and then click **Execute**.

Tell Me More....

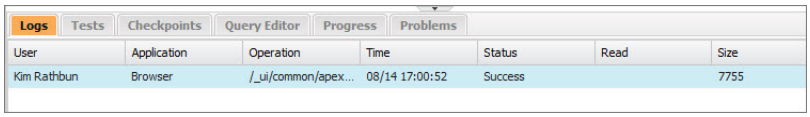
Apex is a programming language that you can learn quickly, especially if you already know similar languages, such as Java, C++, or C#. Although the example above is extremely simple, you can learn a lot about Apex by studying it closely.

- Notice that it is a strongly typed language that supports common language fundamentals, such as variable declarations, assignments, flow control structures such as loops, string concatenation, and comments. It's also an object-oriented language, as the call to the debug method of the standard System class illustrates.
- If you're wondering why it is called *anonymous* code, that's because you are not naming and saving the code for later reuse — you simply execute it, and once you leave the console and clear your work, it's gone. So where's your output? Continue to the next step.

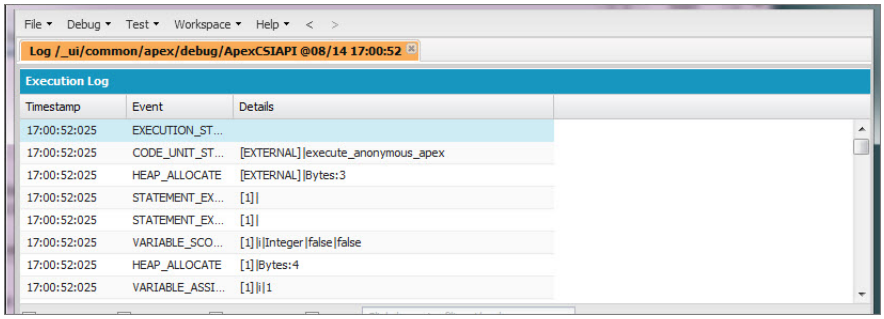
Step 3: Review the Execution Log

Each time you execute some code in the console, you produce a log that contains a series of records that detail what happened during the code execution.

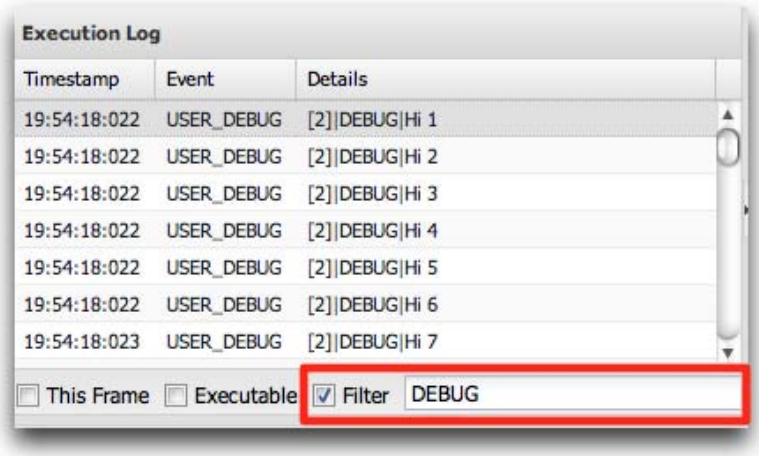
1. Click the Logs tab in the lower portion of the console. A sortable list of the most recent execution logs displays.



2. To look at the records in a specific log, double-click the log of interest. This action creates a new Log tab below with corresponding Stack, Execution Log, and Source and Variables section.



3. In the Execution Log section, enter *DEBUG* (all caps) next to the `Filter` checkbox. Notice that the log only shows records corresponding to `System.debug` calls, which verifies the output of "Hi" plus the value of the loop counter variable as it iterates.



Tell Me More....

There's a lot of information in a log. For example, in the Stack section, the Execution Tree tab shows a hierarchical tree of execution operations, while the Performance Tree tab shows aggregated operation performance data that you can use to diagnose performance issues. The Execution Log section shows individual durations and log records for the log that you select above in the Logs tab. Execution logs can have many records, so it's useful to filter noise out and focus on just what interests you.

Feel free to explore the Developer Console and experiment. Remember that it's always there when you need to quickly test, tune, or debug some Apex code. For more information about the Developer Console, click **Help** to open Help and Training.

Create an Apex Class and Method

Level: Intermediate; **Duration:** 20-30 minutes

In this tutorial, you learn how to create persistent Apex classes with named methods, what some app developers might think of as database-stored procedures.

The Warehouse app currently requires that the user manually enter a line item number for each line item in an invoice. This is not optimal as it can lead to strange sequences of numbers when people are not careful and when records get deleted.

Sometimes you run into situations where you can't solve the problem using declarative tools such as workflow rules, so this tutorial shows you how to build an Apex class method that automatically renumbers all line items for a given invoice. The goal is to make sure that every invoice has a collection of line items that starts with the number 1 and increments by 1 with no gaps (1, 2, 3, ...).

Step 1: Create an Apex Class

An *Apex class* is an encapsulation of related variables, constants, and methods, stored centrally on the platform that your app can use to process work.

To create an Apex class using the Developer Console:

1. In your DE org, open the Developer Console under `Your Name` or the quick access menu ().
2. Click **File** > **New** > **Apex Class**.
3. Name the new class *`InvoiceUtilities`*, then click **OK**.
4. The default Apex class template creates the new class with the following template code.

```
public class InvoiceUtilities {
}
```

5. Comment the code as follows, and then click **File** > **Save**.

```
public class InvoiceUtilities {
   // Class method to renumber Line Items for a given Invoice number.
   // Returns a string that indicates success or failure.
}
```

Step 2: Create a Blueprint Class Method

The Warehouse app currently requires that the user manually enter a line item number for each line item in an invoice. This is not optimal as it can lead to strange sequences of numbers when people are not careful and when records get deleted.

Unfortunately, there's no way to solve this problem using declarative tools such as workflow rules, so this tutorial shows you how to build an Apex class method that automatically renumbers all line items for a given invoice. The goal is to make sure that every invoice has a collection of line items that start with the number 1 and increment by 1 with no gaps (1, 2, 3, ...).

The first thing you might do is determine the basics for the class method you want to build: its name, the parameters it accepts, what values it returns to the calling environment, and perhaps some pseudo code to outline your plan of attack.

- In the console, modify your Apex class to match the following code.
- Don't save or you'll get a compilation error because we haven't added a return statement yet.

For source code, see `https://gist.github.com/3605633`.

```
public class InvoiceUtilities {
   // Class method to renumber Line Items for a given Invoice number.
   // Returns a string that indicates success or failure.
   public static String renumberLineItems(String invoiceName) {

      // Create a copy of the target Invoice object and its Line Items.

      // Loop through each Line Item, re-numbering as you go

      // Update the Line Items in one transaction, rollback if any problems
```

```
        // and return error messages to the calling environment.

        // On success, return a message to the calling program.
    }
}
```

Step 3: Get an Invoice and its Line Items

Now that you have a plan, start filling out the code beneath your comments. Start by creating a local copy of the target invoice and its line items.

1. Beneath the method declaration and first comment, enter the following in the method.

```
Invoice__c invoice =
```

2. Now use a SOQL query that orders existing line items and uses a filter to retrieve the target invoice, as given by the method's input parameter. Notice that the object notation in SOQL is somewhat unique.

```
Invoice__c invoice = [Select i.Name, (Select Name From Line_Items__r ORDER BY Name)
                                      From Invoice__c i
                                      Where i.Name = :invoiceName LIMIT 1];
```

3. Don't save yet or you'll get a compilation error because we haven't added a return statement yet.

Step 4: Create the Final Version of the Class Method

Now that you have a plan, start filling out the pseudo code to build the final class method logic. Start by creating a local sObject copy of the target invoice and its line items (see lines 8-10 below). The method code includes a SOQL query that orders existing line items (see line 8) and uses a filter to retrieve the target invoice, as given by the method's input parameter (see line 10). Notice that the object notation in SOQL is somewhat unique.

Note: Remember to **Save** your class as you go along in this step. On each save operation, make sure to check the Problems pane and confirm that you don't have any compilation errors.

For source code, see https://gist.github.com/3605645.

```
public class InvoiceUtilities {
    // Class method to renumber Line Items for a given Invoice number.
    // Returns a string that indicates success or failure.
    public static String renumberLineItems(String invoiceName) {

        // Create a copy of the target Invoice object and its Line Items.
        Invoice__c invoice =
            [SELECT i.Name, (Select Name FROM Line_Items__r ORDER BY Name)
             FROM Invoice__c i
             WHERE i.Name = :invoiceName LIMIT 1];

        // Loop through each Line Item, renumbering as you go.

        // Update the Line Items in one transaction, rollback if any problems
        // and return error messages to the calling environment.

        // On success, return a message to the calling program.
        return 'Line items renumbered successfully.';
```

```
    }
}
```

Save the updated Apex class. On each save operation, check the Problems pane and confirm that you don't have any compilation errors. If you do, fix them appropriately and save the corrected code.

Next, update the class with a loop to process and renumber each line item (see lines 13-18).

For source code, https://gist.github.com/3605650.

```
public class InvoiceUtilities {

   // Class method to renumber Line Items for a given Invoice number.
   // Returns a string that indicates success or failure.
   public static String renumberLineItems(String invoiceName) {

      // Create a copy of the target Invoice object and its Line Items.
      Invoice__c invoice =
         [SELECT i.Name, (SELECT Name FROM Line_Items__r ORDER BY Name)
          FROM Invoice__c i
          WHERE i.Name = :invoiceName LIMIT 1];

      // Loop through each Line Item, renumbering as you go.
      Integer i = 1;
      for (Line_Item__c item : invoice.Line_Items__r) {
         item.Name = String.valueOf(i);
         System.debug(item.Name);
         i++;
      }

      // Update the Line Items in one transaction, rollback if any problems,
      // and return error messages to the calling environment.

      // On success, return a message to the calling program.
      return 'Line items renumbered successfully.';
   }
}
```

Notice in line 14, the `FOR` loop uses Apex-specific object notation to reference the line items of the invoice. Line 18 includes a System.debug statement to output some handy information to the debug log.

Now create the final version of the class method so that it updates the database with the new version of the invoice's line items (see lines 22-30).

For source code, see https://gist.github.com/3605654.

```
public class InvoiceUtilities {

   // Class method to renumber Line Items for a given Invoice number.
   // Returns a string that indicates success or failure.
   public static String renumberLineItems(String invoiceName) {

      // Create a copy of the target Invoice object and its Line Items.
      Invoice__c invoice =
         [SELECT i.Name, (SELECT Name FROM Line_Items__r ORDER BY Name)
          FROM Invoice__c i
          WHERE i.Name = :invoiceName LIMIT 1];
```

```
        // Loop through each Line Item, renumbering as you go.
        Integer i = 1;
        for (Line_Item__c item : invoice.Line_Items__r) {
            item.Name = String.valueOf(i);
            System.debug(item.Name);
            i++;
        }

        // Update the Line Items in one transaction, rollback if any problems
        // and return error messages to the calling environment.
        try {
            Database.update(invoice.Line_Items__r);
        }
        catch (DmlException e) {
            return e.getMessage();
        }

        // On success, return a message to the calling program.
        return 'Line items re-numbered successfully.';
    }
}
```

This method uses try/catch block to update the database and handle any unforeseen runtime exceptions that might occur.

- In the try block (see lines 22-24), the `Database.update` method is a standard Apex method that you can use to update one or more sObjects. Again, notice the object notation to reference the target invoice's related line items (`invoice.Line_Items__r`).
- The catch block (see lines 25-27) catches any `DmlException`. It contains a `return` statement that returns the exception error message to the caller.
- If the method continues past the try/catch block, which means that no exception was thrown and the method didn't return the exception error message, it simply returns a standard message to indicate success (see line 30).

Step 5: Manually Test the Apex Class Method

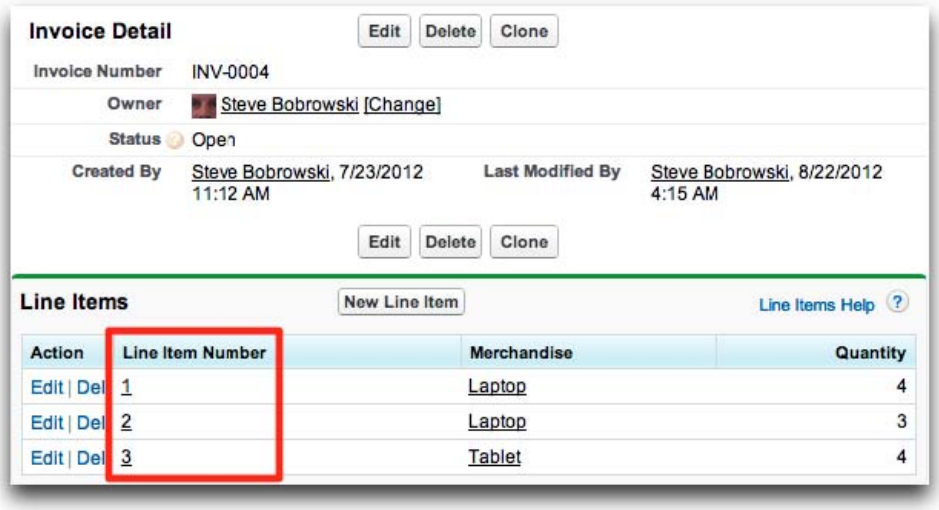
Once you confirm that you can save the Apex class without any errors, it's time to test your new class method.

First, create some test data.

1. In a new browser tab, open up an existing invoice that has some line items. If you don't have any existing invoices, create one. Note the Invoice Number.
2. Update or insert one or more line items so that there is an unwanted sequence of line item numbers (e.g., 1, 3, 6). If you didn't have any test data at the beginning of this step, you may need to create a couple of merchandise records before you can add any line items.
3. Now you can call the method and target the invoice above to renumber its line items.
4. Switch back to the Developer Console. Choose **Debug** > **Open Execute Anonymous Window**.
5. Execute the following anonymous Apex. For the method input parameter, substitute the invoice number that you noted above. For example, if your invoice number is `INV-0000`, substitute that for `INV-0004` in the following code:

```
String s = InvoiceUtilities.renumberLineItems('INV-0004');
```

6. If you switch back to your browser and refresh the Invoice detail page, you should notice that its line items are now in sequence without any gaps.



Congratulations! With less than 20 lines of Apex code, you've built an Apex class method to solve a real-world business requirement.

Tell Me More....

The execution output is interesting to inspect if you want to learn more about Apex code execution. Remember from an earlier tutorial, in the console, you can view logs for code executions. Although a full discussion of the log output for the above class method execution is outside the scope of this tutorial, here are a few highlights.

- Filter for `SOQL_EXECUTE` and you see that the embedded SOQL query retrieved one row from the database.
- Filter for `DEBUG` and you see the output from the `System.debug` calls in the method.

Call an Apex Class Method Using a Button

Level: Advanced; **Duration:** 20-30 minutes

In the previous tutorial, you created an Apex class method that your app can use to renumber an invoice's line items that are out of sequence. But you certainly can't expect users to execute anonymous Apex code to call the method. This tutorial shows you how to create a custom button on the Invoice detail page that calls the method for the current invoice.

Step 1: Create a Custom Button

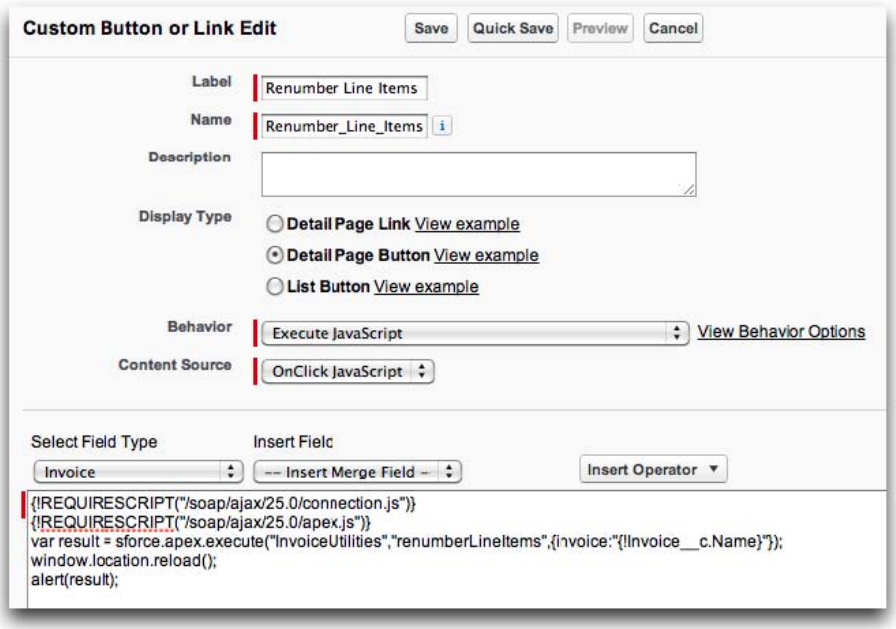
By default, every detail page includes several standard buttons, including Edit, Delete, and Clone. You can also create custom buttons and add them to page layouts as needed. Use the following steps to create a custom button for the Invoice detail page.

1. From Setup in your Developer Edition organization, enter *`Objects`* in the `Quick Find` box, select **Objects**, and then click **Invoice**.
2. Scroll down to the Buttons, Links, and Actions section and click **New Button or Link**.
3. In the `Label` field, enter *`Renumber Line Items`*.
4. For `Display Type` choose `Detail Page Button`.

5. For Behavior choose Execute JavaScript.
6. For Content Source choose OnClick JavaScript.
7. Notice that you are creating a Detail Page button that executes some JavaScript. For your convenience, here is the JavaScript code that you can copy and paste into the form. For source code, see https://gist.github.com/3605659.

```
{!REQUIRESCRIPT("/soap/ajax/29.0/connection.js")}
{!REQUIRESCRIPT("/soap/ajax/29.0/apex.js")}
var result = sforce.apex.execute("InvoiceUtilities","renumberLineItems",{invoiceName:"{!
Invoice__c.Name}"});
alert(result);
window.location.reload();
```

8. Make sure your form matches the following screen, and then click **Save**.
9. When prompted, click **OK**. We'll add this button to a page layout in the next step.



Tell Me More....

Examine the code you pasted.

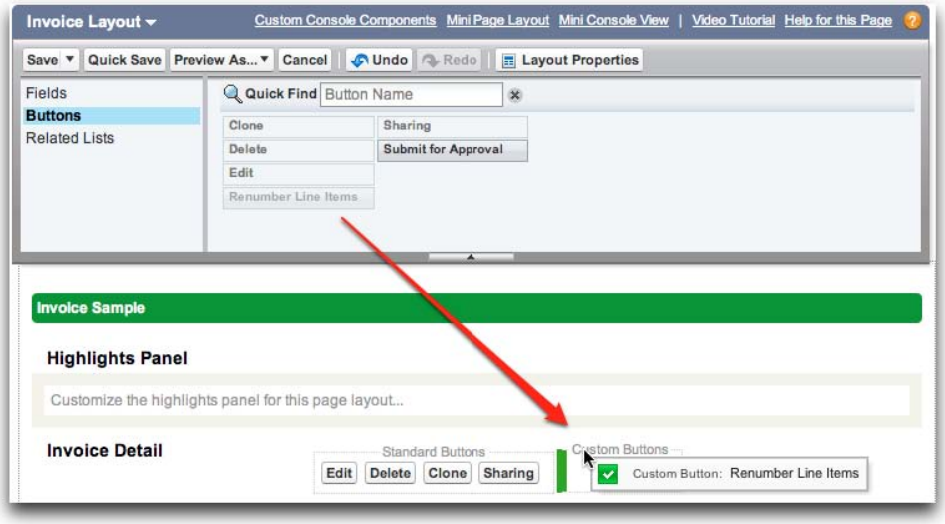
- Lines 1 and 2 load two libraries from the Salesforce AJAX Toolkit, a JavaScript wrapper around the Force.com SOAP API.
- Line 3 leverages the AJAX Toolkit to call the method, passing in the Name of the current invoice.
- Lines 4 and 5 are standard JavaScript calls that display an alert message and refresh the current page.

Step 2: Add the Button to the Page Layout

Next up is to add the new button to the Invoice detail page layout. (Page layout modification is something that this tutorial assumes you already understand, so this step is brief.) To add the custom button to the Invoice page layout:

1. From the Invoice custom object detail page, scroll down to the Page Layouts.
2. Click **Edit** next to the Invoice Layout.

3. In the palette at the top, click **Buttons**.
4. Drag the **Renumber Line Items** button to the Custom Buttons section of the page layout, then click **Save**.



Step 3: Modify the Apex Class

The JavaScript in the custom button leverages the AJAX Toolkit to make SOAP calls from JavaScript. Considering this, there are two minor changes that you need to make to the Apex class and method so that it supports SOAP API calls.

To modify the Apex class without leaving the browser:

1. From Setup, enter "Apex Classes" in the `Quick Find` box, then select **Apex Classes**.
2. Notice next to the `InvoiceUtilities` class there are three links: **Edit, Del,** and **Security**. Click **Edit**.
3. Modify the scope for both the class and the method from public to `global` (line 1) and `webservice` (line 4), then click **Save**.

For source code, see https://gist.github.com/3605663.

Your final code should be as follows:

```
global with sharing class InvoiceUtilities {
   // Class method to renumber Line Items for a given Invoice number.
   // Returns a string that indicates success or failure.
   webservice static String renumberLineItems(String invoiceName) {

      // Create a copy of the target Invoice object and its Line Items.
      Invoice__c invoice =
         [SELECT i.Name, (SELECT Name FROM Line_Items__r ORDER BY Name)
          FROM Invoice__c i
          WHERE i.Name = :invoiceName LIMIT 1];

      // Loop through each Line Item, renumbering as you go.
      Integer i = 1;
      for (Line_Item__c item : invoice.Line_Items__r) {
         item.Name = String.valueOf(i);
         System.debug(item.Name);
         i++;
      }
```

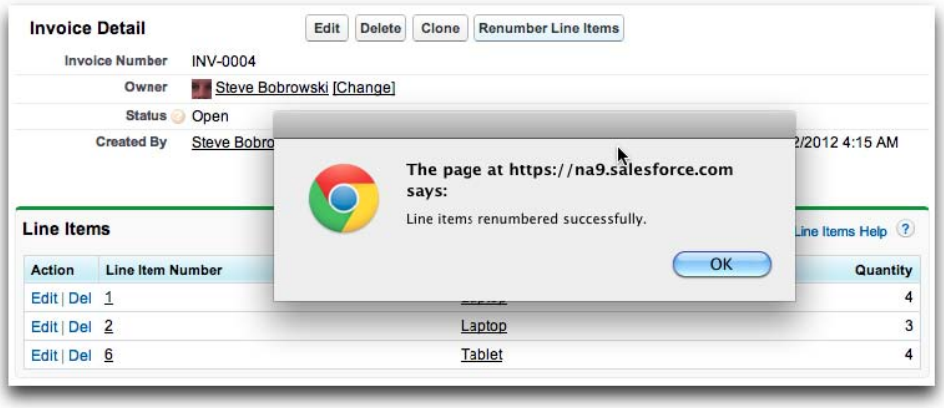
```
        // Update the Line Items in one transaction, rollback if any problems
        // and return error messages to the calling environment.
        try {
            Database.update(invoice.Line_Items__r);
        }
        catch (DmlException e) {
            return e.getMessage();
        }

        // On success, return a message to the calling program.
        return 'Line items renumbered successfully.';
    }
}
```

When you return to the list of Apex classes, notice that there's a new link for the `InvoiceUtilities` class: WSDL. If you click it, you see a WSDL file that apps can use to interface with the class.

Step 4: Test the New Button

Now it's time to test the new button and modified Apex class method.

1. Click the Invoices tab of the Warehouse app.
2. Open any invoice that has line items.
3. Update or insert one or more line items so that there is an unwanted sequence of line item numbers (for example, *1, 3, 6*).
4. Click **Renumber Line Items**. An alert should pop up to indicate success. After you acknowledge the alert, the page refreshes with the updated line items.



Tell Me More....

Another way to call Apex class methods from JavaScript is to use the platform's JavaScript remoting feature. For more information, see the *Apex Developer Guide*.

Create a Database Trigger

Level: Intermediate; **Duration:** 20-30 minutes

Apex triggers are useful for implementing business logic that you can't implement with clicks (such as workflow rules). In this tutorial, the business scenario is this: by default, master-detail relationships automatically cascade the deletion of a master record to all related detail records. Our Warehouse app needs to deviate from this default behavior and employ a trigger that prevents the deletion of Invoices that have Line Items.

Step 1: Create a Database Trigger

To create a trigger on the Invoice object.

1. From Setup, enter *Objects* in the Quick Find box, then select **Objects** and click **Invoice**.
2. Scroll down to Triggers and click **New**.
3. In the editor, replace <name> with *DeleteRestrictInvoice*.
4. Similarly, replace <events> with *before delete*.
5. Replace the template code with the following.

   For source code, see https://gist.github.com/3605667.

```
trigger DeleteRestrictInvoice on Invoice__c (before delete) {

// create a list of Invoices in Trigger.oldMap along with their Line Items
List<Invoice__c> invoices = [Select i.Name, (Select Name From Line_Items__r)
          From Invoice__c i
          Where i.Id IN :Trigger.oldMap.keySet()];

// loop through the Invoices, attaching errors to those that have Line Items
for (Invoice__c invoice : invoices) {
    if (!invoice.Line_Items__r.isEmpty()) {
      Trigger.oldMap.get(invoice.id).addError('Cannot delete Invoice with Line Items');

       }
   }
}
```

6. Click **Save**.

Tell Me More....

- A trigger can fire before or after DML operations. The trigger in this tutorial fires before the execution of a delete operation that targets one or more records in the Invoice object.
- Triggers have special variables accessible to them called context variables. In a nutshell, old and new context variables provide copies of old and new sObjects being updated by the call that fires the trigger. As you can see in the code, context variables are handy to scope processing in a trigger body.
- In the FOR loop, the trigger simply adds a validation error to any Invoice that has Line Items, which in turn causes the Force.com platform to roll back the transaction that fires the trigger (in this case, delete).

Step 2: Manually Test the Trigger

To test that the trigger does what you want it to, open an invoice that has line items, and click **Delete**. When you do, you should see an error.

Next, try to delete an invoice that does not have any line items to make sure that the trigger does not prevent the deletion of such invoices.

Create Unit Tests

Level: Advanced; **Duration:** 20-30 minutes

Apex provides built-in support for unit test creation and execution, including test results that indicate how much code is covered. Before you can add Apex classes and database triggers in your production org, you must create unit tests that programmatically validate at least 75% of the code in your organization. This tutorial gets you started with unit testing.

Why test your code with unit tests? Testing helps verify that your code executes as you expect it to, and that it doesn't consume unnecessary or extraordinary amounts of system resource. As a side-effect, it also helps ensure the integrity of Force.com releases.

Step 1: Create a Unit Test

Unit test methods take no arguments and commit no data to the database. To create unit tests for the `DeleteRestrictInvoice` trigger, complete the following steps:

1. Open the Developer Console under `Your Name` or the quick access menu ( ).
2. Click **File** > **New** > **Apex Class**.
3. In the popup, enter *`TestDeleteRestrictInvoice`* for the class name and click **OK**.
4. Replace the auto-generated code with the following code into the new Apex class editor, and then press **CTRL+S** to save the class.

For source code, see https://gist.github.com/3605669.

```
@isTest
private class TestDeleteRestrictInvoice {

   // Invoice generator, with or without a Line Item
   static Invoice__c createNewInvoice(Boolean withLineItem) {
       // Create test Merchandise
       Merchandise__c merchandise = new Merchandise__c(
           Name = 'Test Laptop',
           Quantity__c = 1000,
           Price__c = 500
       );
       insert merchandise;

       // Create test Invoice
       Invoice__c invoice = new Invoice__c();
       insert invoice;

       // Create test Line Item and insert it into the database, if withLineItem == true
       if (withLineItem) {
           Line_Item__c item = new Line_Item__c(
```

```
                name = '1',
                Quantity__c = 1,
                Merchandise__c = merchandise.Id,
                Invoice__c = invoice.Id
            );
            insert item;
        }
        return invoice;
    }

    // Single row Invoice with no Line Items => delete
    static testMethod void verifyInvoiceNoLineItemsDelete(){
        // Create test Invoice and insert it
        Invoice__c invoice = createNewInvoice(false);

        // Delete the Invoice, capture the result
        Database.DeleteResult result = Database.delete(invoice, false);

        // Assert success, because target Invoice doesn't have Line Items
        System.assert(result.isSuccess());
       }

    // Single row Invoice with Line Items => delete restrict
    static testMethod void verifyInvoiceLineItemsRestrict(){
        // Create test Invoice and Line Item and insert them
        Invoice__c invoice = createNewInvoice(true);

        // Delete the Invoice, capture the result
        Database.DeleteResult result = Database.delete(invoice, false);

        // Assert failure-not success, because target Invoice has tracks
        System.assert(!result.isSuccess());
    }

    // Bulk delete of Invoice, one without Line Items, another with
    static testMethod void verifyBulkInvoiceDeleteRestrict(){
        // Create two test Invoices, one with and without a Line Item
        Invoice__c[] invoices = new List<Invoice__c>();
        invoices.add(createNewInvoice(false));
        invoices.add(createNewInvoice(true));

        // Delete the Invoices, opt_allOrNone = false, capture the results.
        Database.DeleteResult[] results = Database.delete(invoices, false);

        // Assert success for first Invoice
        System.assert(results[0].isSuccess());
        // Assert not success for second Invoice
        System.assert(!results[1].isSuccess());
    }
}
```

Tell Me More....

The comments in the code explain the gist of the test methods. Notice that it's important, when building and testing triggers, to keep in mind that triggers can fire as the result of both single-row and bulk triggering statements. Here are a few important points to understand about building unit tests.

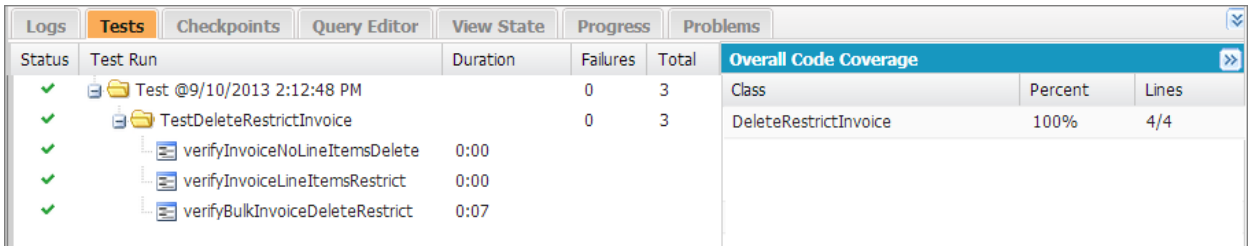
- Use the `@isTest` annotation to define classes or individual methods that only contain code used for testing.
- Test classes must be top-level classes.
- Unit test methods are static methods that are defined with the `@isTest` annotation or the `testMethod` keyword.

Step 2: Run Unit Tests

When you run a test class, the platform executes all of the unit test methods in the class and returns a report for the test run.

1. In the Developer Console, click **Test** > **New Run**.
2. To add your test class, click **TestDeleteRestrictInvoice**, and then click **>**.
3. Click **Run**.

   The test result displays in the Tests tab. You can expand the test run to view which methods were run. You'll see an output similar to this.



Any time you modify the trigger, make sure to run the corresponding unit tests so that you have confidence that the trigger still works properly.

BUILD A CUSTOM USER INTERFACE WITH VISUALFORCE

Duration: 30–45 minutes

Visualforce is a component-based user interface framework for the Salesforce platform. In previous tutorials you built and extended your app by using a user interface that is automatically generated. Visualforce gives you a lot more control over the user interface by providing a view framework that includes a tag-based markup language similar to HTML, a library of reusable components that can be extended, and an Apex-based controller model. Visualforce supports the Model-View-Controller (MVC) style of user interface design, and is highly flexible.

Code a Custom User Interface

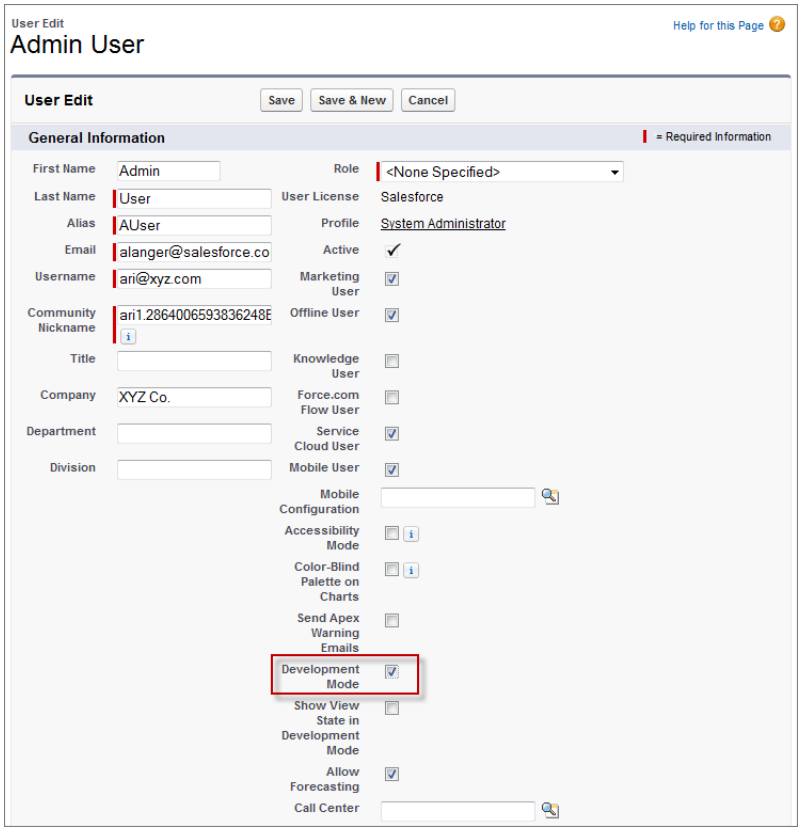
Duration: 30–45 minutes

In this tutorial, you use Visualforce to create a new interface for the Warehouse app that displays an inventory count sheet that lets you list your inventory of merchandise, as well as update the counts on each. The purpose of the count sheet is to update the computer system with a physical count of the merchandise, in case they are different.

Step 1: Enable Visualforce Development Mode

Development Mode embeds a Visualforce page editor in your browser. It allows you to see code and preview the page at the same time. Development Mode also adds an Apex editor for editing controllers and extensions.

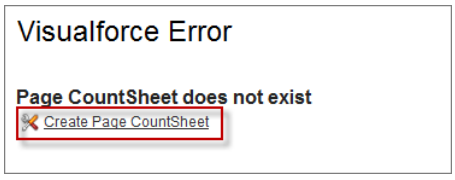
1. From your personal settings, enter *`Advanced User Details`* in the `Quick Find` box, then select **Advanced User Details**. No results? Enter *`Personal Information`* in the `Quick Find` box, then select **Personal Information**.
2. Click **Edit**.
3. Select the `Development Mode` checkbox, and click **Save**.



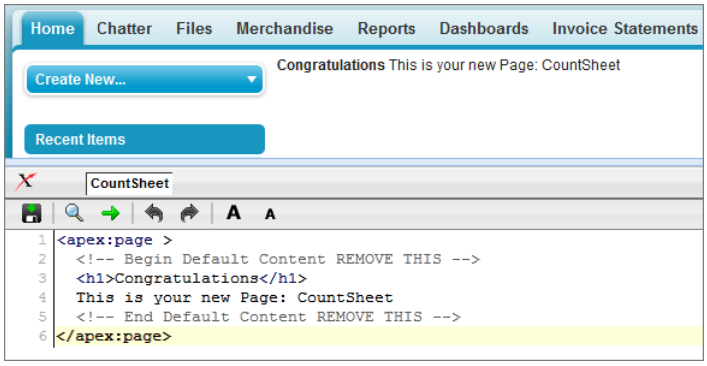
Step 2: Create a Visualforce Page

In this step you create a Visualforce page that will serve as an inventory count sheet.

1. In your browser, add the text `/apex/CountSheet` to the URL for your Salesforce instance. For example, if your Salesforce instance is `https://na1.salesforce.com`, the new URL would be `https://na1.salesforce.com/apex/CountSheet`. You will get an error message: Page CountSheet does not exist.



2. Click the **Create Page CountSheet** link to create the new page.
3. Click the Page Editor link (**CountSheet**) in the bottom left corner of the page. The Page Editor tab displays the code and a preview of the new page (which has some default text). It should look like this.



4. You don't really want the heading of the page to say "Congratulations", so change the contents of the `<h1>` tag to *`Inventory Count Sheet`*, and go ahead and remove the comments. The code for the page should now look like this.

```
<apex:page>

<h1>Inventory Count Sheet</h1>

</apex:page>
```

5. Click the **Save** icon at the top of the Page Editor. The page reloads to reflect your changes.

Tell Me More....

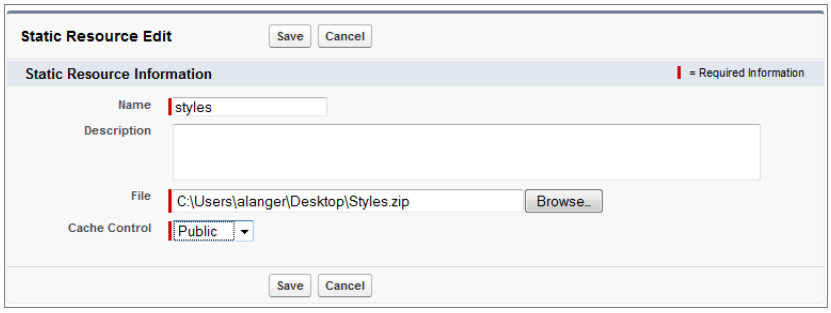
- Notice that the code for the page looks a lot like standard HTML. That's because a Visualforce page combines HTML tags, such as `<h1>`, with Visualforce-specific tags, which usually start with `<apex:>`
- If your browser has trouble displaying Developer Mode, you can turn it off in the same way you turned it on. To create a new Visualforce page, from Setup, enter *`Visualforce Pages`* in the `Quick Find` box, then select **Visualforce Pages**.

Step 3: Add a Stylesheet Static Resource

You want your Warehouse app to look slick, so you're going to use a custom stylesheet (CSS file) to specify the color, font, and arrangement of text on your page. Most Web pages and Web designers use CSS, a standard Web technology, for this purpose, so we've created one for you. In order for your pages to reference a stylesheet, you have to upload it as a *static resource*. A static resource is a file or collection of files that is stored on Salesforce. Once your stylesheet is added as a static resource, it can be referenced by any of your Visualforce pages.

To add a style sheet as a static resource:

1. In your browser, go to developer.force.com/workbook/styles. Download the file to your desktop. If the file automatically downloads, make sure to save it as a .zip file.
2. Back in the app, from Setup, enter *`Static Resources`* in the `Quick Find` box, then select **Static Resources**, and click **New**.
3. In the `Name` field, enter *`styles`*.
4. Click **Choose File** or **Browse...**, and find the `styles.zip` file you downloaded.
5. In the Cache Control drop-down list, select Public.



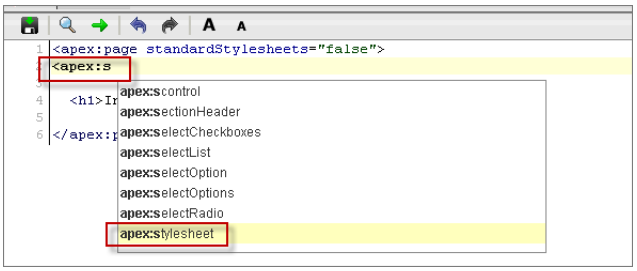
6. Click **Save**.

Now you need to modify your Visualforce page to reference the stylesheet.

1. Just as you did when you created the page, add the text `/apex/CountSheet` to the URL for your Salesforce instance.
2. Modify the attributes of the `<apex:page>` tag and enter the following code to remove the standard stylesheet, the header, and the sidebar.

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false">
```

3. Now you need to tell the page where to find the stylesheet, so add a new line below the first `<apex:page>` tag and type *`<apex:`*
4. The editor has code insight, which gives you a drop-down list of the elements that are available in this context. Start typing *`stylesheet`* and when you see `apex:stylesheet`, select it.



5. Now specify the location of the stylesheet as shown.

```
<apex:stylesheet value="{!URLFOR($Resource.styles, 'styles.css')}" />
```

6. Verify that your code looks like the following:

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false">

<apex:stylesheet value="{!URLFOR($Resource.styles, 'styles.css')}" />

<h1>Inventory Count Sheet</h1>

</apex:page>
```

7. Click the **Save** icon at the top of the Page Editor.

Note how the page now looks very different, the title is in a different font and location, and the standard header and sidebar are no longer present.

Tell Me More....

Let's take a look at that stylesheet code in a little more detail.

- `$Resource` is a global variable accessible in Visualforce pages. With `$Resource.styles`, you refer to the resource called "styles" that you created earlier.
- The `URLFOR()` function locates the static resource, and a file within that resource, and calculates the URL that should be generated in your final page. If the syntax looks familiar, it's because you've already encountered it to dynamically evaluate values when the Visualforce page is rendered.
- Why did you download a .zip file for only one small stylesheet? Usually stylesheets (and other static references) come in bundles of more than one, and so it's useful to see the code that accesses a .zip file. If you had simply uploaded `styles.css` you could refer to it using `<apex:stylesheet value="{$Resource.styles}" />`. While that code is simpler, you wouldn't know how to refer to files in an archive. After the stylesheet is uploaded as a `.zip` file in a static resource, all you need to do is enter the name of the stylesheet between single quotes: `<apex:stylesheet value="{!URLFOR($Resource.styles, 'enter_stylesheet_name.css')}" />`.

Step 4: Add a Controller to the Page

Visualforce's Model-View-Controller design pattern makes it easy to separate the view and its styling from the underlying database and logic. In MVC, the view (the Visualforce page) interacts with a controller. In our case, the controller is usually an Apex class, which exposes some functionality to the page. For example, the controller may contain the logic that should be executed when a button is clicked. A controller also typically interacts with the model (the database)—exposing data that the view might want to display.

All Salesforce objects have default standard controllers that can be used to interact with the data associated with the object, so in many cases you don't need to write the code for the controller yourself. You can extend the standard controllers to add new functionality or create custom controllers from scratch. In this tutorial you'll use the default controller.

1. If the Page Editor isn't open on your Visualforce page, click **Page Editor** to edit the page.
2. Enable the `Merchandise__c` standard controller and add the standard list controller definition by editing the first `<apex:page>` tag. The editor ignores whitespace, so you can enter the text on a new line.

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
standardController="Merchandise__c" recordSetVar="products">
```

3. Click the **Save** icon at the top of the Page Editor. You won't notice any change on the page. However, because you've indicated that the page should use a controller, and defined the variable `products`, the variable will be available to you in the body of the page and it will represent a list of merchandise records.

Tell Me More....

Take a look at what you added to the `<apex:page>` tag.

- Setting the `standardController` attribute connects your page to the standard controller for a specific object, in this case, the `Merchandise__c` object.
- Setting the `recordSetVar` attribute puts a standard controller into "list" mode and sets a *`products`* variable, which will contain the list of merchandise records.

Step 5: Display the Inventory Count Sheet as a Visualforce Page

You now have all the functionality in place to flesh out the Visualforce page. It will display a table of all the merchandise records, together with an input field on each so that you can update the inventory count.

1. In the line below the `</h1>` tag, start typing *`<apex:f`* on a new line, and highlight *`<apex:form>`* when it appears in the drop-down list. The form will allow you to make updates to the page.
2. Press ENTER, and notice that the system generates the opening and closing tags for you.
3. Place your cursor between the tags and create a data table. Start by typing *`<apex:d`* and press ENTER to select *`dataTable`* from the drop-down list.
4. Now you need to add some attributes to the `dataTable` tag. On one or more lines within the tag, enter the following.

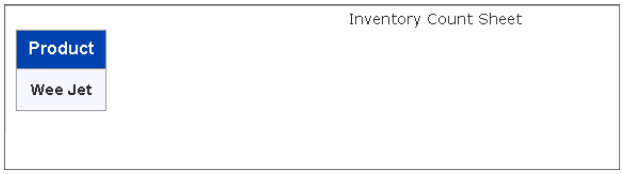
   ```
   <apex:dataTable value="{!products}" var="pitem" rowClasses="odd,even">
   ```

   The `value` attribute indicates which list of items the `dataTable` component should iterate over. The `var` attribute assigns each item of that list, for one single iteration, to the *`pvitem`* variable. The `rowClasses` attribute assigns CSS styling names to alternate rows.

5. Now you are going to define each column, and determine where it gets its data by looking up the appropriate field in the *`pitem`* variable. Add the following code between the opening and closing `dataTable` tags.

   ```
   <apex:dataTable value="{!products}" var="pitem" rowClasses="odd,even">
      <apex:column headerValue="Product">
         <apex:outputText value="{!pitem.name}"/>
      </apex:column>
   </apex:dataTable>
   ```

6. Click **Save**, and you will see your table appear.

   The `headerValue` attribute has simply provided a header title for the column, and below it you'll see a list of rows: one for each merchandise record. The expression `{!pitem.name}` indicates that we want to display the name field of the current row.

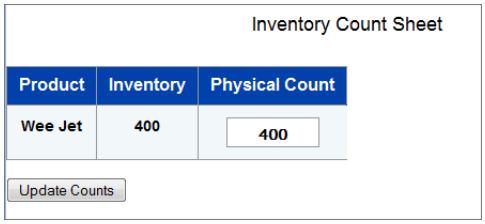
7. Now, after the closing tag for the first column, add two more columns.

   ```
   <apex:column headerValue="Inventory">
      <apex:outputField value="{!pitem.Quantity__c}"/>
   </apex:column>
   <apex:column headerValue="Physical Count">
      <apex:inputField value="{!pitem.Quantity__c}"/>
      </apex:column>
   ```

   Note: The second column is an `inputField`, not an `outputField`. The `inputField` will display a value, but it will also allow you to change it.

8. Click **Save** and you have an inventory count sheet! It lists all the merchandise records, displays the current inventory, and provides an input field for the physical count.
9. As a final embellishment, add a button that will modify the physical count on any row and refresh the values on the page. To do this, enter the following code directly above the `</apex:form>` line.

   ```
   <br/>
   <apex:commandButton action="{!quicksave}"  value="Update Counts" />
   ```



Tell Me More....

- The `dataTable` component produces a table with rows, and each row is found by iterating over a list. The standard controller you used for this page was set to `Merchandise__c`, and the `recordSetVar` to `products`. As a result, the controller automatically populated the products list variable with merchandise records retrieved from the database. It's this list that the `dataTable` component uses.
- You need a way to reference the current row as you iterate over the list. That statement `var="pitem"` assigns a variable called `pitem` that holds the current row.
- Every standard controller has various methods that exist for all Salesforce objects. The `commandButton` component displays the button, and invokes a method called `quicksave` on the standard controller, which updates the values on the records. Here, you're updating the physical count of the product and performing a quick save, which updates the product with the new count.
- Although pagination isn't shown in this example, the functionality is there. If you have enough records to page through them, add the following code below the `commandButton` for page-flipping action.

```
<apex:commandLink action="{!next}" value="Next" rendered="{!hasNext}" />
```

Step 6: Add Inline Editing Support

You now have a Visualforce page that contains a table that displays all the merchandise records and allows you to edit the inventory count through the physical count input field. In this step, you modify this table to add inline editing support for the inventory output field. Also, since inline editing makes the physical count input field unnecessary, you remove the last column, which contains this field. After carrying out this step, you will be able to edit the inventory count by double-clicking a field in the **Inventory** column.

Instead of using an `inputField` to edit the physical count in the last column in the previous step, you can make the inventory column editable by adding an `inlineEditSupport` component as a child component of the `outputField` component. The following procedure shows how to do this.

1. Delete the following markup for the physical count column.

```
<apex:column headerValue="Physical Count">
   <apex:inputField value="{!pitem.Quantity__c}"/>
</apex:column>
```

2. Within the inventory column, break up the `outputField` component so that it has an end tag, as follows.

```
<apex:outputField value="{!pitem.Quantity__c}">

</apex:outputField>
```

3. Between the `outputField`'s start and end tag, insert the `inlineEditSupport` component.

```
<apex:inlineEditSupport event="ondblclick" showOnEdit="update"/>
```

4. Now that you've added the `inlineEditSupport` component, modify the update button to give it an ID and a style class name. The ID is referenced by the `inlineEditSupport` component to show the button during editing. The style class name is used in `styles.css` to hide the update button the first time the page renders. Replace the `commandButton` with the following.

```
<apex:commandButton id="update" action="{!quicksave}" value="Update Counts"
  styleclass="updateButton"/>
```

5. Your Visualforce markup should look like the following.

```
<apex:page standardStylesheets="false" showHeader="false" sidebar="false"
  standardController="Merchandise__c" recordsetVar="products">
   <apex:stylesheet value="{!URLFOR($Resource.styles, 'styles.css')}"/>
   <h1>Inventory Count Sheet</h1>
   <apex:form>
      <apex:dataTable value="{!products}" var="pitem" rowClasses="odd,even">
         <apex:column headerValue="Product">
            <apex:outputText value="{!pitem.name}"/>
         </apex:column>
         <apex:column headerValue="Inventory">
            <apex:outputField value="{!pitem.Quantity__c}">
               <apex:inlineEditSupport event="ondblclick" showOnEdit="update"/>
            </apex:outputField>
         </apex:column>
      </apex:dataTable>
      <br/>
      <apex:commandButton id="update" action="{!quicksave}" value="Update Counts"
        styleclass="updateButton"/>
   </apex:form>
</apex:page>
```

6. Save. The page now displays the inventory count table with two columns. Notice that the **Update Counts** button is hidden initially.
7. Double-click any cell in the inventory column. The field dynamically becomes an input field and the **Update Counts** button appears.

8. Modify the count value and click **Update Counts** to commit this update.

Tell Me More....

- The `event` attribute of the `inlineEditSupport` component is set to "ondblclick", which is a DOM event and means that the output field will be made editable when you double-click it. Also, the `showOnEdit` attribute causes the Update Counts button to appear on the page during an inline edit. This attribute is set to the ID of the Update Counts button.
- The Update Counts button is hidden through its style specification in the static resource file `styles.css`. The `styleclass` attribute on `commandButton` links this button to an entry in `styles.css`.

Summary

Congratulations! You have created a new interface for your Warehouse app by creating a Visualforce page that uses a standard controller. Your page is highly configurable. For example, you can easily modify which data is displayed in each row by modifying the column components. The page also makes use of a lot of functionality provided by the standard controller behind the scenes. For example, the controller automatically queries the database and finds all merchandise records and assigns them to the `products` variable. It also provides a way of saving records, through its quicksave method.