Beginning Oracle Application Express 5

Your ticket to easy and robust
web-application development using
Oracle's powerful toolset for
power-users, programmers, and
database administrators

—

Doug Gault

Apress®

⟨IOUG⟩
independent oracle users group

Beginning Oracle Application Express 5

Doug Gault

⟨**IOUG**⟩
independent oracle users group

Apress®

Beginning Oracle Application Express 5

To those in search of knowledge and better understanding,
I dedicate this effort. Hopefully, as your skills grow,
you too will continue to share the wealth.

—Doug Gault

About IOUG Press

*IOUG Press is a joint effort by the **Independent Oracle Users Group (the IOUG)** and **Apress** to deliver some of the highest-quality content possible on Oracle Database and related topics. The IOUG is the world's leading, independent organization for professional users of Oracle products. Apress is a leading, independent technical publisher known for developing high-quality, no-fluff content for serious technology professionals. The IOUG and Apress have joined forces in IOUG Press to provide the best content and publishing opportunities to working professionals who use Oracle products.*

Our shared goals include:

- Developing content with excellence
- Helping working professionals to succeed
- Providing authoring and reviewing opportunities
- Networking and raising the profiles of authors and readers

To learn more about Apress, visit our website at **www.apress.com**. Follow the link for IOUG Press to see the great content that is now available on a wide range of topics that matter to those in Oracle's technology sphere.

Visit **www.ioug.org** to learn more about the Independent Oracle Users Group and its mission. Consider joining if you haven't already. Review the many benefits at www.ioug.org/join. Become a member. Get involved with peers. Boost your career.

www.ioug.org/join

Apress®

Contents at a Glance

Contents

About the Author

Doug Gault is a Consulting Member of Technical Staff at Oracle Corporation and has been working with Oracle since 1988, starting with version 5.1B, SQL*Forms 2.0, and RPT/RPF. He has focused his career on Oracle's development technologies, spending the majority of that time dedicated to web-based technologies, including the OWA Web Toolkit, PL/SQL Server Pages, WebDB, Oracle Portal, and more recently HTML-DB and APEX.

His many years of Oracle experience have taken him all over the world to participate in some truly groundbreaking projects. Doug has presented and participated in roundtable discussions at a number of conferences, including Oracle OpenWorld, UKOUG, and ODTUG's APEXposed & Kaleidoscope conferences. He holds an Associate's Degree in Computer Science and an honorary Master's Degree from The School of Hard Knocks, believing there is no replacement for hard-earned experience.

Doug can be found on Twitter as `@dgault_apex` and on his blog at `douggault.blogspot.com`.

About the Technical Reviewer
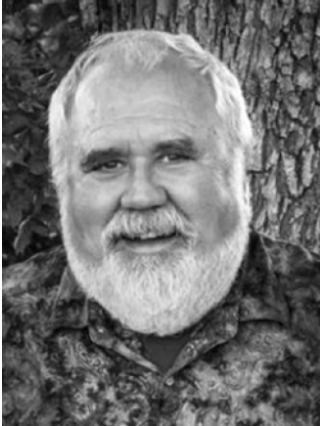
Warren Capps, president of Illuminations Inc, has worked with Oracle since 1987 when he worked on version 5.1a. Since 1991, his principal efforts have been spent in training clients in the use of Oracle products, concentrating on database server technologies. He is a well-known presenter at user-group conferences and has written numerous articles and book reviews for a variety of publications. He also ran an Oracle bookstore for ten years and is a retired Navy Commander.

When not teaching, Warren has myriad activities to keep him busy. He is an avid photographer and has run photography workshops in southern New Mexico. His photography has led him to visit over 25 countries. Additionally, he plays classical guitar, collects coins, and loves to travel the country with his wife and cat. He is currently a resident of Austin, Texas.

Acknowledgments

■ ■ ■

An Introduction to APEX 5.0

Welcome to the wonderful world of Oracle Application Express (APEX). You're about to learn how to use a tool that will revolutionize the way you think about and approach writing web-based Oracle systems. It certainly has done so for me.

Prior to the advent of APEX, developing fully interactive, web-based systems for data that resided within an Oracle database almost always meant learning a new and often complex language like Java, .NET, or PHP and then figuring out how to integrate your chosen language seamlessly with that data. Often this also meant trying to incorporate business rules that were already coded in the form of PL/SQL program units.

In such situations, it could take months or even years just to become proficient enough with your chosen language to begin to write a functional system. If you're like many developers, you become frustrated with the fact that you've spent an inordinate amount of time doing what seems to be a relatively easy task.

Fear not! The days of long-winded and complex web-development platforms may be behind you.

What Is APEX?

APEX is a 100% browser-based rapid application development (RAD) tool that helps you to create rich, interactive, Oracle-based web applications very quickly and with relatively little programming effort.

There are many RAD development tools and platforms on the market. If you're dealing with data that resides in an Oracle database, a number of things make APEX distinctive and thus more attractive as a development platform. First and foremost, APEX is built on and uses as its core languages SQL and PL/SQL. This is a huge advantage for those of you who have already been working with the Oracle database, because it means you can immediately draw on what you know. Even if you don't have an Oracle background, but are going to be working with an Oracle database, you need to learn about its particular flavor of SQL and will at some point likely find a need for the PL/SQL procedural language.

PL/SQL program units become even more beneficial when migrating from an Oracle-based system that already has a significant amount of business logic coded into stored PL/SQL program units. In this instance, you can almost immediately take advantage of that logic with very little effort or changes to the existing code.

Another great advantage is that APEX is a declarative tool that provides a feature-rich core of functionality designed to make your job easier. Because APEX takes care of many of the underlying functions common to all web-based applications, you can focus on the logic specific to your application.

A large share of what you need to accomplish can be done using one of the many built-in wizards provided as part of the APEX Application Builder. The wizards walk you through the process of defining what you want your application to do and then store that information as metadata. Once a wizard is complete, you can edit and enhance the functionality or even replace it with your own custom SQL and PL/SQL routines. After you become proficient with APEX, you might even find yourself bypassing the wizards altogether and generating more-complex definitions directly.

During the course of this book, you'll likely discover that you want a few other tools at your disposal, but, in truth, you could easily develop a very rich application using nothing but your web browser and what APEX provides for you.

A Brief History of APEX

APEX has been around for quite some time—perhaps even longer than most people know. The first public release of APEX, or HTML DB, as it was called then, came in 2004, but its history reaches back a long way.

Ancient History

APEX has its roots in technology that has been around for quite a while. In fact, parts of the PL/SQL Web Toolkit, which is used under the covers by APEX to generate the HTML that is sent to the browser, date back to as early as 1994.

At that point in time, you *could* actually write web applications in PL/SQL by hand, and unfortunately many of us did. This required not only a thorough knowledge of PL/SQL and HTML, but also the patience of a saint and the determination of a headstrong mule. The end result wasn't very pretty, and it was definitely not secure by today's terms, but it was functional, if somewhat limited.

Not long after, Oracle introduced PL/SQL Server Pages (PSPs). This involved first coding the static HTML and including special Oracle markup to indicate where dynamic data would go. Once you had the output looking as you wanted, you then ran it through a program called `LOADPSP`. This would translate the raw HTML and the special Oracle markup into a PL/SQL procedure that, again, used the PL/SQL Web Toolkit to emit the HTML, including the dynamic data you requested. At the time, this was a huge leap forward. I worked at a company where I built an entire framework using PSP technology and deployed it at several clients.

Finally, in 1997, WebDB came on the scene. The true grandfather of what is now called APEX, WebDB was revolutionary in that it was a 100% web-based tool that allowed developers to design web applications. It was written entirely in PL/SQL, even though Java seemed to be taking over the world. Developers could point WebDB at their database and generate code that would produce forms, reports, charts, and calendars. There was no session-state management, and there were no templates; once the code was generated, you couldn't go back through the tool.

WebDB allowed a large number of companies that wanted to jump on the web-based bandwagon to do so without spending vast amounts of time and effort retraining their staff. As a tribute to its success, I know of a number of companies that still have WebDB systems running in production environments.

Unfortunately, WebDB's days were numbered. Because it generated code (and if you didn't like the code it generated, then too bad for you), it had already begun to fade from favor by the time it was absorbed into Oracle's Portal product. However, creator Mike Hichwa didn't forget the glimpse of greatness that WebDB had seen.

More Recent History

Around 1999, Oracle CEO Larry Ellison presented Mike Hichwa (VP of Software Development) with the task of creating an internal calendaring and scheduling system for Oracle Corp. The original remit was to use WebDB to generate the initial code and then hand-code all the changes from that point forward. Mike, however, saw this as an opportunity to completely rewrite WebDB into something that could be far more useful. Thus, with the help of Joel Kallman and Tom Kyte, Oracle Flows was born.

Based on the success of the internal calendaring and scheduling system, the team was allowed to move forward toward making Oracle Flows a product. In 2001, using what was then known as Flow Builder, Mike and his team began implementing systems for various customers, including one situation where they managed to replace a Java development project that was going horribly wrong.

By 2003, the team had proven the tool's power, and they were given permission to release it as a product. HTML DB 1.5 was released to the public as a no-cost option of Oracle 10gR1.

Since then, various releases have been introduced, each providing improved features and functionality. The following is a very brief list of the releases and some of the more notable features:

- HTML DB 1.6 (2004) introduced themes, master-detail forms, page groups, page locking, and some multilingual capabilities.

- HTML DB 2.0 (2005) introduced SQL Workshop, a graphical query builder, a database object browser, and session-state protection.

- APEX 2.2 (2006) introduced packaged applications, the APEX dictionary views, and the access control wizard.

- APEX 3.0 (2007) introduced PDF printing with BI Publisher, migration from Microsoft Access, and page and region caching.

- APEX 3.1 (2008) introduced interactive reports, the runtime-only installation capability, and improved security.

- APEX 3.2 (2009) introduced a migration helper for Oracle Forms–based systems and various security enhancements.

- APEX 4.0 (2010) was a huge leap forward, introducing dynamic actions and plug-ins–declarative ways to introduce server-side logic and extend the core APEX environment, respectively. Also introduced was the new Team Development module.

- APEX 4.1 (2011) included a new user-facing data-uploading feature, enhanced error-handling capabilities, and much-improved support for tabular forms.

- APEX 4.2 (2012) originally introduced some new themes as well as enhancements to the debugging API, but over its more than two-year life span, patch releases introduced such changes as HTML 5 charting and deeper security enhancements.

APEX 5.0 and the Future

And so we arrive at the release of APEX 5.0. While the changes introduced with versions 4.0 through 4.2 undoubtedly changed the landscape of APEX development, the changes introduced in version 5.0 have brought APEX to a point where it can rightly be compared with many of the popular desktop-based development environments.

The original focus of APEX 5.0 was to make development of rich, interactive web applications easier by providing the developer with a vastly enhanced development environment. However, the development team has introduced so many new features—indeed, new ways to attack problems—that it will be hard *not* to choose APEX as the preferred development platform for Oracle-based applications.

APEX's new Page Designer Integrated Development Environment (IDE) completely changes the way developers will interact with page design. Modeled after many of the popular desktop IDEs, developers now interact with items, placement, attributes, and actions all on one page. A new drag-and-drop page-layout interface has been introduced that allows developers to easily position regions and items. Group editing

allows developers to edit the attributes of several items at once. The only downside to the new Page Designer is that you may find yourself wanting a bit more screen real estate due to the nature of its layout. However, with widescreen monitors becoming ubiquitous, this shouldn't be an issue for most.

Apart from the new Page Designer IDE, one of the most exciting new features of APEX 5.0 is the Universal Theme. This new application user interface does away with the need for the complex templates from days gone by and enables developers to build more modern, responsive, and consistent applications without needing to know the intricate details of HTML, CSS, or APEX template design.

The new Universal Themes (Desktop theme 42 and Mobile theme 51) allow you to adjust a number of attributes with what is called a Theme Style–a Cascading Style Sheet (CSS) that is added to the base CSS. This can be done via the new Theme Roller tool, allowing you to visually alter a theme. The Universal Themes also allow you to easily customize how items on the page are displayed by using Template Options.

After having been in the cards for quite some time, the Flexible Workspace Authentication has finally been implemented by the APEX team; this allows APEX administrators to define how APEX itself will authenticate developers. Much like APEX applications, workspaces may now be authenticated against Single Sign-On servers, LDAP, and so forth.

Interactive Reports are no longer limited to being one-per-page, freeing you from the restriction that had plagued them since their inception. Interactive Reports also get a few new features. A Pivot View has been added that allows end users to select the column(s) and provide the function(s) by which to pivot the report. This was functionality previously available only by either a lot of hand coding or by creating or using plug-ins. When using the new Universal Theme, Report column headers can now be defined so that they remain fixed in position while the user scrolls down the page.

Native support for Dialog page types has been introduced, thus allowing any page to be displayed either normally or as a pop-up dialog. Pages can be defined as either "Modal" or "Non-Modal." Modal pages do not allow the end user to interact with the underlying page, whereas Non-Modal pages allow such interaction.

New jQuery Mobile and Tablet themes have been introduced and make use of the newer features of the latest jQuery Mobile libraries. Panels, pop-ups, and dialogs (among other things) are now all available in the mobile interface.

An improved charting engine provides enhanced performance for large datasets. Improvements to accessibility for the visually impaired have been added. A new APEX_AUTHORIZATION package has been added to aid in the management of authorization within an application. And the list goes on.

As you can see, the APEX core functionality continues to grow with each release. But what you may not know is that you can help drive the future direction of APEX. By going to the following URL, you can not only request new features, but also view and vote on features that others have requested. You need an Oracle Technical Network account, but it's free and easy to sign up:

https://apex.oracle.com/pls/apex/f?p=55447:1

To get a view of what the APEX team is committed to providing, you can read the most recent Statement of Direction (SoD). It may take a short time after a release for this to be updated, but it normally contains an overview of the main functional areas for the next planned release. You can find the SoD at the following URL:

www.oracle.com/technetwork/developer-tools/apex/application-express/apex-sod-087560.html

What You Need to Get Started

The goal of this book is to get you started using APEX, to launch you in a way that enables you to grow toward mastery of the product. To begin, you need three things: access to an APEX instance, access to a web browser, and a copy of SQL Developer.

Access to an APEX Instance

This is definitely a hands-on book, so to work through the examples and exercises you need access to an instance of APEX 5.0. There are a number of different ways you can access APEX; depending on your level of comfort and expertise with Oracle, some may be better for you than others. Here is a description of the three most common scenarios:

- By far the easiest is to sign up for an account on Oracle's hosted version of APEX at `https://apex.oracle.com`. It's free for nonproduction applications and is a great place to get started, because you don't have to worry about installing either the database or APEX.

- If you already have an Oracle database installed locally, you can download and install APEX 5.0 into that instance. Simply go to the Oracle APEX home page at `http://otn.oracle.com/apex` and download the latest version of the software.

- If you don't have an Oracle database already but would like to install one locally, you can download a free developer's license version of the database from Oracle Technology Network (OTN) at `http://otn.oracle.com/database`. Both Oracle 11g and 12c run APEX 5.0. Both allow you to install APEX (albeit an earlier version) as an option during the database install.

Although having a locally accessible instance of the Oracle database gives you more direct access to the data, it's definitely not necessary for completing the exercises in this book. All code and instructions have been written so that they can be completed on Oracle's hosted instance with no special access required.

■ **Note** Oracle provides very good documentation on the installation process for both the database and APEX, so it isn't covered in detail here. However, if you're planning to install APEX on an environment in your organization, you should coordinate with the database administrator responsible for that instance to ensure that no mishaps occur.

Web Browser

The APEX documentation states that to view or develop APEX applications, you must have a web browser that supports cookies, JavaScript, HTML 5, and CSS 3. However, although you can deploy to any browser that supports these things, the list of supported browsers is fairly narrow. Currently, the following browsers are supported: Internet Explorer 9+, Firefox 35+, Apple's Safari 7+, and Google Chrome 40+.

Without getting into a religious debate about which web browser is the best on the market, the author's preference for development is either Firefox or Chrome due to the number of developer tools and add-ons that can help you with APEX development. Note that because of the difference in the way each browser interprets HTML and JavaScript, you must test your application in *any and all* web browsers that your target audience might use.

SQL Developer

As mentioned before, all the exercises and scripts in this book can be loaded and run directly within the APEX interface. However, if you have chosen to install or have access to a local instance of the Oracle database, a SQL IDE will definitely make your life easier.

SQL Developer is a free SQL and PL/SQL IDE provided by Oracle. You can download SQL Developer from the OTN's home page at `http://www.oracle.com/sqldeveloper`.

Using SQL Developer, you can browse database objects, edit row data, develop and test stored PL/SQL program units, code and test SQL statements, and interactively debug PL/SQL code. SQL Developer also has many direct integration points with APEX that make reporting in, monitoring, and maintaining APEX instances and applications easier. This book doesn't cover those, but it's definitely worth your time to look into this tool.

Summary

Oracle Application Express has come a long way from its simple beginnings, and the APEX community is poised at the beginning of a new cycle of growth. APEX 5.0 provides so much possibility and promise that it's hard not to be excited about what the future holds. With that spirit, you're ready to begin your journey to discover how APEX can make development easier and more fun.

CHAPTER 2

■ ■ ■

A Developer's Overview

You're probably anxious to get started, but there are a few concepts you should understand before you jump into APEX development headfirst. This chapter will introduce the fundamental development architecture of APEX and then walk you through the different areas of the developer interface.

You will delve deeper into the details as you go through the book and put the architecture to work, but it will help tremendously to know how things are structured ahead of time. This chapter is designed to ease you in, but it isn't a complete guided tour of every nook and cranny. Be patient; you'll get there.

The Anatomy of a Workspace

APEX was designed from the beginning to be a multi-tenant architecture where many different development environments (called *workspaces*) can exist in a single APEX instance. For instance, apex.oracle.com, Oracle's free hosted instance, holds over 10,000 active workspaces, each of which is a completely separate environment unable to see or interact with any of the others. You can think of this as Software as a Service (SaaS) or a cloud-computing architecture, but basically it means each workspace is distinct and segregated from all others.

In simple terms, each workspace represents a virtual private container in which developers create and deploy their APEX applications. The development process takes place in the context of a workspace, so it's important to know how a workspace is structured. Figure 2-1 uses database entity-relationship diagram parlance to help explain the makeup of the objects in a workspace.
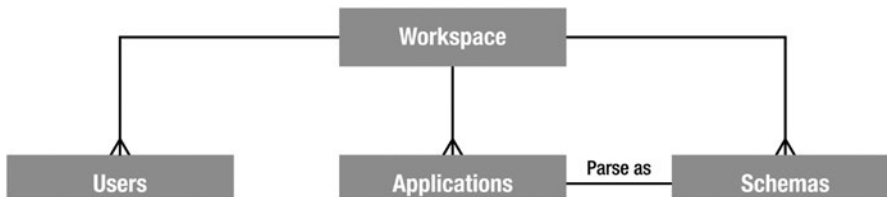
Figure 2-1. Logical makeup of a workspace

A workspace may have:

> *One to many users*: These users may one of three types: Administrator, Developer, or End User.
>
> *Zero to many applications*: Applications can be added from the list of packaged applications, imported, or created from scratch.
>
> *One to many schemas*: Although a workspace must be assigned at least one schema when it's created, an Instance Administrator may assign multiple schemas to a workspace.

There can be many applications and many schemas in a workspace, but an application may only parse as one (and only one) schema, which can only be set during development. The following sections delve more deeply into this to give you a full understanding of how these concepts relate.

APEX Users

To log in to an APEX workspace, you must have access to a valid APEX user. A number of different user roles are available that dictate what you can do when you log in. The roles are as follows:

- *Instance Administrators* are special users who manage and maintain the overall APEX instance. They can set instance-level preferences and messages, create and manage workspaces, monitor space utilization, and perform many other actions related to the overall APEX installation. Instance Administrators are only able to log in to the special INTERNAL workspace, which houses the APEX Admin Services application.

- *Workspace Administrators* are responsible for managing the details of a specific workspace and can manage user accounts related to the workspace, monitor workspace activity, view log files, override developer locks and settings, and so on. Although it isn't good practice, the Workspace Administrator can also act as a Developer, creating and modifying applications.

- *Developers* are the users who create and edit the applications in the workspace. They have access to the underlying tables in the schema(s) assigned to the workspace and may create and modify database objects and stored PL/SQL units. Most people writing APEX applications only need this level of access.

- *End Users* are only able to run applications in a workspace. They don't have direct access to any of the underlying database objects, nor do they have access to any of the APEX development modules. End users can't log directly into a workspace.

With the exception of the APEX Instance Administrator, in a default installation APEX users are specific and unique to a workspace, meaning you can have users with the same name in multiple workspaces in a single APEX instance, but each of these users is unique. They can have their own passwords and settings and aren't linked together in any way.

APEX 5.0 introduces the ability to use an external repository, such as Single Sign-on or LDAP, as a source to assign and validate APEX users, meaning that a single user could have access to multiple workspaces. However, this functionality is not set up by default and requires an Instance Administrator to configure.

When you're developing, you should get in the habit of logging in as a Developer as opposed to a Workspace Administrator. Several safeguards are available to help keep developers from stepping on each other in a workspace. If you log in as a Workspace Administrator, these safeguards are bypassed, and you may accidently interfere with something someone else is working on. Although this isn't a problem in a workspace with only one developer, it's still good to get into that habit.

8

■ **Note**   This book uses the last three types of user. It assumes that APEX has been installed, a workspace has been created, and you have been given the Workspace Administrator's login credentials. If you're using the hosted instance at `apex.oracle.com`, then the user name you were given when you signed up has the credentials of a Workspace Administrator. If, however, you're using a local instance, either refer to the APEX documentation or get your Instance Administrator to help you set up a workspace.

Applications, Pages, Regions, and Items

Although a workspace starts off basically empty, you can have many applications that reside in a workspace. There is no specific rule, but it's likely that all the applications in a workspace share something: they might all use the same underlying database objects, target the same user community, or use the same method for authenticating users.

As you build an application, you add new pages and build out those pages with regions and items. Figure 2-2 shows the hierarchy of the different types of objects.
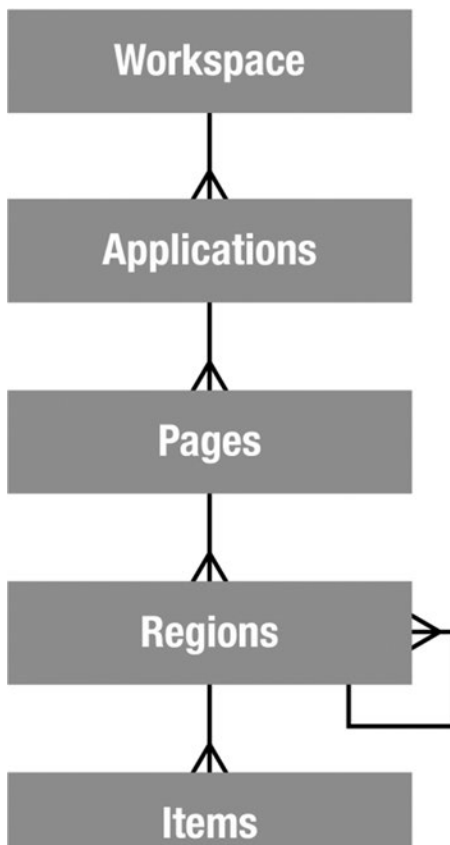
Figure 2-2. *General application hierarchy*

Applications are basically groups of pages that perform a task (or set of tasks) related to a business function. During the course of this book, you'll build one application in a single workspace, but it's important to know that in a typical development environment, you'll probably be working on many applications across several workspaces.

Pages are the basic building blocks of applications and contain both the user-interface (UI) components and the programming logic that processes the user's input. We cover the rendering of the UI versus the processing of user input later, but for now consider a page to be roughly equivalent to a screen in desktop UI lingo.

Regions are UI items that serve as content containers. You can have any number of regions on a page, and regions can be nested in other regions. This gives you the opportunity to create things like dashboards, where you might nest a data report region and a graph region in a single parent HTML region.

Items are the HTML form elements that are used to present the UI to the user. These include things such as buttons, select lists, text fields, check boxes, radio groups, and so on. There are two categories of items: page-level items and application-level items. The difference is that the latter are defined at the application level and aren't rendered directly on the page. You can think of these as global variables. Page-level items are defined on a specific page and are assigned to a region in order to control where and how they display to the user.

There is obviously a lot more to an application than these simple building blocks, but if you understand the basic hierarchy between these, you'll have a jumpstart when it comes to building your first pages and a solid foundation when it's time to perform more intricate tasks.

Workspaces, Applications, and Schemas

Although the relationship between workspaces and applications is straightforward, it becomes a bit more complex when you introduce the relationship with database schemas. Figure 2-3 diagrams this relationship.
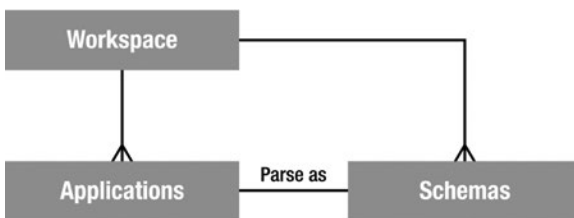
Figure 2-3. *How schemas relate to workspaces and applications*

When a workspace is created, it's linked with at least one, and possibly many, underlying database schemas. This provides access to database objects such as tables, views, stored PL/SQL program units, and so on.

When an application is created, it's assigned a single "parse as" schema from the list of schemas associated with the workspace. A *"parse as" schema* is the Oracle database user in which all SQL queries and PL/SQL calls run by that application are executed. So, if your application was defined with a "parse as" schema of DOUG, a query such as

```
select * from emp
```

would execute in the database as if it were written

```
select * from DOUG.emp
```

Because APEX applications are portable and may not necessarily be run in the same schema they were developed in, it's not good practice to hard code the schema names into your SQL or PL/SQL. Instead, APEX provides a replacement variable (one of many you'll be introduced to throughout the course of this book) for the "parse as" schema. The #OWNER# replacement variable is substituted for the actual "parse as" schema for the application at runtime. So the statement

```
select * from #OWNER#.emp
```

resolves to

```
select * from DOUG.emp
```

In the most common implementations, a workspace is created and associated with a single underlying database schema. The applications developed in that workspace have their "parse as" schema set to the only schema associated with the workspace and use the database objects belonging to that schema.

Where a workspace has more than one schema assigned to it, things can become a little more complex. You might be tempted to think that if you associate three schemas with a workspace, any application in that workspace can automatically access the data in all three schemas. However, you would be mistaken.

Because an application is assigned one—and only one—"parse as" schema, all SQL statements and PL/SQL calls are executed as that schema. Although the workspace may be associated with multiple schemas, the application itself isn't. If you want to access data in a schema other than the application's "parse as" schema, you must make sure the correct database-level grants are in place, just as you would when using any other Oracle tool or development environment.

Take a look at the example shown in Figure 2-4, where two tables you wish to join as part of a SQL statement are owned by separate schemas.
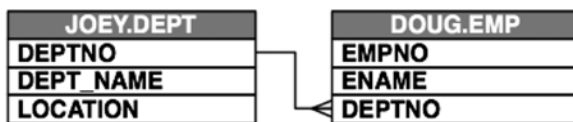
Figure 2-4. *Tables joined across schemas*

If your "parse as" schema is DOUG, then you must be specifically granted privileges on the objects in the JOEY schema to be able to access it. To do this, you sign on to the database as JOEY (or as a DBA) and grant the appropriate database privileges on JOEY.DEPT to DOUG.

In this example, if you needed to join the two tables together in a select statement, granting the SELECT privilege on JOEY.DEPT to DOUG would suffice. Then, you could write your select statement as follows:

```
select e.empno,
       e.ename,
       d.dept_name,
       d.location
  from #OWNER#.emp e,
       JOEY.dept d
 where e.deptno = d.deptno
```

The #OWNER# substitution variable would be resolved to your "parse as" schema (DOUG), and the join would work correctly as long as the correct privileges were in place.

■ **Note**   Because the grants that allow the select from the JOEY schema are put in place at the database level, it isn't necessary to associate the JOEY schema to your workspace. You only need to associate a schema to a workspace if you'll be using it as the "parse as" schema for an application in that workspace or need to access the schema objects directly from within the SQL Workshop.

A Final Word on Workspaces

As you have learned, an APEX instance can have many workspaces. But how many workspaces should there be? The answer isn't straightforward.

Unless you're in a very small organization with very few apps, you probably shouldn't have only one workspace. On the other hand, you probably shouldn't create a new workspace for every new application you code, either.

There are a couple schools of thoughts on this, but I tend to think in terms of application suites. If a number of applications are performing similar tasks against the same underlying data sets and are aimed at the same target set of users, then they would probably do well in the same workspace.

The key is to use your judgment and try to keep things easy to develop and maintain. There is nothing worse than logging in to a workspace to find you have to page through tens or even hundreds of apps to find the one you want to work on.

A Tour of the APEX Modules

Now that you have a little background on how things are logically architected, it's time to get a closer look at the APEX development environment. This section will introduce you to the different sections of the APEX environment and give you an overview of how things are laid out.

Figure 2-5 shows a hierarchical layout of the APEX menu structure. Later, you will look at each of the main sections and glimpse what's under the covers; this is just an introductory tour. You will get a much deeper look as we work our way through the development processes.

As you can see, the development environment is broken into five main sections:

- The *Application Builder* is where you create and modify applications and pages, and it's where you'll probably spend most of your time.

- The *SQL Workshop* is where you deal directly with the underlying database objects and their related data. Think of it as a web-based version of SQL*PLUS with some GUI goodness thrown in to make things easier.

- *Team Development* is the section that lets you enter and track information related to the development of APEX applications.

- *Packaged Apps* provides a way to install and manage the myriad of applications that come with Oracle APEX. Many of these applications can be used out of the box to solve real business problems. Others are merely sample applications to help demonstrate the capabilities of APEX.

- *Administration* is where you can manage the details of your workspace—its defaults, users, groups, and so on. Be aware that a Workspace Administrator has more options available to them than a standard developer has.
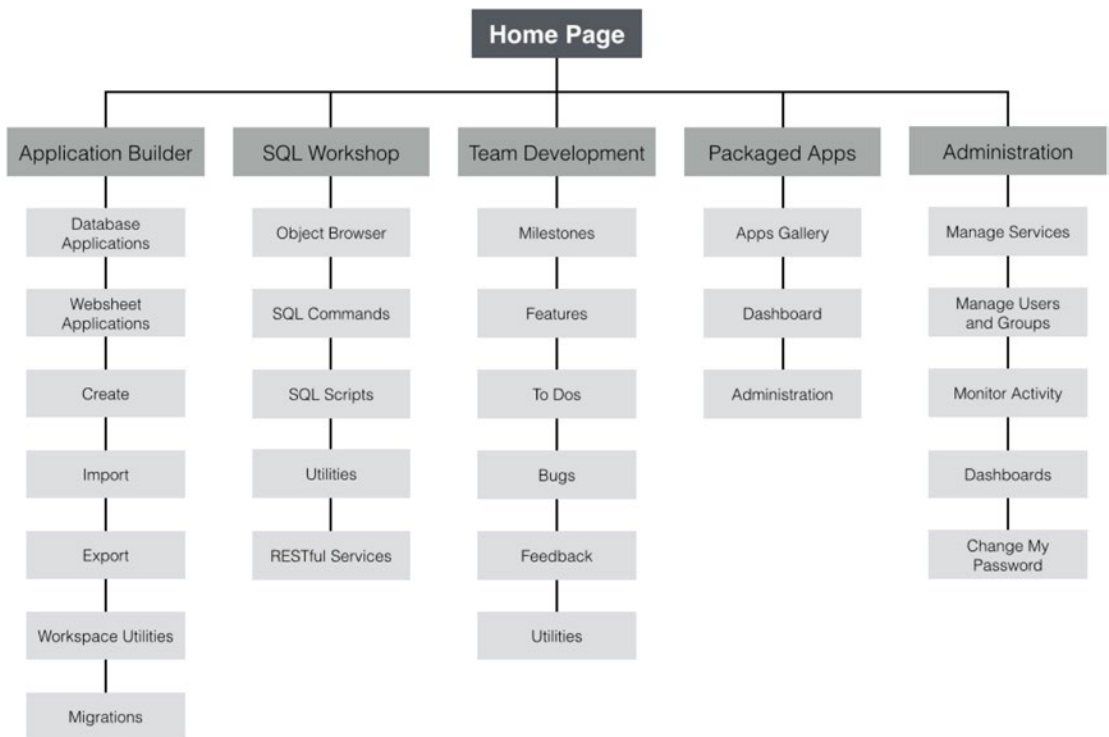
Figure 2-5. *APEX 5.0 hierarchical menu structure*

The Home Page

Once you log in to your workspace, you're presented with the workspace Home page, as shown in Figure 2-6. The Home page is your gateway to the rest of the development environment and provides some high-level information about what's going on in the workspace.
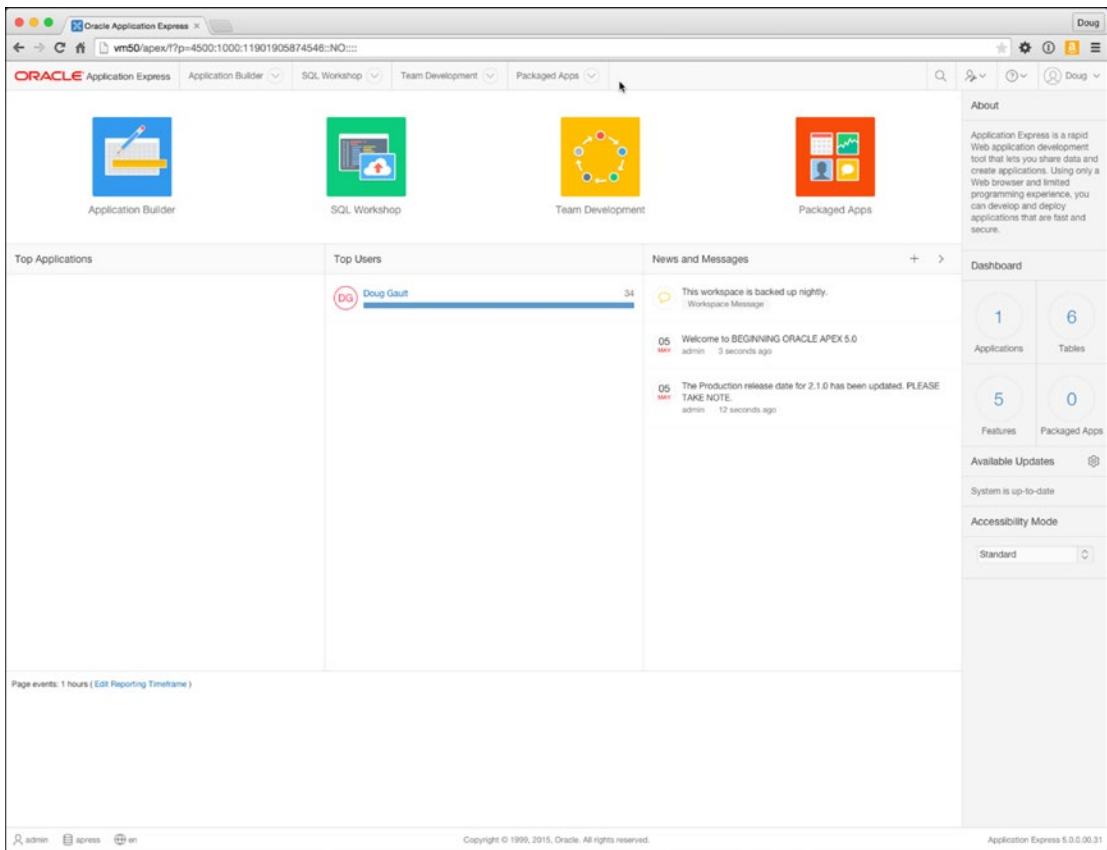
Figure 2-6. *APEX development Home screen*

Along the top is the navigation bar containing the main navigation structure available to you throughout the developer interface. It gives direct access to many of the sections you will need quick access to while you're developing applications. It's worth noting that each main option of the menu bar is broken down into two pieces. For instance, if you click directly on the Application Builder item, you're immediately taken to the Application Builder home page. However, if you click the small downward-pointing arrow just to the right, you're presented with a more detailed drop-down menu that lets you choose your destination a bit more granularly, as in Figure 2-7
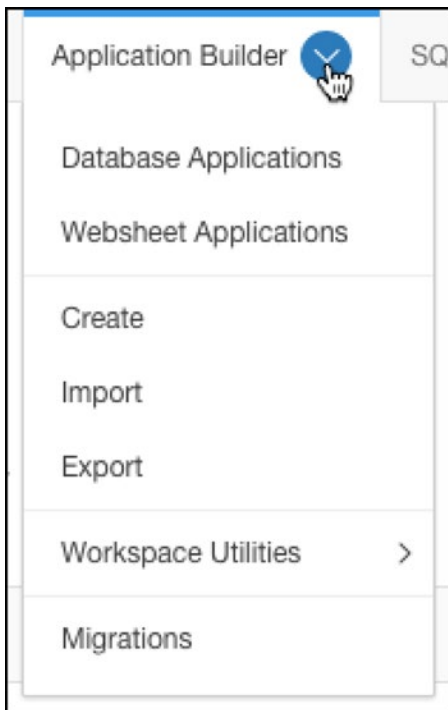
Figure 2-7. *Using the drop-down menus on the menu bar*

At the right of the navigation bar is a set of four menu options represented by icons, as shown in Figure 2-8.

Figure 2-8. *Right-hand icons on the navigation bar*

First is a search icon that, when clicked, allows you to perform context-sensitive searches. The context of the search depends on where you are in the Application Builder. For instance, if you're on the workspace Home page, your search is across the entire workspace. However, if you're in the Application Builder or the Administration section, the search is limited contextually to those specific areas.

Second is the Administration menu. This menu will be available to you whether you are a Workspace Administrator or a Developer. The difference will be what functionality you have access to. Developers will have access to monitoring certain areas of the workspace activity and to the dashboards, while Workspace Administrators will have full access to all functionality including user maintenance and service requests.

Third is the help menu, which provides access to online documentation, the APEX Support Forums, the APEX section of the Oracle Technical Network site, and an About section.

Last is a link to the profile of the currently logged in user. Here, the user will be able to edit their details, update their profile picture, and change their password.

At the very bottom of the browser is an information region that displays the currently logged in user, the current workspace, the language, and the current version of Oracle APEX.

The rest of the page is dedicated to either giving you a quick link to the four main sections or providing you with information about what's going on in the workspace.

The first two regions, from left to right, show an overview of the activity in the workspace. They show the Top Applications and the Top Users in the workspace. The News & Messages region allows the developers in a workspace to enter information they want others in the workspace to see. In a new workspace, there probably won't be anything in these regions, but as you work your way through the book, you'll see that start to change.

Notice that most of the main pages for each section of the development environment adhere to this dashboard-style home page interface, the notable exception being the Application Builder. Let's look at that section first.

Application Builder

The Application Builder is the core of the APEX application-development environment. Whereas you'll use the SQL Workshop to manipulate the underlying database objects, you'll use the Application Builder to do most of the real work when it comes to coding, testing, and debugging your applications.

The Application Builder Home Page

Clicking the Application Builder menu option takes you to the Application Builder home page. Like most of the home pages, it's laid out with the menu bar across the top and regions that hold tasks and quick links down the right side.

The main difference is that Application Builder home page doesn't house any dashboard-style summaries. Instead, this is where you see a list of the different applications contained in your workspace. (Figure 2-9 provides an example.) It's possible, depending on your APEX instance settings, that you might see some sample applications installed by the Workspace Administrator, but don't be alarmed if you don't see any applications at all.
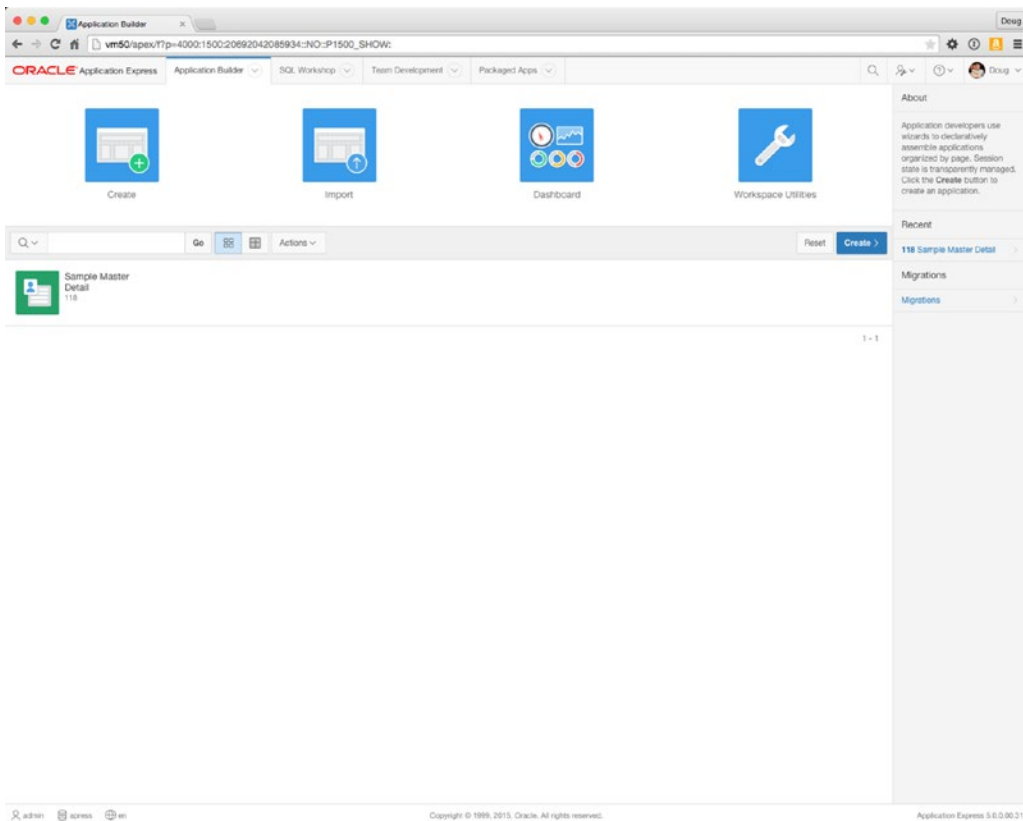
Figure 2-9. *The Application Builder home page*

Figure 2-9 shows one application in the workspace, named Sample Master Detail. However, there isn't much information about it other than its name and the application ID (118). This is where you begin to see the beauty of what APEX can do, not only in the developer UI, but also in your applications.

The list of applications you see is actually a style of report called an *Interactive Report* (IR). IRs allow us to customize how reports and their contents are displayed. IRs are used throughout the APEX development interface and can also be used when creating your own applications. They're extremely powerful tools, and you'll use them a lot.

On the right side of the page are three regions that show About information, recently edited applications, and a link to the Application Migration wizard. You will deal more with these later; for now, we will drill in to see the details of an application.

The Application Home Page

Clicking any one of the applications listed drills into the Application home page, as shown in Figure 2-10. This page is very similar to the Application Builder home page, but it shows all the pages in a specific application. Again, it uses an IR, so you can customize the way you see this data.

17
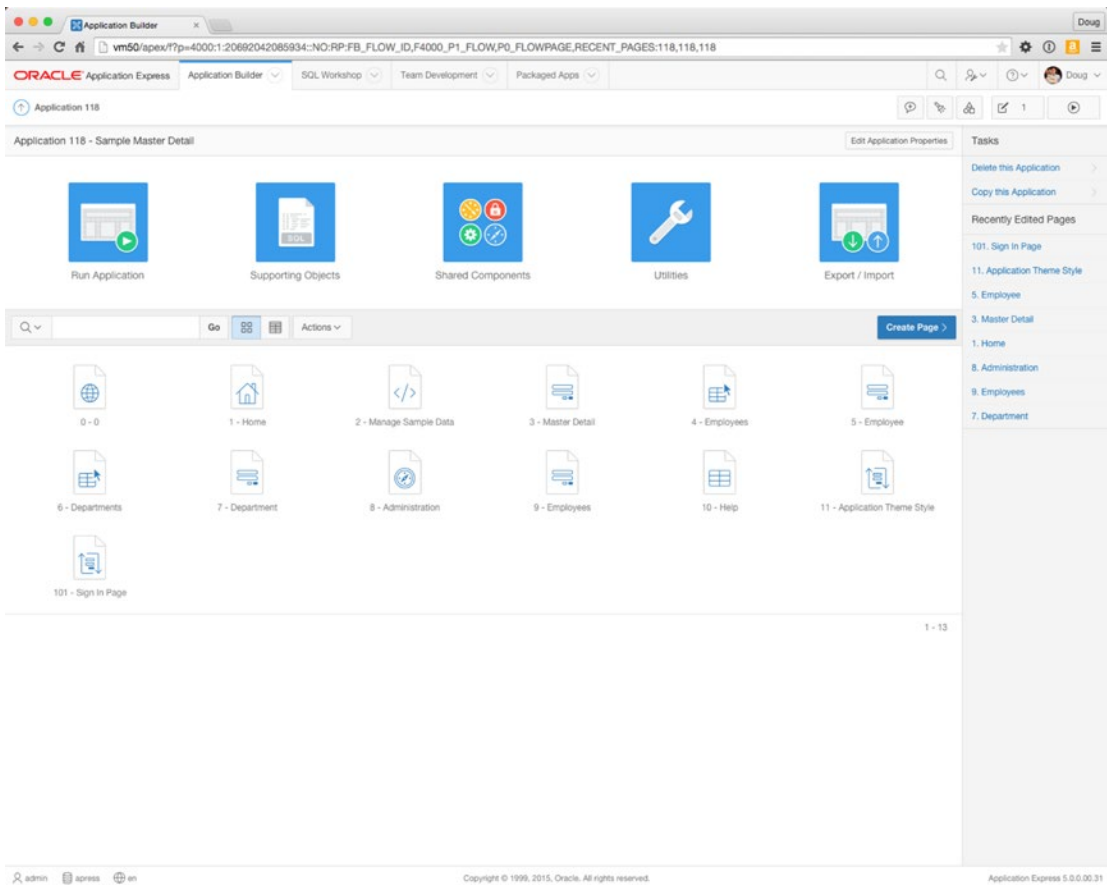
Figure 2-10. *The Application home page*

Notice the way the page is structured, with page-related tasks and recently edited pages presented along the right side of the page. This layout will become a familiar theme as you navigate through the interface.

From here, you can click any of the listed pages to edit that page using the Page Designer. You can also run, export, and import the application, edit the supporting objects or shared components, and access the application-related utilities.

The Page Designer

The Page Designer is where you'll be spending most of your time as a developer creating and editing pages, regions, and items. The Page Designer in APEX 5.0 is a complete departure from previous versions and is now presented in a way that much more closely resembles traditional Desktop IDE layout. This change has brought us the ability to manage components and edit their layout and properties from a single-page interface.

One of the biggest changes is that, due to the single-page interface, alterations to a page must now be explicitly saved. While this may seem disruptive, it actually brings with it some useful functionality. For instance, now multiple changes can be made and saved all at once, potentially reducing development time. Also, unsaved changes can now be easily undone.

Another major time-saving feature is the ability to select multiple components on a page using `Shift+Click` (or `Cmd+Click` on Mac). Once multiple items are selected, you can edit their common properties in the property editor. This can be useful if, for instance, you want to edit the attributes of all buttons on a page to set their visual properties to all be the same.

Region and item placement has been enhanced with the introduction of a drag-and-drop interface. All rendering components can be easily placed or rearranged on the page.

The layout of the new Page Designer is quite in-depth and, if you're not familiar with it, can potentially be a bit perplexing. Appendix A at the back of this book will give you a detailed tour of the Page Designer and its components, laying out the nomenclature that will be used through the rest of this book. Take a moment to thumb through Appendix A to familiarize yourself with the terms and the placement of the tools.

It is my goal for the rest of this book to take you through the development process in a way that will help you naturally learn how to use the Page Designer. However, if you're ever confused by an instruction or forget what a particular tool is called, referring to Appendix A should help clear things up.

SQL Workshop

The SQL Workshop is a suite of tools that provides developers with the ability to view and manage database objects in the underlying schema(s) assigned to the workspace. The SQL Workshop home page shown in Figure 2-11 lets you access each of the underlying tools and gives some high-level information about recently created objects and commands that that have been run.
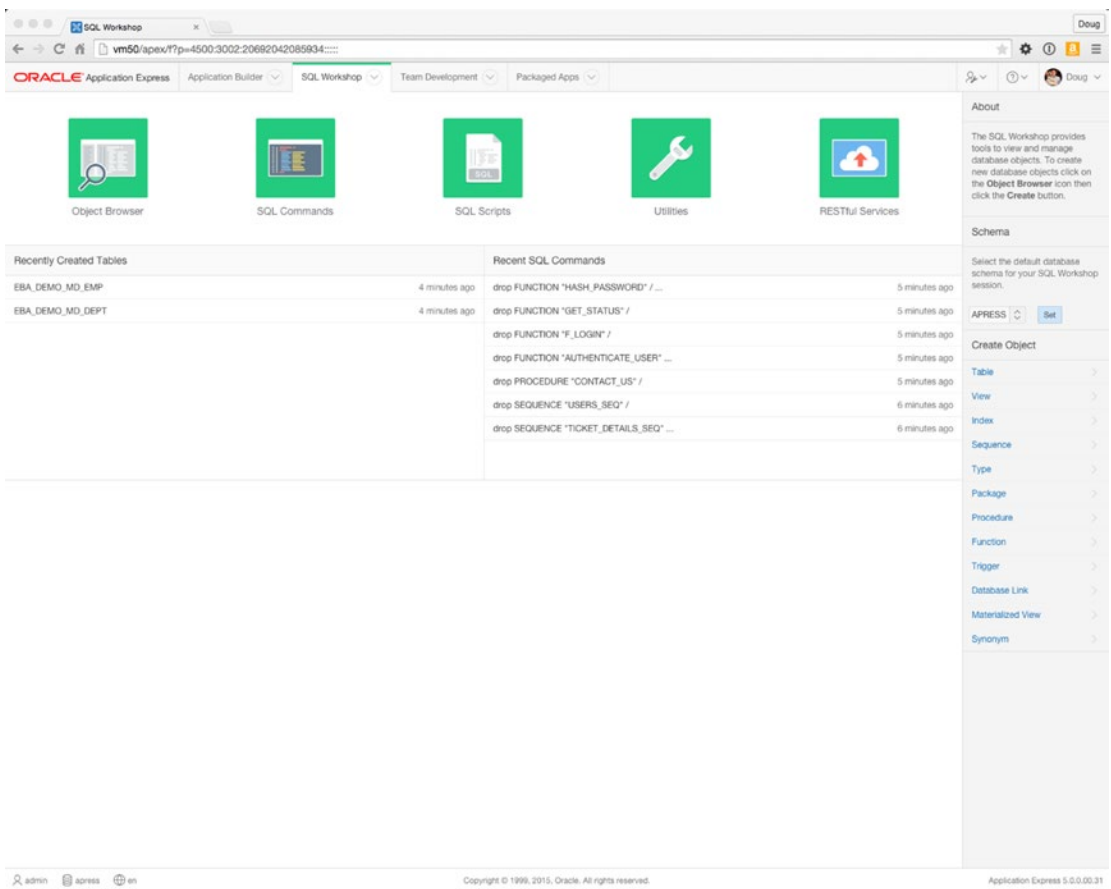
Figure 2-11. *The SQL Workshop home page*

Because there may be more than one schema assigned to the workspace, a schema-selection dialog at right allows you to select and set the default schema for all the tools. You may change the schema you're working in within each of the tools as well.

The main tools available as part of the SQL Workshop are displayed in the toolbar at the top of the page. Each of the individual tools deserves its own introduction, so let's spend some time now looking at what they are and what they can do. You'll use this area of APEX more heavily when you create the database objects for your application.

The Object Browser

If you've been working with databases for any length of time, you've probably used one of the more popular GUI tools that allows you to browse and manage database objects in a schema. The APEX Object Browser is a very similar tool presented through your web browser. Figure 2-12 shows the Object Browser being used to examine the table EBA_DEMO_MD_DEPT.
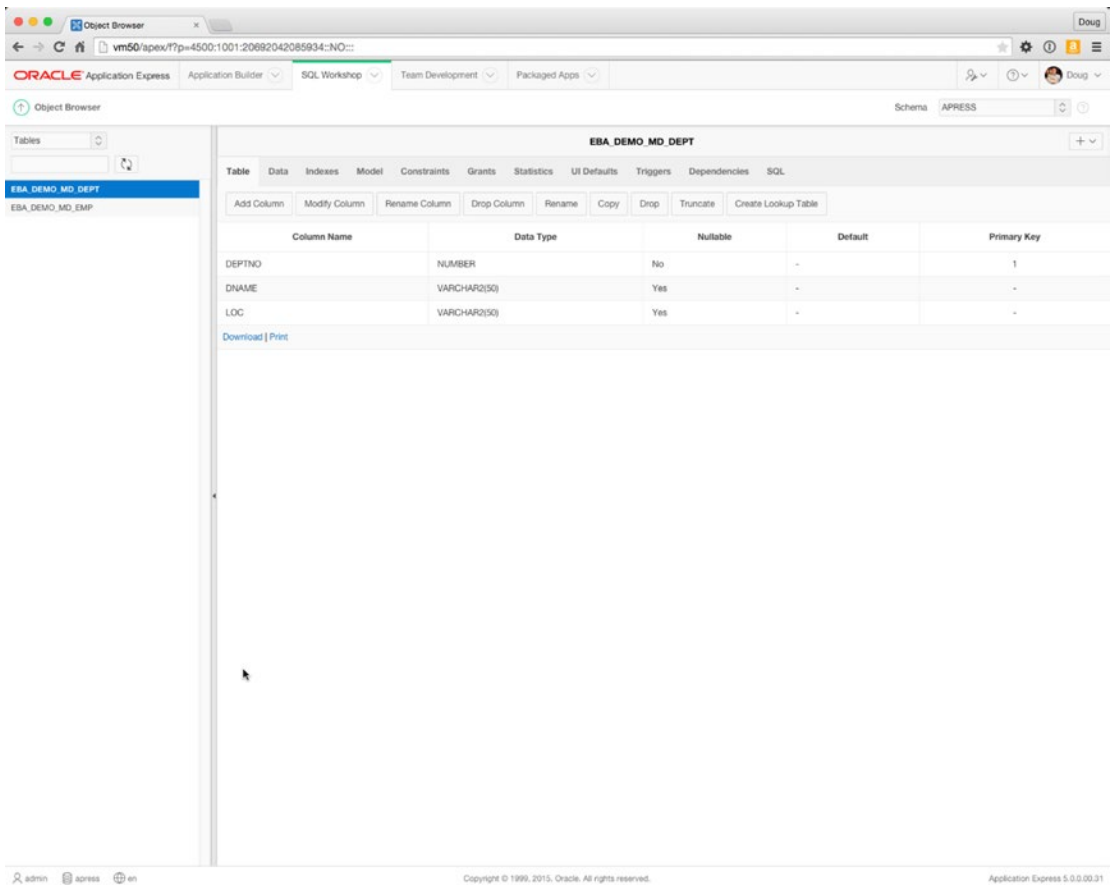
Figure 2-12. *The APEX Object Browser*

The name *Object Browser* is somewhat of a misnomer, because the tool can be used not only to browse the objects in the underlying schema(s), but also to create new objects, browse and edit data, delete objects, and edit object definitions. Although there are some limitations on the types of objects it can manipulate, it's powerful enough to do most of the daily tasks that an application developer needs to tackle.

You choose the object type you want to work with by selecting it from the drop-down list in the upper-left corner. You can search the selected object type by entering a text string in the search box just below it and clicking the refresh icon to the right. Clicking the name of an object displays its properties along with links to drill into more details.

Although the interface for the Object Browser is pretty intuitive, there are some interesting things to note. In the upper-right corner is a drop-down list that allows you to set the current schema. The list contains all schemas currently assigned to the workspace. You can switch between them simply by choosing a new one from the list.

The SQL Commands Interface

The SQL Commands interface allows you to interact with the underlying schema(s) using standard SQL commands or PL/SQL as you would in any other GUI tool or in SQL*Plus. The difference is that you can save the statements for use at a later time. Figure 2-13 shows a simple SQL statement as executed in the SQL Commands interface.

21
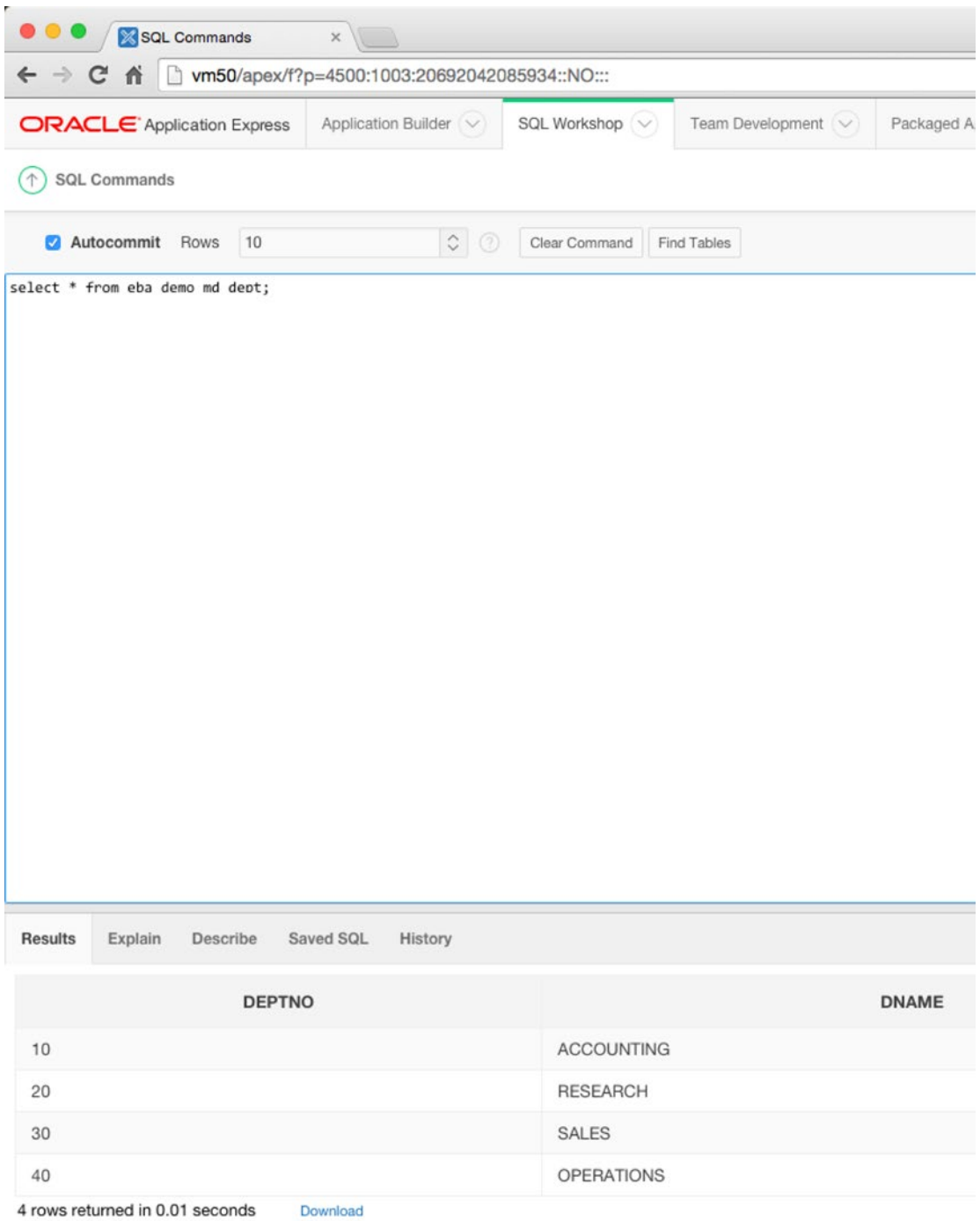
Figure 2-13. *The SQL Commands interface*

Although its core function is quite straightforward, the SQL Commands interface is more robust than it first appears. Beyond the ability to save and retrieve SQL and PL/SQL, it can also run explain plans on statements and allow you to view your statement history. Therefore, if you ran a script or statement that was particularly useful, but you forgot to save it, you still have the ability to retrieve it from the history buffer.

The SQL Commands interface also integrates with the Query Builder (described later), allowing you to load and manipulate saved statements that were built in the Query Builder.

■ **Note**   By default, all SQL statements executed via the SQL Commands interface are automatically committed. To override this setting and enter into transactional mode, uncheck the Autocommit check box in the toolbar. Once this is done, you can manually both commit and roll back your SQL statement.

There is no way to turn off Autocommit permanently, so you need to remember to do this any time you want to enter transactional mode.

SQL Scripts Interface

The SQL Scripts interface allows you to manage and run sets of SQL commands that are saved into script files. A single script can contain one or more SQL statements or PL/SQL blocks. SQL scripts coded outside of APEX can be loaded into the SQL script repository and edited or run from there. You can also create SQL scripts from scratch using the SQL Scripts interface. Figure 2-14 shows the main SQL Scripts interface page.
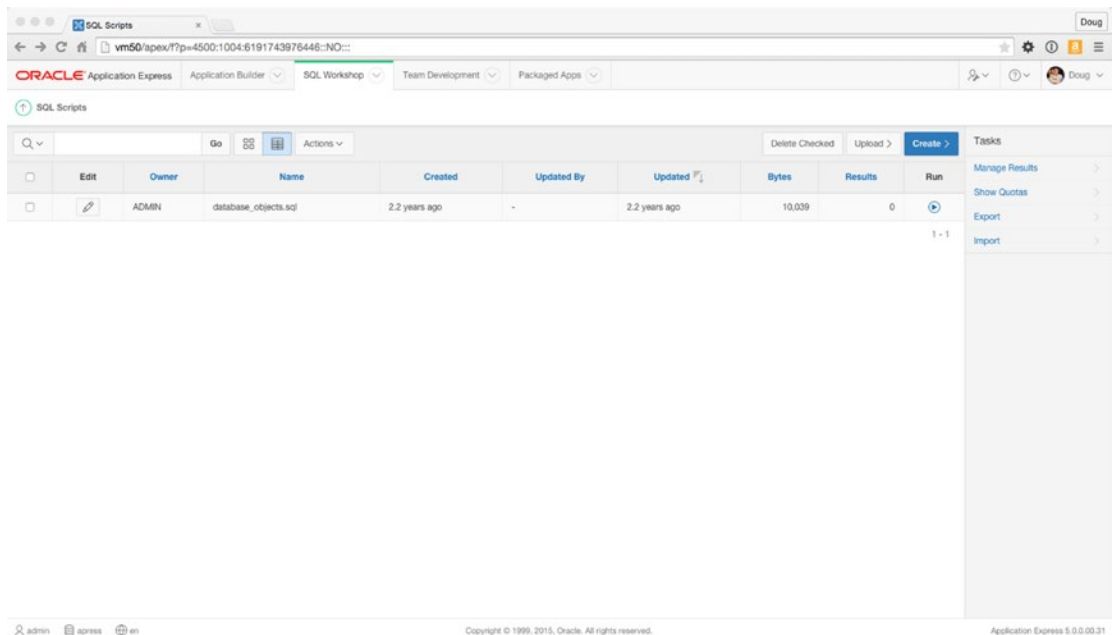
Figure 2-14. *The main SQL Scripts interface page*

23

In this example, one script, called database_objects.sql, is loaded into the script repository. By clicking the Edit icon, you can edit the contents of the script, as shown in Figure 2-15. Helpfully, APEX provides syntax highlighting in the Script Editor. The editor also has a Find and Replace function and autocomplete, as well as undo and redo capabilities.

You can also download the script to a local file so you can edit it in your favorite local text editor. When you're done, simply cut and paste it back into the editor or upload it as a new script file.
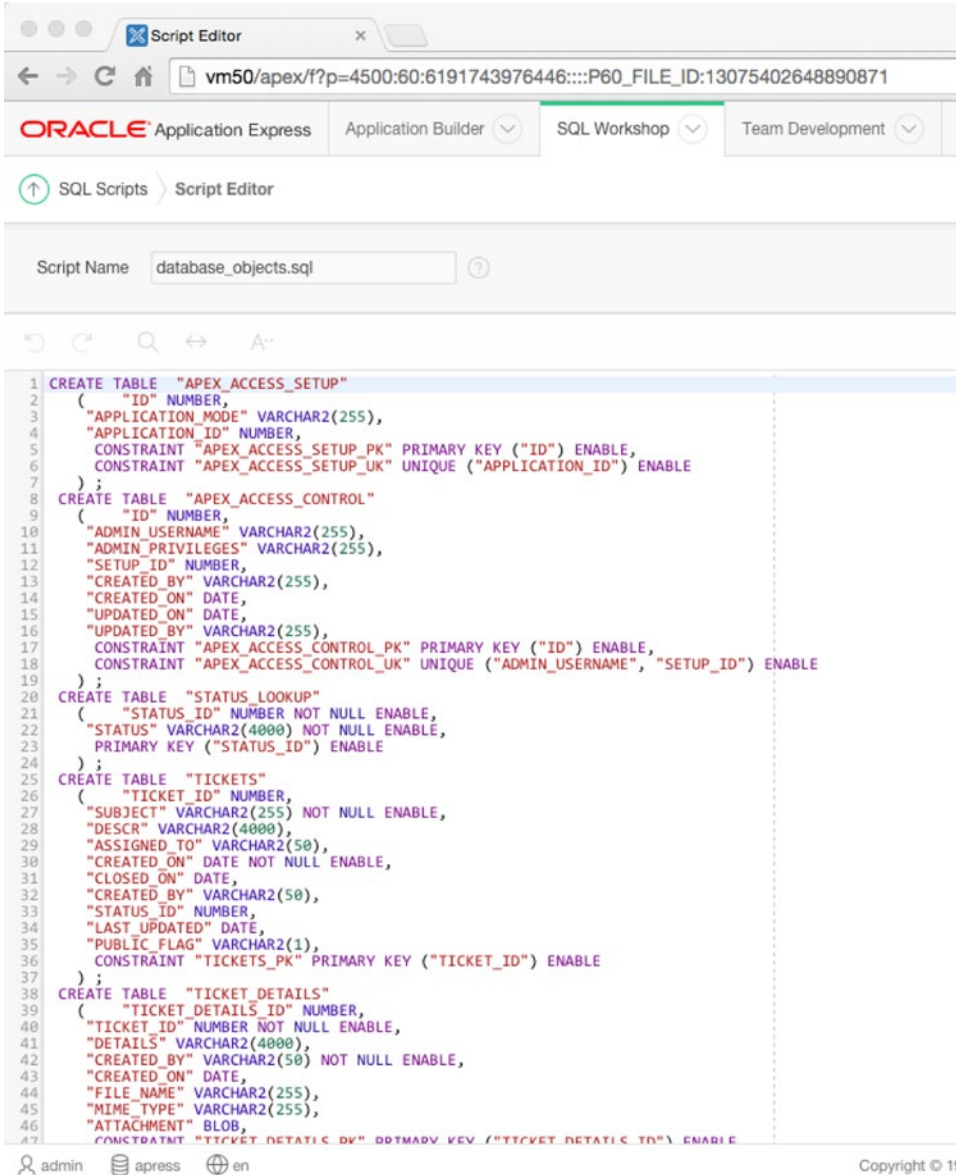
Figure 2-15. *The SQL Script Editor*

■ **Note**   When you upload a script file to the repository, the name of the script must be unique. You can't overwrite an existing script file of the same name with a new version without first deleting the existing script from the script repository.

Once a script is ready to run, you can click the Run icon in the list (or the Run button in the editor), and you're stepped through the Run Script wizard. This allows you to choose whether you want to run the script immediately or run it in the background. If you choose Run in Background, your script is entered into a queue, and it is executed when it reaches the front of the queue.

Either way, you're taken to the Manage Script Results page of the SQL Scripts interface, as shown in Figure 2-16. This screen allows you to see the status and certain high-level details of the script's execution. In the case of scripts that have been submitted in batch mode, you can also see the status of specific scripts in the queue.
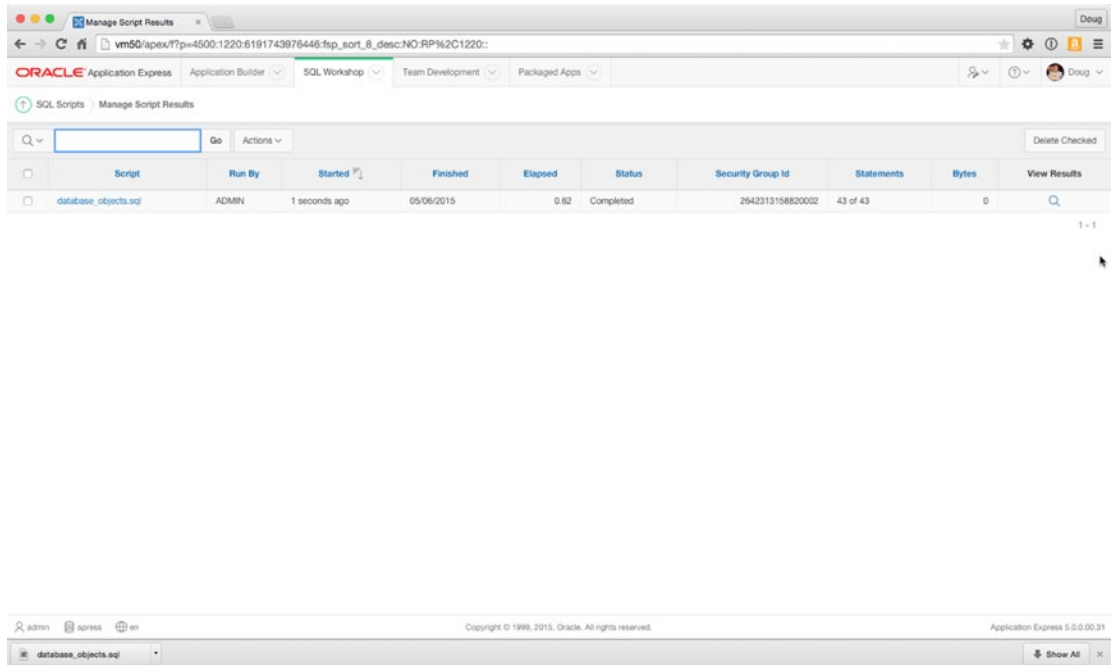
Figure 2-16.   *The Manage Script Results page*

Clicking the View Results icon shows you the final results of running the script. In Figure 2-17, you can see that the script had errors, the details of which are displayed in the body of the report. If the script were successful, no errors would be shown, and the statement results at the bottom of the page would show zero errors.

	Feedback	Rows
	Table created.	0
	Table created.	0
	Table created.	0
	Table created.	0
	Table created.	0
	Table created.	0
	Table altered.	0
	Table altered.	0
	Table altered.	0
	Function created.	0

row(s) 1 - 10 of 43   Next ▶

1

With Errors

l.   Application Express 5.0.0.00.31

⬇ Show All   ×

Figure 2-17. *An example of errors from the SQL Scripts interface*

■ **Note**   Although both the SQL Commands and the SQL Scripts interfaces can accept and run standard SQL statements, the extended commands of SQL*PLUS aren't valid in these tools.

The SQL Commands interface throws an error when it encounters any SQL*PLUS-specific commands. However, the SQL Scripts interface warns the user of the existence of SQL*PLUS commands in a script being run and then ignores them if the user chooses to continue. Because of this, the SQL Commands and SQL Scripts interfaces can't perform many of the functions of extended SQL*Plus scripts.

The Query Builder

Although the Query Builder has been relegated to the Utilities page, it merits discussion specifically because it's helpful to beginners. The Query Builder allows you to build SQL select statements using a more graphical interface, and although it's not quite drag and drop, it's fairly intuitive.

When you first enter the Query Builder, you're presented with a screen that lists all the tables and views available in the currently active schema. Figure 2-18 shows the initial Query Builder screen.
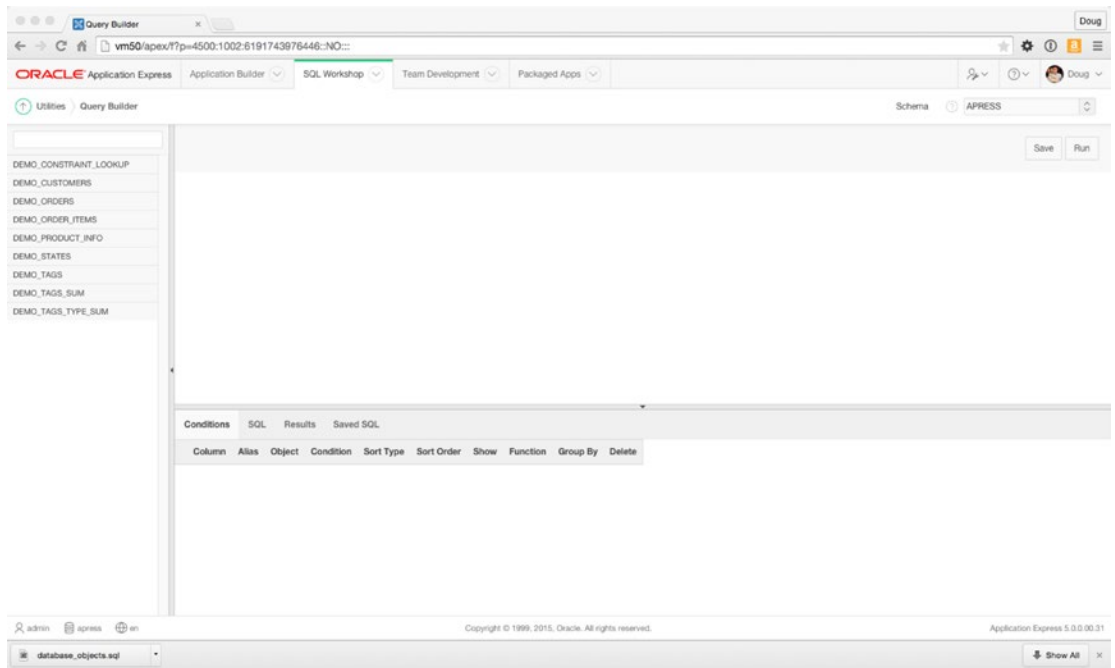
Figure 2-18. *The initial Query Builder screen*

From here, you can begin to build your query. To include a table in your select statement, simply click it in the list to the left. A representation of the table is placed in the blank region of the screen above the Conditions region. You may add as many tables as you like to your query, and can even include the same table more than once by clicking it again. Notice that if you include more than one instance of the same table, the new instance is suffixed with a sequence number differentiating it from the original table.

Figure 2-19 shows an example graphical representation for the `DEMO_ORDERS` table and outlines the different interactive features.
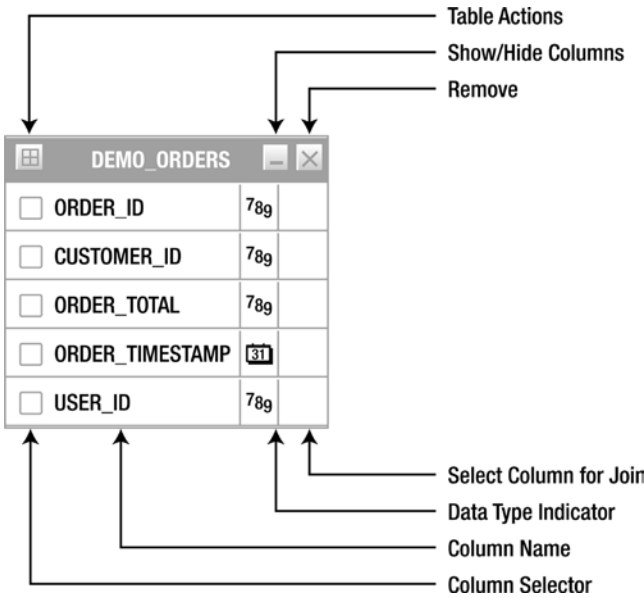
Figure 2-19. The `DEMO_ORDERS` table as represented in the Query Builder

Taken from top to bottom as they appear in Figure 2-19, these action areas are as follows

- *Table Actions* displays a dialog allowing you to do one of several things:

  - *Check All* allows you to quickly select or deselect all columns of the object for inclusion in the query being built.

  - *Add Parent* allows you to select and add a parent table, as defined by foreign-key relationships, to the Query Builder.

  - *Add Child* allows you to select and add a child table, as defined by foreign-key relationships, to the Query Builder.

- *Show/Hide Columns* expands and collapses the object so the column definitions are shown or hidden.

- *Remove* deletes the table and any of its related clauses from the `select` statement.

- *Select Column for Join* is activated by clicking the blank square next to a column name. Doing so darkens the square and puts the Query Builder into Table Link mode. Then you can click another blank square, either in another table or in the same table, and the Query Builder inserts an `EQUALITY where` clause between the two columns in the SQL statement.

28

- *Data Type Indicator* indicates the data type of the column, such as `number`, `character`, `date`, and so on.

- *Column Name* indicates the column name as defined in the table description.

- *Column Selector* allows you to individually select or deselect columns to be included in the SQL statement for processing. This may also include columns that you want to use in the `where` clause but not display in the output of the SQL statement. The basic rule is that you need to select all the columns you want to display, but you don't necessarily have to display all the columns you select.

As you add and join tables and select columns to operate on, the region at the bottom of the screen begins to change. This region is subdivided into several tabs, as follows:

- The *Conditions* tab shows one row for each column selected in the area above and allows you to further define its attributes. (More on this feature in just a moment.)

- The *SQL* tab displays the SQL statement as the wizard builds it. Although it's not directly editable, you can easily highlight the statement and copy it to the clipboard from here.

- The *Results* tab shows the results of running the SQL statement and allows you to download the resulting data in CSV format.

- The *Saved SQL* tab allows you to save, recall, and manage statements that have been built with the Query Builder. There are also filters that allow you to search and limit which saved queries are displayed.

All but the Conditions tab are self-explanatory, so let's go over this one in a little more detail. Figure 2-20 shows an example two-table join, with five columns selected to operate on.

	Column	Alias	Object	Condition	Sort Type	Sort Order	Show	Function	Group By	Delete
▲ ▼	USER_NAME	USER_NAME	DEMO_ORDERS		Asc		☑		☑	✕
▲ ▼	CUSTOMER_ID	CUSTOMER_ID	DEMO_CUSTOMERS		Asc		☑		☑	✕
▲ ▼	CUST_FIRST_NAME	CUST_FIRST_NAME	DEMO_CUSTOMERS		Asc	2	☑		☑	✕
▲ ▼	CUST_LAST_NAME	CUST_LAST_NAME	DEMO_CUSTOMERS		Asc	1	☑		☑	✕
▲ ▼	ORDER_TOTAL	SUM_OF_ORDERs	DEMO_ORDERS	<500	Asc		☑	SUM	☐	✕
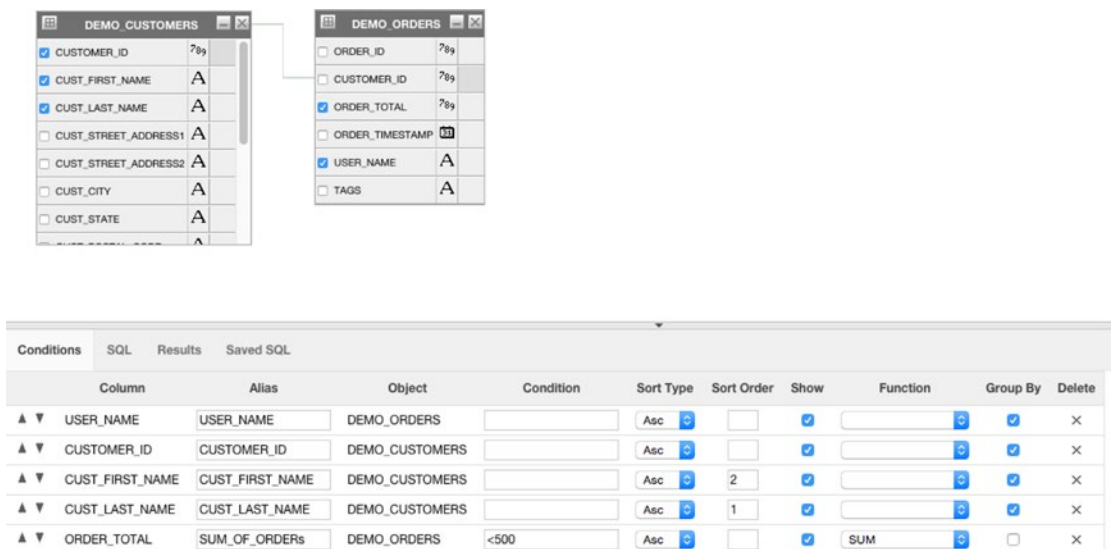
Figure 2-20. *An example two-table join*

In this example, the following modifications have been applied to the query:

- Changed the alias of the ORDER_TOTAL column to SUM_OF_ORDERS

- Limited the result set to only those records where ORDER_TOTAL is less than 500

- Sorted the records returned by CUST_LAST_NAME, CUST_FIRST_NAME ascending

- Performed a SUM function on the ORDER_TOTAL column

- Grouped the query by USER_NAME, CUSTOMER_ID, CUST_FIRST_NAME, CUST_LAST_NAME

Based on the column selections as well as the restrictions and changes introduced in the Conditions tab, the SQL statement (as it appears in the SQL tab) looks like this:

```
select DEMO_ORDERS.USER_NAME as USER_NAME,
       DEMO_CUSTOMERS.CUSTOMER_ID as CUSTOMER_ID,
       DEMO_CUSTOMERS.CUST_FIRST_NAME as CUST_FIRST_NAME,
       DEMO_CUSTOMERS.CUST_LAST_NAME as CUST_LAST_NAME,
       sum(DEMO_ORDERS.ORDER_TOTAL) as "SUM_OF_ORDERs"
  from DEMO_ORDERS DEMO_ORDERS,
       DEMO_CUSTOMERS DEMO_CUSTOMERS
 where DEMO_CUSTOMERS.CUSTOMER_ID=DEMO_ORDERS.CUSTOMER_ID
   and DEMO_ORDERS.ORDER_TOTAL <500
group by DEMO_ORDERS.USER_NAME,
         DEMO_CUSTOMERS.CUSTOMER_ID,
         DEMO_CUSTOMERS.CUST_FIRST_NAME,
         DEMO_CUSTOMERS.CUST_LAST_NAME
order by DEMO_CUSTOMERS.CUST_LAST_NAME ASC,
         DEMO_CUSTOMERS.CUST_FIRST_NAME ASC
```

Although the Query Builder is very useful and allows you to put together a basic query fairly quickly using a simple GUI, it does have its limitations, such as nested subqueries and complex unions. We can use the Query Builder to get the skeleton of a query defined; we can then take the query to the SQL Commands window or a SQL IDE and fine tune it from there.

As a final note, it's worth mentioning that the Query Builder is linked to from several places in APEX, so any time you're prompted for a SQL statement (for example, as the basis for a report), you can open the Query Builder in a pop-up window and return the query to the calling form.

Utilities

The SQL Workshop Utilities section gives you access to tools and reports that help you view and manage information about the underlying database objects and their data. This section introduces each tool set and its main purpose. However, the majority of these tools are very straightforward, so in most cases the deep details are left for you to explore on your own.

The Utilities home page (as shown in Figure 2-21) presents a quick, icon-based menu you can use to reach the individual utility areas. Clicking any one of these icons will take you directly to the tools page for that category.
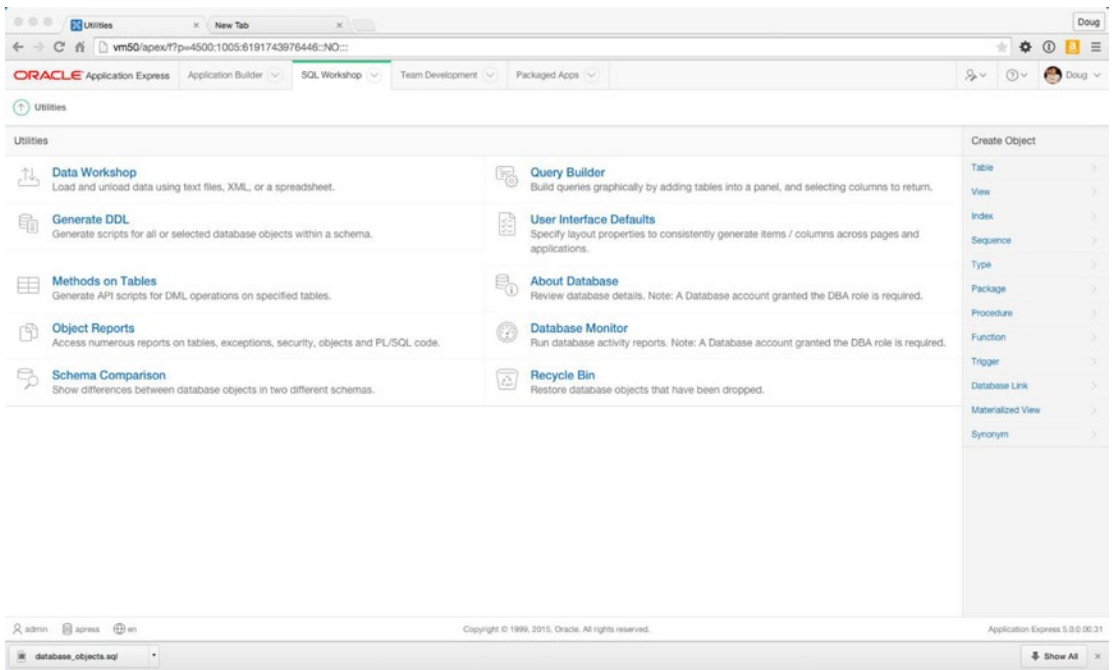
Figure 2-21. *The SQL Workshop Utilities home page*

You've already seen the Query Builder, which gives users the ability to visually create queries.

The *Data Workshop* provides tools that import and export data in many different formats, including comma- or tab-separated data, XML data, or spreadsheet data. These tools also help you manage files that you have loaded into either the text or spreadsheet repository.

The *Generate DDL* wizard allows you to choose a schema associated with the workspace and then generates a script that can be used to re-create some or all of the objects with that schema based on your selection. The generated script doesn't include any insert statements for the data that resides in the database objects, but it's a good way to easily re-create the underlying objects an application might use.

The *Methods on Tables* wizard generates an Application Programming Interface (API) based on a specific table or set of tables. For each table selected (up to ten named tables), the generated package contains a procedure for each of the following actions: Insert, Update, Delete, and Select. The benefit of using table APIs instead of accessing the table directly is that any required validation logic can be included once, in the API, and accessed from various alternate interfaces including APEX.

The *Object Reports* are actually a set of utilities that let you get detailed information about the different types of objects that live in the "parse as" schema(s) assigned to the workspace. Although most of the reports have to do with tables, others deal with PL/SQL objects, invalid objects, grants and permissions, and so on. This is a good place to find details of the objects in your working schema.

The *Schema Comparison* utility allows you to compare the objects in two separate schemas and create a difference report. You may choose to compare only certain attributes or all attributes of the objects in the selected schemas. The limitation here is that both schemas must be assigned to the workspace in order for the comparison to take place.

User-Interface Defaults allow you to define default display attributes for APEX regions and items. The utility lets you manage these UI defaults at two different levels: Table Dictionary and Attribute Dictionary. UI Defaults will be discussed in more detail later.

About Database and *Database Monitor* are special utilities that require the user running them to have access to a database login that has been granted the DBA role. The Database Monitor utilities allow the privileged user to view Sessions, Systems Statistics, Top SQL, and Long Operations reports. The About Database report shows detailed information about the database instance and the APEX environment. Depending on the settings the Instance Administrator has chosen, these two utilities may not appear in the list, because they can be turned off.

When an object is dropped, Oracle doesn't immediately remove the space associated with the table, but instead renames the table and places it and its associated storage in the *Recycle Bin*. The Recycle Bin utility allows you to view and potentially recover objects that have been dropped from the schemas associated with a workspace. You may also purge the Recycle Bin, allowing the space to be reclaimed by the Oracle database for use somewhere else.

Packaged Apps

The Packaged Apps section is where you will install and managed the applications that are bundled with the APEX distribution. The main page, shown in Figure 2-22, shows the Packaged Apps home page. From here you can see which applications are installed and can navigate to the three subsections.
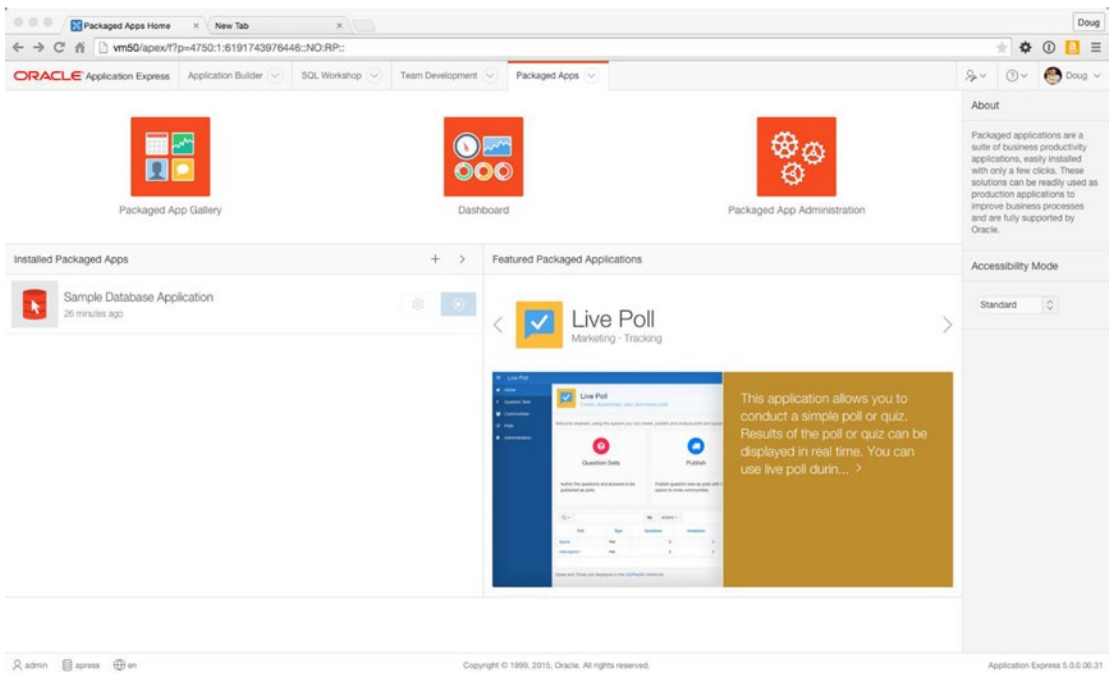
Figure 2-22. *The Packaged Apps home page*

Packaged App Gallery

The Packaged App Gallery presents all of the applications that come bundled with the APEX distribution. There are 35 separate packaged apps that can belong to several different categories, including Software Development, Tracking, Team Productivity, Marketing, Knowledge Management, IT Management, Project Management, Sample Application and Sample Websheet.

Clicking on the icon of an application takes you to a detailed information page for that application. Here, you'll be able to see a screenshot of the application, read its full description, and see version information, as seen in Figure 2-23.

Figure 2-23. *The Checklist Manager Packaged Apps information page*

Clicking the Install Application button will step you through the process of installing the selected application in your current workspace. A pop-up installation wizard will present you with a choice of authentication method, which normally defaults to Application Express Accounts. Clicking the final Install Application button in the wizard will install the application and any of its supporting objects, including any required database objects. Once the installation is complete, you will be taken back to the Application's Information page where, as shown in Figure 2-24, you will see the application has been successfully installed and you will be given the options to manage and run the application.
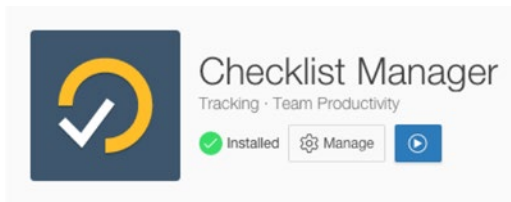
Figure 2-24. *See that the Checklist Manager App has been successfully installed*

There are a few things that you should know about Packaged Applications:

- Any application that has "Sample" in the name is there to demonstrate functionality available in APEX and therefore will be installed in an *unlocked* state by default. This means that developers will be able to edit the application and see how the APEX team developed the app.

- Applications that *do not* have "Sample" in the name are provided as production ready and will be installed in a *locked* state that does not allow any editing until the app is specifically *unlocked.* This can be done via the Manage button, as shown in Figure 2-24. Any of these applications that are installed and remain locked are fully supported by Oracle in a production environment. These locked applications can also be upgraded to more current versions that may come with future versions of APEX. The moment you unlock them, all support from Oracle ceases, upgradeability expires, and there is no way to re-lock the application.

- All Packaged Applications are installed into the workspace's default "parse as" schema. Currently, there is no direct way to install them in a secondary "parse as" schema without first unlocking and exporting the application, thereby voiding any support.

Even though the Sample apps were written as learning aids, there is a lot to be learned from many of the production-ready applications as well. I heartily suggest your first act after finishing this book is take a look at the inner workings of some of the Packaged Applications.

Packaged App Dashboard

As shown in Figure 2-25, the dashboard page presents an overview of the utilization of Packaged Applications in the current workspace. You're presented with the total number of available apps, the number installed, and whether there are any applications that can be upgraded. You can also see who has installed the apps and how frequently they are used.
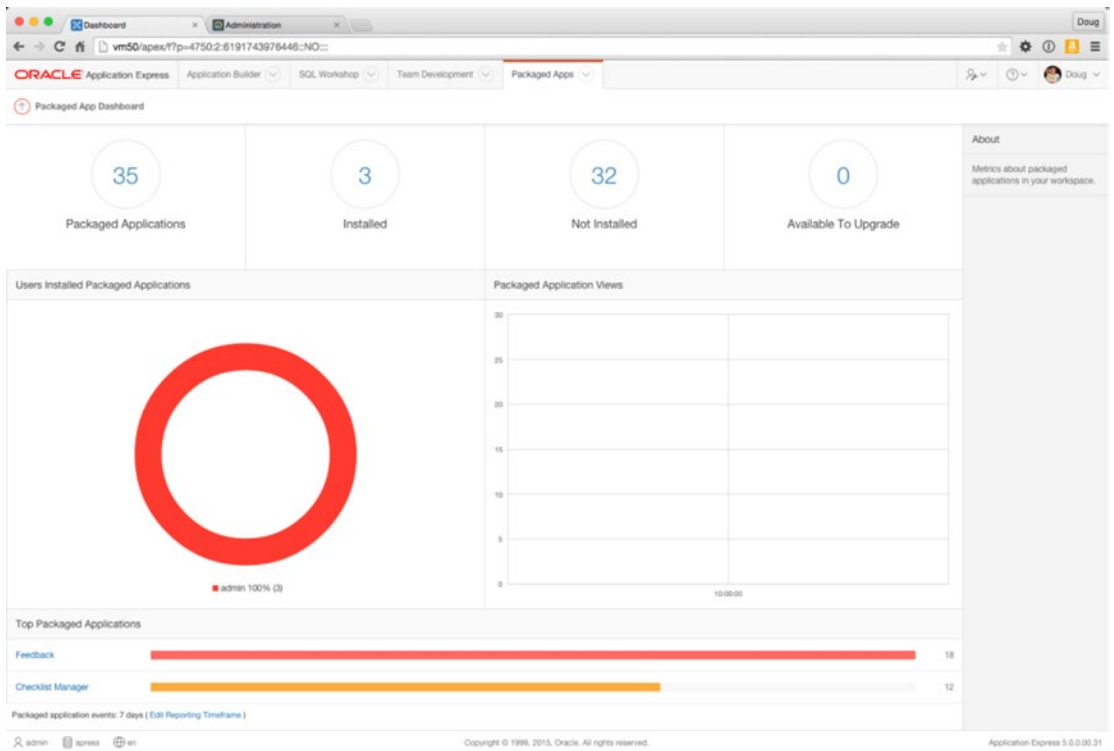
Figure 2-25. *The Packaged Apps Dashboard*

Packaged App Administration

The Packaged App Administration page provides a list of administration tasks specifically related to Packaged Applications installed in the current workspace. If you are logged in as a developer, you'll only see options relating to managing Interactive Report settings and Activity reports. However, when logged in as a workspace administrator, you'll see a section called *Manage Services* that shows a small subsection of what is available to you in the full Administration section.

Administration and Team Development

The last two functional areas of the UI, Administration and Team Development, are complex enough to truly deserve their own chapters. Therefore, we refer you to the chapters that cover these areas in depth. Chapter 10 covers deploying applications, Chapter 14 is about managing workspaces, and Chapter 15 goes over Team Development.

You will dip into administrative tasks throughout this book, so if you want to have a full understanding of administration before you start, you should take a detour and read these chapters now to get a good foundation. However, if you're prepared to learn on the fly, go to the next chapter, where you start the real programming.

Summary

The architecture of APEX may seem a bit daunting at first, but once you actually start working with it, things will begin to fall into place, and you'll understand more and more about how everything fits together. If you take away only one thing from this chapter, let it be that a workspace is essentially your development sandbox. Everything you do happens in the context of a workspace. Everything else—from a development standpoint—is much like any other development environment. Are you building a new application? Then it needs to be created in a workspace. Do you need access to a schema to build that app? Then it needs to be assigned to your workspace. You get the picture. Now, on to the fun!

■ ■ ■

Identifying the Problem and Designing the Solution

Every computer system is (or at least should be) the result of solving some type of problem. Although "Hello World" apps are great, I firmly believe that the best way to learn any technology is to apply it to a real problem and see how things actually work.

I adhere to that principle throughout this book. This chapter will discuss a very common problem in most organizations that can be solved technically. You will also look at some of the elements you need to consider when designing web-based systems in general and with APEX specifically.

Identifying System Requirements

Almost every company, no matter the size, will at some point need to implement some sort of help desk. Whether it's an internal one to track employee questions and problems or an external one to track client issues with commercial software or hardware, the basics of a help-desk system are fairly standard.

Most help-desk systems are driven by the notion of a *trouble ticket* or simply a *ticket*. This term is left over from the days before computers: most problems were reported over the phone, and troubleshooters used a physical paper ticket to log a call. The information contained on that paper ticket included a description of the problem, the name of the person having the problem, when the problem was logged, and so on. Then, throughout the process of troubleshooting and, hopefully, solving the problem, the engineers wrote down each step of the process and included any documentation of the problem they gathered along the way. Today, it would be very surprising to see a help-desk system that wasn't computerized, even if it's only a spreadsheet of issues with notes and statuses.

In this chapter, you will attack the help-desk system with APEX. Before you dive in, you need to clearly understand the problems you're trying to solve. If nothing else, you need to review the current system.

Never a Clean Slate

Almost no computer system written today starts from scratch. There is almost always something in place already, even if it's just some loose guidelines or ideas.

For this example, let's say your company has a very basic system in place, but it's no longer meeting the needs of your growing user community. Your goal is to create a new system that will make the logging of issues and their solutions much easier for everyone involved; however, to do that, you must understand the needs of the users and the functionality of the system that is currently in place.

A Broken System

In general, the users of help-desk systems can be categorized into two groups: people who log problems (end users) and people who help solve the problems (technicians). Depending on which user community you fall into, it's likely you have different needs, but, overall, the system should help the end users and the technicians communicate with each other about the problem or issue.

The first step is to understand how your help desk is being managed today and why it's not working. Speaking to both the technicians and the end users can provide a huge amount of information, but the challenge is that this information usually comes in the form of complaints about the current system.

Quizzing the end users reveals that their main complaint is that they never know the status of the problems they've logged. They can go days, sometimes weeks, without communication from the technicians, and in the eyes of the users, no communication means no one is working on their problem. Another user complaint is that the help-desk technicians often don't know how to contact them to ask further questions or communicate progress.

On the other end of the issue, the technicians are overloaded. Ticket information is kept in an Excel spreadsheet. Originally, the help desk was only one person, but now there are several technicians working independently. While performing their daily duties, each needs to update the spreadsheet with information regarding the tickets assigned to them. The increasing number of people accessing a single spreadsheet causes problems, because only one person can open and update the spreadsheet at any given time. The technicians are also tired of constantly being called by users wanting an update on the status of their issues.

It's obvious that the system is broken. Neither the users nor the technicians are happy about the situation. It's your job to take the information you've gleaned from these conversations and design something that will address the needs of both user communities.

How Do You Fix Things?

With the information you've gathered so far, you can now define some loose requirements and break them down by user type to give you a much clearer understanding of what each community needs. Then, from those requirements, you can begin to think about the database design that you'll need to create in support of them.

Defining the Requirements

You can look at requirements from two perspectives. End users have one set of requirements and technicians another. Some requirements overlap between the two groups. Others are unique to one group or the other.

End users should be able to

- create a new ticket outlining their problem

- see the status and progress of tickets

Technicians should be able to

- easily identify and view new tickets

- easily identify which tickets are directly assigned to them

- search existing tickets

- create new tickets on behalf of an end user

- assign tickets to other technicians

- add details (comments, information, and attachments) to tickets

- update the status of a ticket

Although you could go a lot further, these requirements form the basis of a pretty complete help-desk system. You can always add functionality to it later when you have a better understanding of what else the users and the company might need.

Extrapolating to a Database Design

Having stated the requirements, you can begin to extrapolate the database objects you need to create to store the data. If you're new to database design, here's a quick trick to help you identify the entities for which you need to build tables: go back through your requirements and look for *concrete nouns* that represent the highest-level objects you need to track. As you find these nouns, try to identify if they're actually at the highest level or if they're merely attributes of something bigger.

If you follow the described process with your brief requirement specification, the nouns USER and TICKET jump out as being the two main things you want to track. It's tempting to split users into two different sets—technicians and end users—but the type of user is merely an attribute of a user.

An object that is a little harder to identify is TICKET DETAIL. It's completely valid to think that this would merely be an attribute of a TICKET; however, the clue comes in the fact that you can't concretely identify how many TICKET DETAIL entries there will be for any given TICKET. The fact that the number is unknown indicates that you should create a table that is a child of the TICKET entity called TICKET DETAIL. This way, you can enter as many detail records as you need.

So, you've identified three major entities: USERs, TICKETs, and `TICKET DETAILs`. You now need to think about the attributes of each of these entities and what type of data they will hold. Searching back through the statement of requirements, talking to the technicians about what they track today, and thinking about what types of things you'd want to be able to track during the process of solving a problem, you can identify a number of attributes about your objects. Tables 3-1 through 3-3 show these attributes.

Table 3-1. *USER Attributes*

Attribute Name	Type of Data	Comment
User ID	Number	A unique ID for each user
User Name	Text	A login ID for each user
Password	Text	The password used to log in to the system

Table 3-2. *TICKET Attributes*

Attribute Name	Type of Data	Comment
Ticket ID	Number	A unique way to identify the ticket
Subject	Text	A brief one-line statement of the problem
Descr	Text	A detailed description of the problem
Status	Text	The status of the ticket during processing (OPEN, PENDING, CLOSED, and so on)
Created By	Text	The user who logged the ticket
Created On	Date	The date the user created the ticket
Closed On	Date	The date the ticket was closed
Assigned To	Text	The technician who is assigned to work on the ticket

Table 3-3. *TICKET DETAIL Attributes*

Attribute Name	Type of Data	Comment
Ticket Details ID	Number	A unique way to identify this detail entry
Ticket ID	Number	Which ticket this detail is linked to
Details	Text	A text description of any details entered by the technician
Created By	Text	The user who logged the ticket
Created On	Date	The date the user created the ticket

Although it's good to try to be as detailed as possible as early as you can, you don't have to be perfect here. You can always go back and alter or expand the data you wish to capture as you identify other potential attributes.

System Design with APEX in Mind

Because APEX not only resides in, but is also built on, the Oracle database, you would think that designing database objects for APEX would be the same as designing for any other system that uses Oracle as a data store—and in some aspects you would be right. However, there are definitely some things you need to understand when designing for an APEX system that will make your life much easier.

Most of what you do with APEX, at least initially, uses a series of wizards. If the database objects are designed with APEX in mind, the wizards will do far more work for you; therefore, you'll need to do far less fine tuning manually. The following sections will discuss the most important design considerations and how they affect what the wizards do for you.

Table Definition and User-Interface Defaults

One such area you will see in more detail later is that of user-interface defaults (UI Defaults). It's important to know that when you use UI Defaults, certain table attributes are translated into default settings used across APEX. Here are some of the more far-reaching things you can do at the table level to help make UI Defaults more useful:

- Placing comments on a table column seeds that item's UI Default help text with the text of the comment.

- Marking a column as NOT NULL at the database level triggers a Required flag to be set in the UI Defaults.

- Date and Timestamp data types are set up to display as Date Pickers on input forms.

- The order in which the columns appear in the table is the default order in which the UI Defaults will set them to display on a form or report.

- Defining a column as a BLOB sets the form-level UI Defaults to use APEX's declarative BLOB functionality.

You will set up and modify UI Defaults in a later chapter so you can see for yourself how design decisions affect the way they are set up.

APEX and Primary Keys

APEX is set up to make the best use of sequence-based surrogate primary keys of no more than two columns. Although you can still use APEX on table structures that use multicolumn natural keys, it's far easier and you get much more out of the box if you give APEX what it likes.

I have worked with many systems over the years that implemented multicolumn natural keys, and I've successfully implemented APEX systems on top of these types of data structures. However, I ended up hand coding the logic that APEX would have provided for free had the structures used one- or two-column surrogate keys.

In APEX 4, the ability to use ROWIDs in place of primary keys was introduced to help solve the problem of multicolumn primary keys. This feature provides a way to bypass the perceived limitation of APEX's two-column primary-key limit by using the ROWID as the primary key.

Although using ROWIDs in this manner is technically and syntactically correct, when building an APEX application from scratch, it's still considered a best practice to use single-column surrogate primary keys based on a database sequence (Oracle 11g and below) and assigned by either database triggers or an identity column (Oracle 12c).

If you take the example of the TICKET table, the ID for a ticket is an arbitrary piece of data used only to uniquely identify one ticket from another. Therefore, it easily fits into the realm of a surrogate primary key. Even if the spreadsheet that the help-desk technicians currently use has IDs assigned to the tickets, you can load those values and start your sequence counting at a point above the highest current TICKET ID. The same is true for TICKET DETAILS. Even in the USER table, where you have a unique, single-column natural key (the User Name), it behooves you to implement a surrogate key so as to be able to take advantage of the built-in APEX code paths.

Business Logic vs. User-Interface Logic

Because it's primarily written in PL/SQL, APEX takes full advantage of everything that PL/SQL has to offer. The APEX development team has made thorough use of stored PL/SQL program units for their business logic, and you can take a very important lesson from them.

Although it's arguably a valid development method to prototype your business logic by first coding it as an anonymous PL/SQL block inside of APEX, it's foolish to leave it there long term. By moving it out into stored program units, you gain in many different ways.

One very important gain is made in the realm of performance. Anonymous PL/SQL blocks are stored in the APEX metadata as uncompiled PL/SQL code. Each time they're required to run, they must first be extracted from the APEX metadata, parsed, compiled, and then run. This process carries quite an overhead if the PL/SQL in question is part of a page that gets thousands or even hundreds of thousands of hits a day. If you move that code into a stored program unit in the database, the retrieval, parse, and compile steps are all skipped, and the code is run directly.

Another benefit is reusability. If the same logic is used in more than one place, it can simply be called instead of duplicated in two anonymous blocks. Therefore, any change to the business logic need only happen in one place. Another reusability benefit might occur if multiple systems (some being non-APEX) need access to the same business logic. When stored in a PL/SQL program unit, it doesn't matter whether the calling system is APEX, .NET, Java, or PHP–they can all use the same logic.

Finally, by moving business-logic code into stored program units, you gain the ability to code, debug, and test these program units outside of the restrictions of APEX, using your favorite PL/SQL coding tool instead. However, not all code needs to be moved out into the database. User-interface logic that manages and manipulates items on the page, such as computations, validations, and processes, is often best kept as part of the page. Such logic is often so page specific and so small in footprint that the gain from moving it out to the database isn't worth the extra management overhead. As a general rule of thumb, logic that controls or manipulates the UI is best placed in APEX, and logic that implements business rules or controls the data is best placed in stored program units in the database.

Placement of Database Objects

The Oracle database is very flexible, allowing data from multiple schemas to be granted to and queried by other schemas, even across database links. The APEX wizards have been coded to work best when the database objects reside in a "parse as" schema assigned directly to the workspace.

The APEX wizards make heavy use of database metadata for the objects in the "parse as" schema. If you're trying to create applications against synonyms from another schema or across a database link to another database, in many cases the wizards won't be functional, because the metadata for these objects is unavailable. Some features won't work at all, such as the management of BLOB data across database links.

In general, reports are much easier to deal with when it comes to disparate data, because you can supply a working query and create a report. Forms, however, become much more difficult, because the insert, update, and delete logic must be coded manually instead of relying on the APEX-supplied automated DML processes.

Although it's not always possible, the best practice is to create the underlying database objects in the "parse as" schema for the application. This is how you will architect your help-desk system.

Translating Theory to Practice

Now that you have a reasonable understanding of the things you need to think about when designing the database objects for your system, you can translate your text-based tables into a real schema definition. Although it's very easy to take the previously described objects and attributes straight to SQL Workshop and start entering their definitions, it's usually a good idea to go through the steps of creating an entity-relationship diagram (ERD). Often, the action of doing this can bring other design considerations to light.

There are dozens of ways to draw ERDs, from pen and paper to high-end database-design tools. However, I tend to take the middle ground and use Oracle's SQL Developer Data Modeler, a robust and free tool from Oracle.

Figure 3-1 shows the results of using the Data Modeler to create the ERD from the information in the initial definitions.
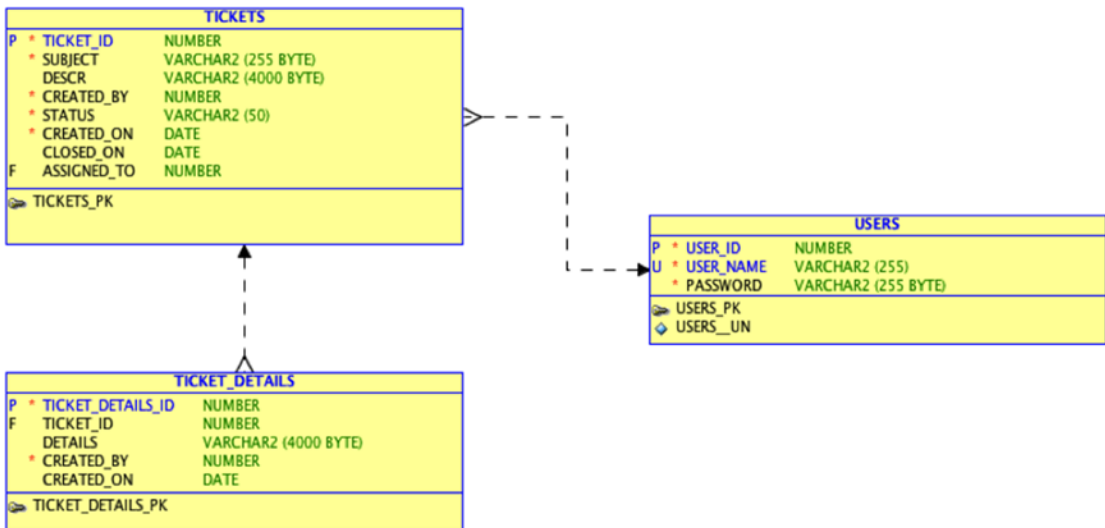

Figure 3-1. *First draft of database design*

The diagram shows each table having a surrogate primary key that uniquely identifies the records. As discussed in the previous section, this allows the APEX wizards to work more seamlessly and generate more-complete objects.

There is a foreign key in place between TICKETS and USERS to identify the person to whom the ticket is currently assigned. In addition, a unique constraint is placed on the USER_NAME column of the USERS table to make sure someone doesn't accidentally create two users with the same USER_NAME.

Although this isn't likely to be the final version of the data model, it's probably complete enough for a start. Using your ERD tool, you could go ahead and generate the database-object-creation scripts and then upload and run them through APEX SQL Workshop's SQL Scripts interface. However, because your data model is so small, in the next chapter you will use the Object Browser tool to create the objects from scratch.

Summary

Identifying the problems your APEX application is supposed to solve is only half the battle. Good database design—and designing specifically with APEX in mind—is the key to creating a successful APEX application. Taking the time to make sure you have a solid foundation means you can take full advantage of everything APEX gives you so that there is less work to do later.

■ ■ ■

SQL Workshop

Now that you have a graphical representation of what your underlying tables should look like, in the form of an entity-relationship diagram (ERD), it's time to dig in and start creating the objects. As mentioned before, you could use your ERD tool to generate the scripts, but to get used to using the SQL Workshop, here you'll create these objects from scratch.

■ **Note**   For this and many of the following chapters, you need to download the code that accompanies the book. If you haven't already done so, download the code `.zip` file from this book's home page at `www.apress.com`. Then unzip it to a directory from which you can retrieve the files easily.

Creating Objects with the Object Browser

SQL Workshop's Object Browser is somewhat misnamed, because it not only allows you to view database objects, but also lets you create and edit them. For now, you'll skip the USERS table; you will come back to it later in the book. Right now, you'll focus on the TICKETS and TICKET_DETAILS tables. From this point forward, you'll follow step-by-step instructions interspersed with figures and discussions about what you're trying to achieve and why you're doing it the way you are. Let's get started:

1. Log in to your APEX workspace. You're presented with the workspace's Home page, which, unless you've been doing other work in this workspace, probably looks a little sparse.

2. Using the tabbed navigation bar across the top of the Home page, pull down the **SQL Workshop** submenu by clicking the arrow on the right side of the tab (see Figure 4-1).
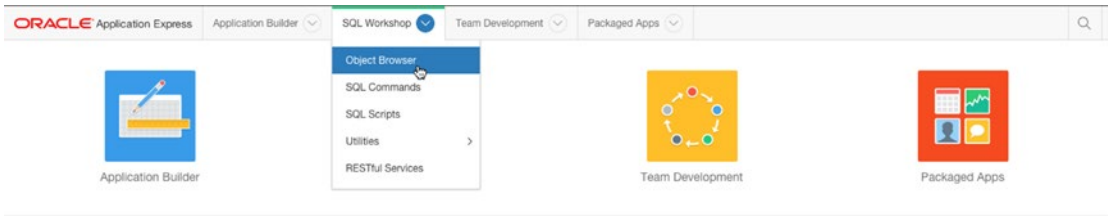
Figure 4-1. *Navigate to the Object Browser*

3. Click the **Object Browser** option.

4. In the Object Browser, click the **"+"** icon (which stands for Create) button in the upper-right corner and select **Table** from the drop-down menu.

The Create Table Wizard opens. The first screen (Figure 4-2) allows you to name the table and enter the details for each of the table's columns. Using the two arrows in the Move column, you can move the columns into whatever order you like. This affects the order in which they're defined and stored in the table. If you run out of empty rows in which to enter columns, you can click the Add Column button to add a new, empty column-definition row to the form.
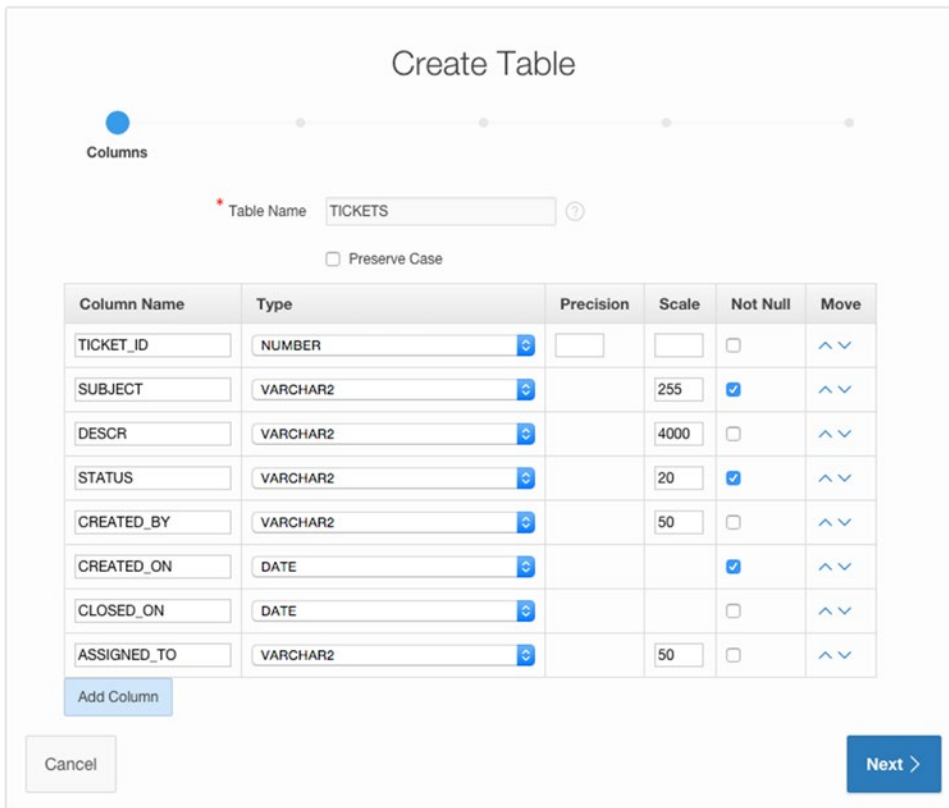
Figure 4-2. *Defining the table and its columns*

5. Enter the details for the TICKETS table as indicated in the ERD from the end of Chapter 3 and in Figure 4-2. Make sure you include the appropriate checks in the Not Null column of the form. Then click **Next**.

The next step in the wizard (Figure 4-3) lets you choose how you would like the primary key to be populated and which column to use as the primary key. The four options for primary key are fairly self-explanatory, but the two in the middle are probably the most common. You're starting from scratch and therefore don't have any existing sequences defined in your database. By selecting "Populate from a new sequence," you tell APEX to create a sequence for you and to create a database trigger on the table that will populate the selected primary-key column with the next value from the sequence, unless the field already has a value. You're given the chance to name the sequence in this step as well. In this instance, you'll use the default name given.

Figure 4-3. *Defining the table's primary key*

6. Select the **Populated from a new sequence** radio button. After the screen changes, select **TICKET_ID (NUMBER)** for the **Primary Key**. Leave the **Sequence Name** set to its default and click **Next**.

7. You're not going to create any foreign keys in this table just yet, so leave the defaults and click **Next**.

The Constraints screen in Figure 4-4 allows you to add either Unique or Check constraints to the table definition. You add a constraint by defining the constraint in the Add Constraints region and clicking the Add button to add it to the list. Below the Add Constraints region are two Help regions. Clicking the arrow to the left of the region title expands the help and shows the columns you defined in the table and examples of how to code various check constraints.
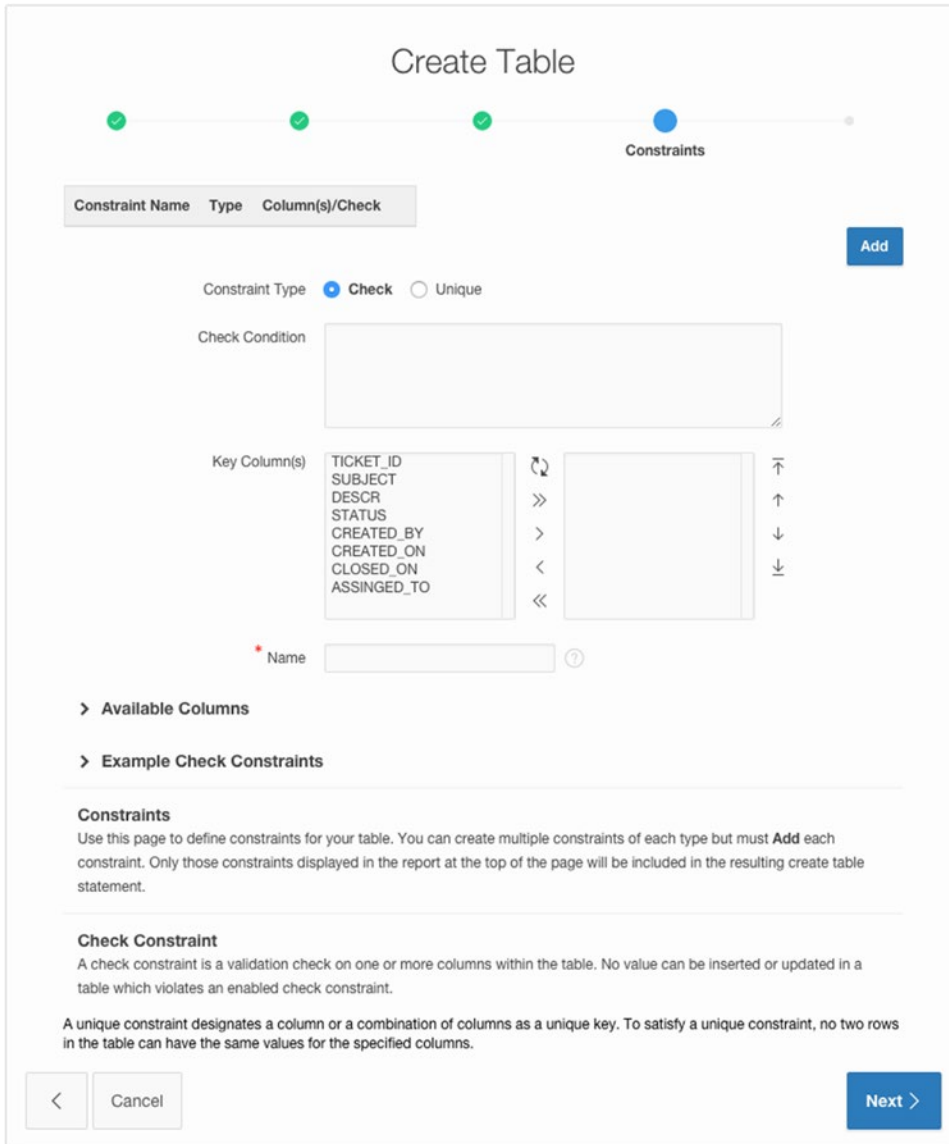

Figure 4-4. *The constraints definition step*

When you click the Add button, the definition of the constraint is added to the list of constraints at the top of the page. You can define as many constraints on a given table as is necessary. Once you're done, simply continue with the wizard.

8. You're not going to create any Unique or Check constraints here, so stick with the defaults and click **Next**.

The final step of the Create Table Wizard gives you the chance to confirm your request and, if desired, review the code that will be executed. If you need to make changes to the table definition, you can use the buttons at the bottom of the region to navigate back through the wizard steps. To view the code, click the arrow to the left of the **SQL** label to expand the region, as shown in Figure 4-5.
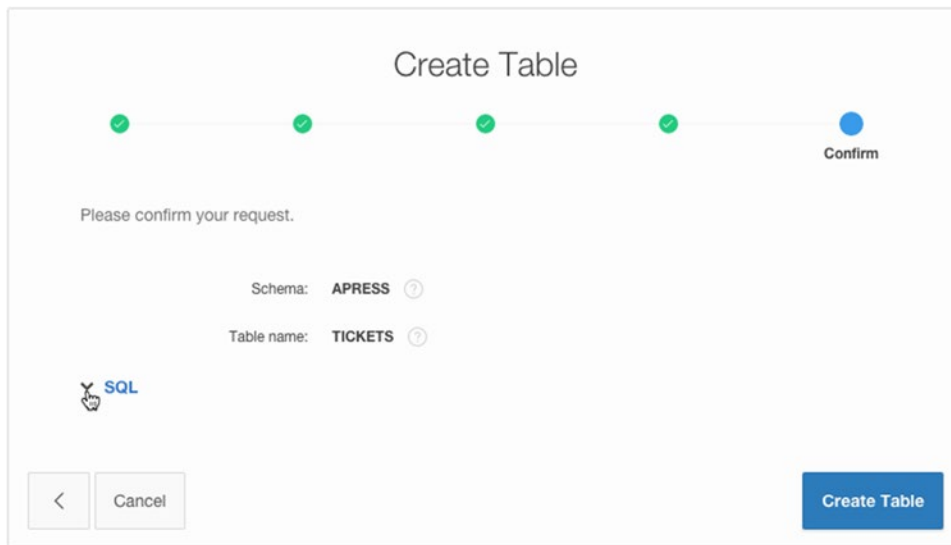
Figure 4-5. *Review the Create Table Wizard's SQL*

9. Review the text in the SQL region presented by the Create Table Wizard. Click **Create Table** to complete the wizard.

Once you've successfully completed the wizard, you're taken back to the Object Browser, and the definition of the TICKETS table is displayed. Take a moment to examine the definition of the table. You should see all the columns that you defined listed. If you click the Constraints tab at the top of the definition region, you will see a number of different constraints, including the primary-key constraint on TICKET_ID.

In the upper-left corner of the Object Browser is a select list that defines the object type being browsed. Use this select list to choose Sequences. You see that APEX created a sequence called TICKETS_SEQ that will be used to fill the TICKET_ID.

Once again, use the Object Type select list and choose Triggers. You will see a trigger named BI_TICKETS (BI stands for "before insert"). Selecting the BI_TICKETS trigger on the left-hand side and then clicking the Code tab above the trigger details will show the code for the trigger that is using the TICKETS_SEQ sequence to fill the TICKET_ID if it is null. You should see code similar to the following:

```
create or replace trigger "BI_TICKETS"
  before insert on "TICKETS"
  for each row
begin
  if :NEW."TICKET_ID" is null then
    select "TICKETS_SEQ".nextval into :NEW."TICKET_ID" from sys.dual;
  end if;
end;
```

Now that you have the TICKETS table defined, let's go back and create the TICKET_DETAILS table. This time, you'll create a foreign key to the TICKETS table, as a CASCADE DELETE. This means that if you delete a ticket, the ticket details will automatically be deleted as well.

10. Start the Create Table Wizard using the **Create** (+) button.

11. Enter the table name and column definitions based on the ERD and Figure 4-6, and click **Next**. Again, make sure you check the appropriate Not Null checkboxes.
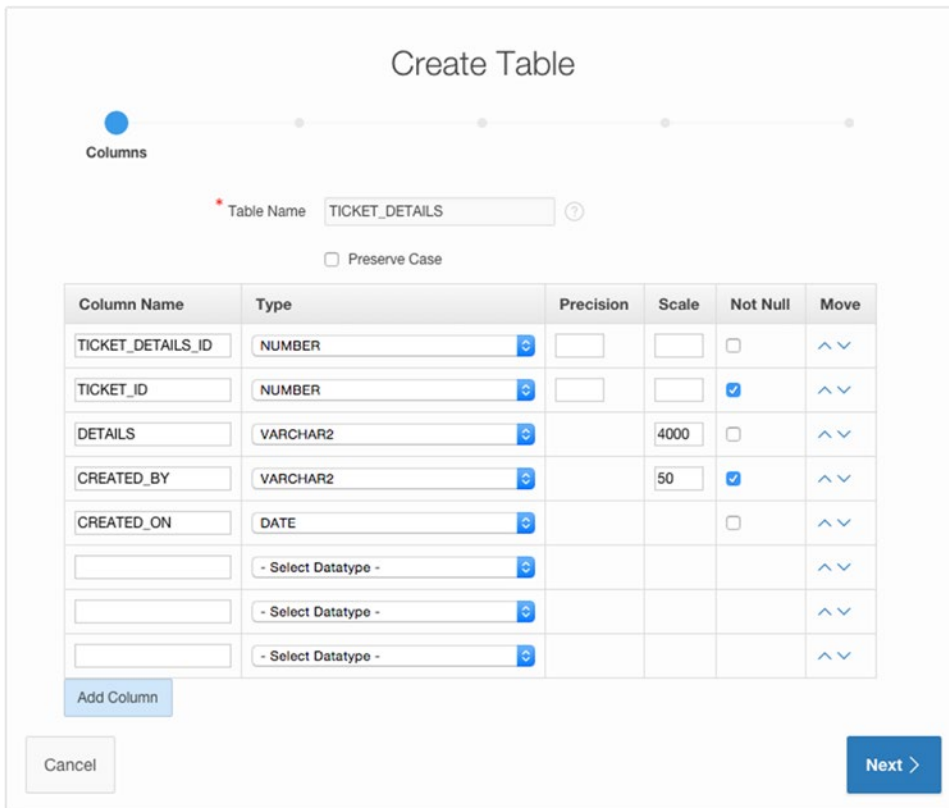

Figure 4-6. *Defining the TICKET_DETAILS table*

The next set of steps is purposely a bit more vague than the previous ones. You should be used to using the Create Table Wizard by now, but if you need a refresher, just look at the previous steps.

12.   Choose **Populate from a new sequence** for the primary key, select **TICKET_DETAILS_ID(NUMBER)** as the **Primary Key** column, and click **Next**.

13.   Add a foreign key between the TICKET_ID in the TICKET_DETAILS table and the TICKET_ID in the TICKETS table. Make sure the Delete action is set to **Cascade Delete**. Your screen should look similar to that in Figure 4-7. Additionally, make sure you tab out of the References Table field in order to cause APEX to display the shuttle control that allows you to choose the referenced columns.

Figure 4-7. *Defining a cascade-delete foreign key for TICKET_ID*

14.   Click the **Add** button to add the new foreign-key constraint.

15.   Click **Next** (see Figure 4-8).

Figure 4-8. *Foreign key as defined in the table wizard*

16. No constraints are required for this table. Click **Next**.

17. Review the SQL and click **Create Table** to complete the wizard.

Loading Data with the Data Workshop Utility

Now that you have your two base tables defined, you can begin working to migrate the old data into your shiny new data structure. You can use SQL Workshop's Data Workshop utility to load and unload data from an Oracle schema in a number of ways, as shown in Figure 4-9. The Data Load option allows you to choose Text Data, XML Data, and Spreadsheet Data.
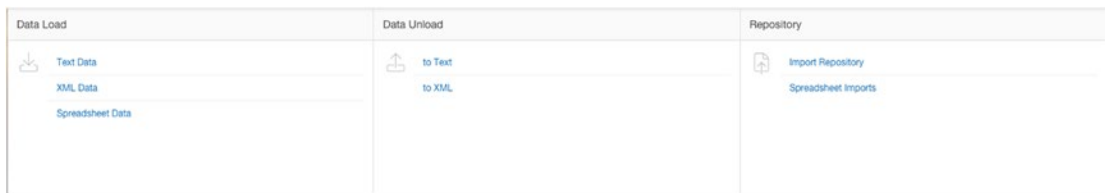
Figure 4-9. *Data Load and Unload methods provided by the Data Workshop utility*

Although three separate options are presented, the Text Data and Spreadsheet Data options actually use the same Data Load Wizard. There is little or no discernible difference in the actions of the wizard regardless of which option you select.

The third option (XML Data) allows you to load data that has been exported in Oracle's proprietary XML Data Transport format. The format looks like this:

```
<ROWSET>
<ROW>
  <USER_ID>2</USER_ID>
  <USER_NAME>DOUG</USER_NAME>
  <PASSWORD>A69856770A9AB9CBB0479573FCB3E2A5</PASSWORD>
 </ROW>
 <ROW>
  <USER_ID>3</USER_ID>
  <USER_NAME>DAVID</USER_NAME>
  <PASSWORD>E2E89134B8AC6E1FFC14139A6FB2C10B</PASSWORD>
 </ROW>
</ROWSET>
```

In your imaginary company, the help-desk technicians have been using Microsoft Excel to track tickets, so you're going to load the data using the Spreadsheet Data option. A quick glance at the spreadsheet your technicians use shows you that they have two separate sheets in the Excel workbook: TICKETS and TICKET_DETAILS.

Knowing that you're using preexisting tables that already have primary and foreign keys in place, you need to be careful about how you load the data. TICKET_DETAILS depend on TICKETS for their parentage, so you need to load the TICKETS data first. Your spreadsheet should look like that in Figure 4-10.

	A	B	C	D	E	F	G	H
1	ticket_id	subject	descr	status	assigned_to	created_on	closed_on	created_by
2	1	Cannot log into E-Mail	User called and cannot log into his MS Outlook e-mail Account.	OPEN	SCOTT	1-Jan-07	3-Jan-07	PAUL
3	2	PC will not turn on	The user's PC will not turn on when the power button is pressed.	CLOSED	DAN	1-Jan-07		RINGO
4	3	Need more memory	User needs more memory installed	OPEN	DOUG	1-Jan-07		GEORGE
5	4	MSIE Crashed 4 times	MSIE keeps on crashing for any site	CLOSED	SCOTT	1-Jan-07		JOHN
6	5	Need to install SP2	SP2 Upgrade needed in order to be compliant	OPEN	TIM	1-Jan-07		ALEX
7	6	Network drive not being mapped	X: drive not being mapped to \corp\share	OPEN	TIM	1-Jan-07		GEDDY
8	7	BSOD after rebooting	Blue Screen of Death every time system is rebooted	OPEN	DOUG	2-Jan-07		NEAL
9	8	Wireless signal not strong enough	Wi-Fi signal not as strong as it was last week	CLOSED	SCOTT	2-Jan-07	3-Jan-07	JOHN
10	9	I think I have a virus	Something is not right - PC is slow	OPEN	DAN	2-Jan-07		ROBERT
11	10	Virus Definitions Dates	Message stating that virus updates are needed keeps appearing	CLOSED	SCOTT	2-Jan-07		JOHN
12	11	Funny smell coming from PC	There is an odd odor emanating from my PC...	OPEN	TIM	3-Jan-07		JIMMY
13	12	Accidentally deleted Q2.ppt	File Q2.ppt placed in Recycle Bin; bin emptied	OPEN	DAN	3-Jan-07		EDDIE
14	13	Several dead pixels on screen	There are at least 4 dead pixels on the display	PENDING	DOUG	3-Jan-07		ALEX
15	14	Smartphone will not sync with Outlook	Motorola Q does not sync with Outlook contacts and calendar events	OPEN	SCOTT	3-Jan-07		MICHAEL
16	15	Getting Out of Memory errors	Same Out of Memory error occurs when Office starts	PENDING	DAN	3-Jan-07		DAVID
17	16	VPN Client Install Issues	Cannot install VPN client - installer errors out each time	OPEN	DOUG	4-Jan-07		JACKIE
18	17	Mouse is not working	Mouse does not move the pointer anymore	OPEN	TIM	4-Jan-07		TITO
19	18	Speakers are too soft	Cannot get good quality of sound from built-in speakers	OPEN	SCOTT	4-Jan-07		JERMAINE
20	19	Keyboard busted	None of the keys work (I had to use someone elses PC to enter this)	PENDING	DAN	5-Jan-07		MICHAEL
21	20	Disk is Full	No more space error keep coming up	OPEN	DOUG	5-Jan-07		MARLON

Figure 4-10. Spreadsheet data from the TICKETS tab of your Excel workbook

Once you have the TICKETS data in the clipboard, you can switch back to APEX and use the Data Load Wizard to insert this data into your TICKETS table. Here are the steps to follow to load data from the spreadsheet into the database:

1. Locate the helpdesk_spreadsheet.xls file where you downloaded the supporting files for this book, and open it with Microsoft Excel. Navigate to the TICKETS tab. Notice that you have a row for each ticket and a header row that contains the column headings for each of the columns.

2. Select all the data, including the column headings, and copy it to the clipboard. Be cautious not to accidentally select any rows that don't have data in them, because that may cause phantom rows or errors in the Data Load Wizard.

3. Switch back to your web browser, and, using the pull-down menu on the SQL Workshop tab, select **Data Workshop** under the **Utilities** section.

4. In the **Data Load** region, click **Spreadsheet Data**. You should see the Load Data dialog shown in Figure 4-11.

Figure 4-11. Preparing to copy and paste the spreadsheet data and load it into the existing TICKETS table

5. In the wizard, select **Existing table** for **Load To** and **Copy and paste** for **Load From**, and click **Next**.

6. Select your "parse as" schema from the **Table Owner** select list. This is the same schema in which you created your tables in the Object Browser.

7. Select **TICKETS** for the **Table Name**, as shown in Figure 4-12, and click **Next**. This is the table into which you'll load the TICKETS data.

Load Data - Table Owner and Name ×

Table Owner and Name

Select the database schema and name of the table you would like load data into.

Table Owner   APRESS

Table Name   TICKETS

< Cancel   Next >

Figure 4-12. Enter the name of the table into which you're going to load the data

8. Paste the data that you copied to the clipboard in step 2 into the **Data** text area. Change the **Separator** from a comma to **\t**, which stands for Tab Delimited. Now ensure that the **First row contains column names** box is checked, as shown in Figure 4-13. Click **Next**. (You may have to scroll within the dialog to see all the options.)
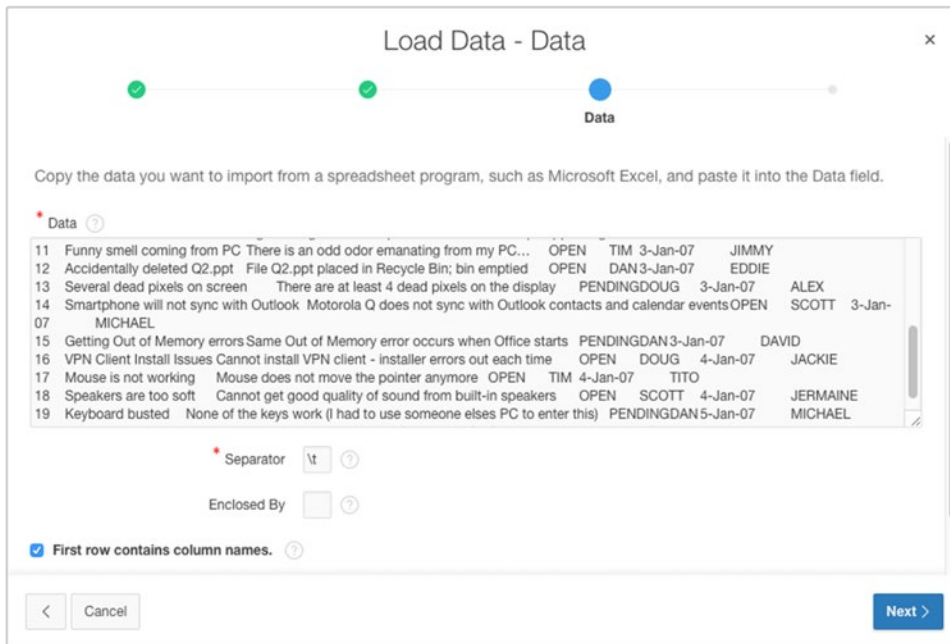
Figure 4-13. *Pasting the spreadsheet data into the Data text box*

When you click Next, APEX parses the data you've pasted in and does its best to match the column names in the first row of the spreadsheet data to the column names of the table into which you're loading the data. On the next screen, you're presented with column mapping so you can check its accuracy and, if necessary, make alterations and corrections.

APEX is very good about matching column names as defined in the spreadsheet with those that have the same name in the table. However, if the names differ, it doesn't try to guess but instead leaves the mapping to you.

If you scroll to the right, you should see that APEX has matched all the column names from the spreadsheet correctly to the table columns. If, for some reason, the mappings aren't right, you can adjust them using the drop-downs shown in Figure 4-14.
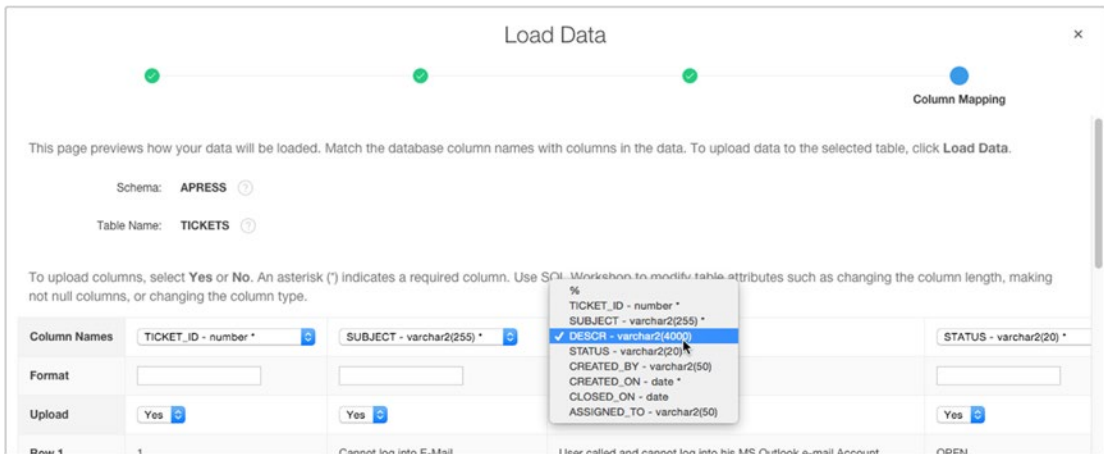
Figure 4-14. *Manually mapping the data columns to the table*

9. When you're sure all the mappings are correct, click the **Load Data** button to load the data into the TICKETS table.

After the data is loaded, you're presented the Spreadsheet Repository screen shown in Figure 4-15. That screen shows that twenty rows were loaded into the database and zero errors occurred during loading.
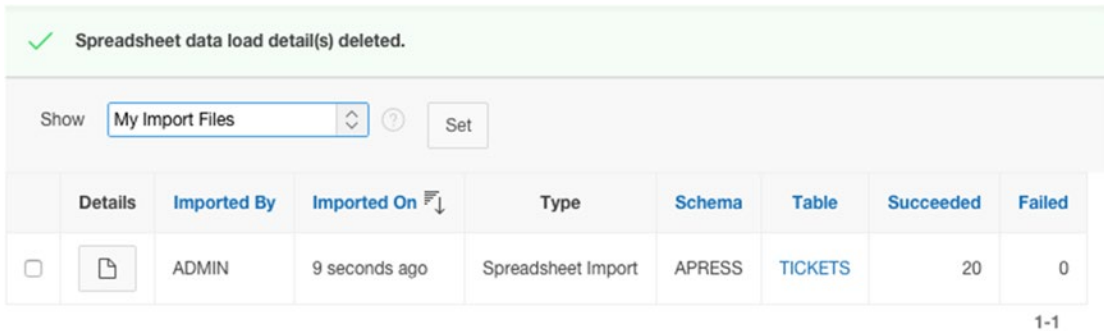
Figure 4-15. *Data has been loaded into the TICKETS table*

If you navigate to the Object Browser, select the TICKETS table, and look at the data in that table, you can see that the records that were in your spreadsheet have been loaded into the database. To finish the job, you need to load the data for TICKET_DETAILS. Here's what to do:

10. Navigate to the Data Workshop, click the **Spreadsheet Data** link in the Data Load region, and click **Next**.

11. In the wizard, select **Existing Table** for <u>Load To</u> and **Copy and paste** for <u>Load From</u>, and click **Next**.

12. Select your "parse as" schema from the <u>Table Owner</u> select list. This is the same schema in which you created your tables in the Object Br**owse**r.

13. Select **TICKET_DETAILS** for the <u>**Table Name**</u>, and click **Next**.

14. In Microsoft Excel, navigate to the TICKET_DETAILS tab and copy all the data, including the column headings, in that spreadsheet to the clipboard.

15. In your browser, paste the data you copied to the clipboard into the <u>**Data**</u> text area, change the <u>**Separator**</u> to \t, and ensure that <u>**First row contains column names**</u> is checked, and click **Next**.

16. Review the mappings made by APEX in the <u>**Define Column Mapping**</u> region. It should have mapped everything correctly. Click **Load Data** to complete the data load. The summary should say that twenty-two records were loaded into the TICKET_DETAILS table with zero errors.

You now have both of the main tables created and loaded with the legacy data. This alone is enough to start developing an application, but you're not quite ready to begin yet.

Creating a Lookup Table

Have a look at the definitions and data of the tables you just created. They're basically mirror images of the spreadsheet tabs the technicians were using before. If you examine the data closely, you will notice that there are still some areas where the data isn't quite normalized as well as it could be.

For instance, in the TICKETS table, the STATUS column has only three values—OPEN, CLOSED, and PENDING—which repeat over and over. The data values in this column indicate that it's a perfect candidate for creating a lookup table. Although it's tempting to create the table manually with the Create Table Wizard and then manually migrate the data, APEX can create a lookup table—complete with its own sequence, trigger, and foreign key—and modify the original table so it points to the new lookup table, all without you writing a line of code. Here's how:

1. Navigate to the Object Browser and select the **TICKETS** table in the <u>**Object List**</u> on the left side of the screen. You should see results similar to those shown in Figure 4-16.
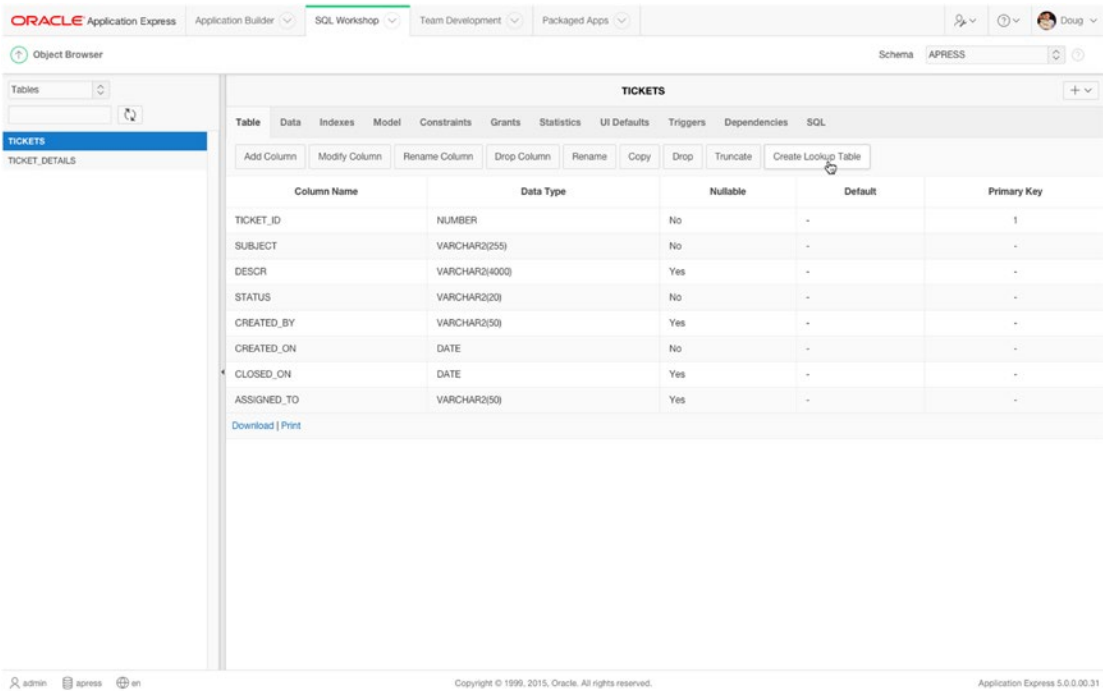
Figure 4-16. *Clicking the Create Lookup Table button starts the Create Lookup Table Wizard*

2. Make sure the **Table** tab is selected.

3. Below the tab bar is a set of button-like links. Click the **Create Lookup Table** button, as shown by the mouse arrow in Figure 4-16; it starts the Create Lookup Table Wizard.

The first step of the Create Lookup Table Wizard (Figure 4-17) gives you the option to show either only VARCHAR column types or all column types. It defaults to VARCHAR because that's most likely to be the candidate for lookup tables. Looking at the columns presented in the wizard, you will see that one of the VARCHAR columns is your STATUS column.

Figure 4-17. *Selecting the STATUS column as the source of your lookup table*

4. Select STATUS as the column from which you want to create the lookup table, and click **Next**.

5. The next step allows you to name your lookup table and the sequence that is related to it. APEX has chosen a reasonable name for the new table and sequence, so take the defaults and click **Next**.

6. The final screen of the wizard (Figure 4-18) provides you with information about the choices made and the action that is about to be performed. It's easy to miss the SQL syntax link just below the wizard region. Click the **SQL** link to show the SQL.

Figure 4-18. *Clicking the SQL syntax link shows the SQL about to be executed*

Examining the SQL shows the steps that will be taken to create the new lookup table, associated sequence, and trigger; insert the data into the table; and update the data in the originating table so that it references your new lookup table. That's quite a lot of work saved on your part.

7. Click **Create Lookup Table** to complete the wizard. You're taken back to the Object Browser. The STATUS_LOOKUP table is highlighted and its details are shown.

Use the Object Browser to examine the objects that the wizard created.

Loading and Running SQL Scripts

The SQL Scripts tool of SQL Workshop allows you to create, upload, manage, and run SQL scripts. These scripts are similar to SQL*PLUS scripts in many ways. However, if you use scripts written for SQL*PLUS, APEX ignores any SQL*PLUS-specific syntax.

Once a script is created or loaded, it's moved into the script repository, where it remains until you decide to remove it. From the script repository, you can decide to edit or run the script. When you run a script, APEX stores the results for you to view later. For example, you can come back to review the results for possible error messages.

You're now going to load and run a script that will modify the underlying data just a bit. Here's why: In the real world, the spreadsheet you received from the help-desk team would have current dates and data in it; however, the ticket dates in the spreadsheet that is downloaded with the .zip file accompanying this book very likely aren't current. This would cause you to have to search back in history for the tickets if you were searching by date. This script will update these dates so they're recent.

Another thing you need to take into consideration is that you loaded a bunch of data into your tables that already had IDs assigned to them. Because the IDs were loaded with the data, you didn't use your database sequences. Therefore, your sequences are out of synch with the data. You need to drop and re-create your sequences so the next sequence number is greater than the largest ID used in the associated table.

You're also going to alter the Before Insert trigger that was automatically created on the TICKETS table so that it automatically fills in the CREATED_ON column. You'll also create a couple of database views that will be used later to retrieve data formatted for some of the specific charts and calendars you're going to create.

Finally, you'll create a function that, when passed a status name such as OPEN, passes back the ID for that status. This function is used in a number of places, because you can't guarantee you know the ID value of a given status. Therefore, this function is the only safe way to get the associated ID for a given status.

When you're in any of the SQL Workshop tools, you can use the pull-down menu of the SQL Workshop tab as a quick way to navigate to each of the other tools. Figure 4-19 shows this menu and highlights the SQL Scripts option.
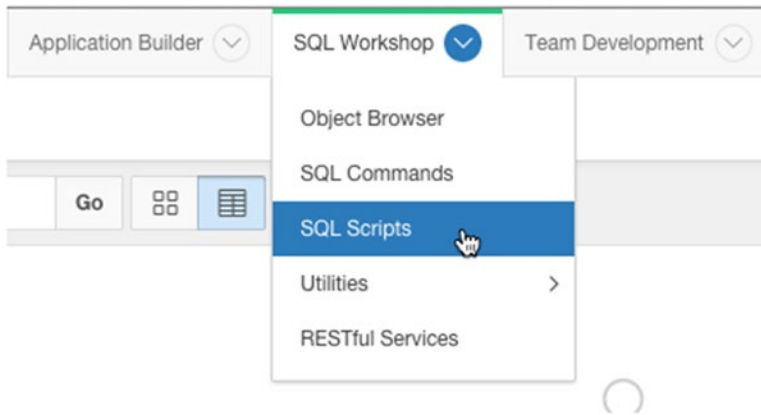
Figure 4-19. *Using the SQL Workshop menu to navigate to the underlying tools*

Here's what to do to run the script that will update your schema objects appropriately:

1. Navigate to the **SQL Scripts** tool using SQL Workshop menu.

2. Click the **Upload** button in the upper-right section of the screen.

3. Click **Browse** or **Choose File** buttons to search for the SQL file to upload.

4. In the pop-up file-finder window, locate and select the ch4_schema_changes.sql file and click **Upload**. You don't need to give the script a name; it defaults to the name of the script as it appears at the OS level.

Once the file has been uploaded, you're presented with a SQL Scripts report showing the script that you just uploaded. From this point, you can either edit or run the script. If you want to see what the script contains, feel free to edit it. You can run the script from the edit screen as well.

5. Run the script by clicking either the **Run** button (if you're editing the script) or the **Run** icon (if you're still viewing the SQL Scripts report).

6. As shown in Figure 4-20, you're asked to make a selection between Run in Background and Run Now. Select **Run Now**.
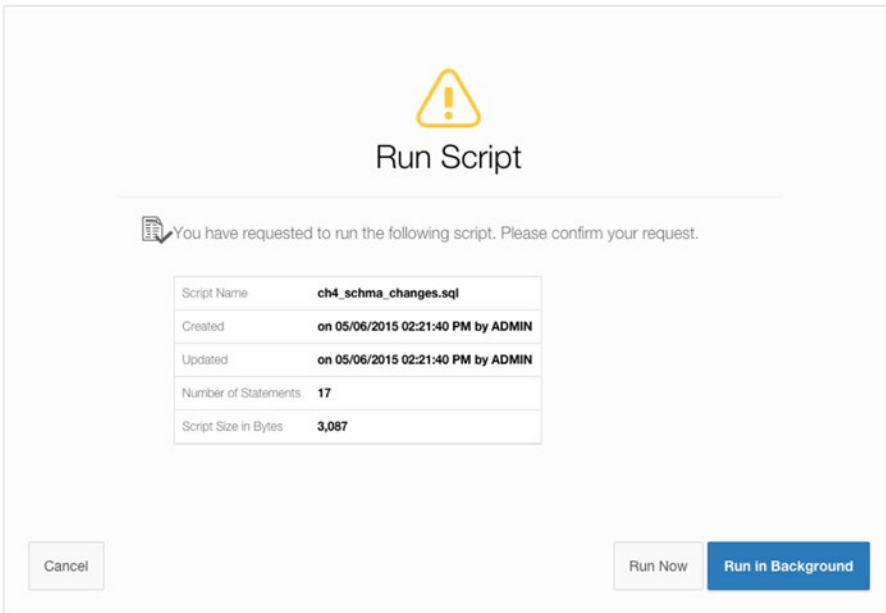
Figure 4-20. *Choose whether to Run in Background or Run Now*

The script is run, and you're immediately taken to the Manage Script Results page. You'll most likely see that your script status is COMPLETED.

7. Click the **View Results** icon at the far-right end of the report row to see the results of the script. Figure 4-21 shows where to click.

Figure 4-21. *Click the View Results icon to view the results of running the script*

The View Results page allows you to see what happened when the script was run. The default view shows an overview by displaying the first 50 or so characters of each statement along with some brief feedback and the number of rows affected by the statement. Figure 4-22 shows the results from a run of the script.

Figure 4-22. *The summary view of the script results*

You can, however, get more detailed feedback by changing the report view to Detail. Doing so gives you far more insight, especially if you have a script that had errors during execution. Figure 4-23 shows a detailed view.

Figure 4-23. *The detailed view of the script results*

In either view, you can quickly see whether the script encountered any errors by scrolling to the bottom of the page and looking at the report footer, which is where the report displays the total number of statements processed, the number of those that were successful, and the number that generated errors. Figure 4-24 shows the number of statements processed from a run of the script.

Run By	ADMIN
Parsing Schema	APRESS
Script Started	Wednesday, May 6, 2015
	3 minutes ago
Elapsed time	0.11 seconds
Statements Processed	17
Successful	17
With Errors	0

Figure 4-24. *In the footer of either report is the success summary for the script*

User Interface Defaults

Before you start to write your application, one last thing you can do that will make your life easier along the way is to create some User Interface (UI) Defaults. This, in my opinion, is one of the most underutilized features of APEX.

Understanding User Interface Defaults

UI Defaults allow you to customize the default display attributes for tables, views, and their columns. They can be used to control many properties, including alignment, searchability, display sequence, what type of item is created for a column, default values, and many more.

For instance, when you're creating a new form or report via a wizard (which is most of the time), APEX asks if you wish to use UI Defaults. If you select "Yes" and defaults are available, APEX applies them to the appropriate regions or items based on the tables or columns for which the attributes are defined. UI Defaults are divided into two categories: Attribute Dictionary and Table Dictionary.

The Attribute Dictionary allows you to create more-generic UI Defaults based on attribute names. Consider this a more macro-level definition.

Let's say you create an attribute-level default for any attribute named PHONE_NUMBER. If a column named PHONE_NUMBER appeared in a table and didn't have a Table Dictionary default assigned, the Attribute Dictionary default would take effect.

Attribute Dictionary definitions can also be assigned synonyms, allowing more than one attribute name to share the same actual definition. So, for instance, you could create the synonyms PHONE, TELEPHONE, PHONENUMBER, and so on for the original PHONE_NUMBER definition. If the wizard ran into a column with any of those names, it would apply the PHONE_NUMBER defaults to the APEX item that is created.

The Table Dictionary allows you to define defaults for a specific table or column, and those defaults are only applied to APEX regions or items created for those specific items.

Here are some things to note about UI Defaults:

- Table Dictionary defaults always override Attribute Dictionary defaults.

- When an item is created using UI Defaults, no relationship is established with the UI Default. Therefore, if you later change the definition of the UI Default, the changes aren't propagated to previously created items.

- Items created before UI Defaults have been established don't inherit properties of the UI Default.

- Developers can choose not to use UI Defaults, and even if they're used, can override them after the component is created.

Having said that, UI Defaults do help ensure consistency across your application and make your job much easier as a developer.

Defining UI Defaults for Tables

UI Defaults can be managed either from SQL Workshop's Object Browser or from SQL Workshop's Utilities page. Here's what to do:

1. Navigate to SQL Workshop's UI Defaults page via the drop-down menu on the SQL Workshop tab and select **Utilities**; then, choose **User Interface Defaults** from the drop-down menu.
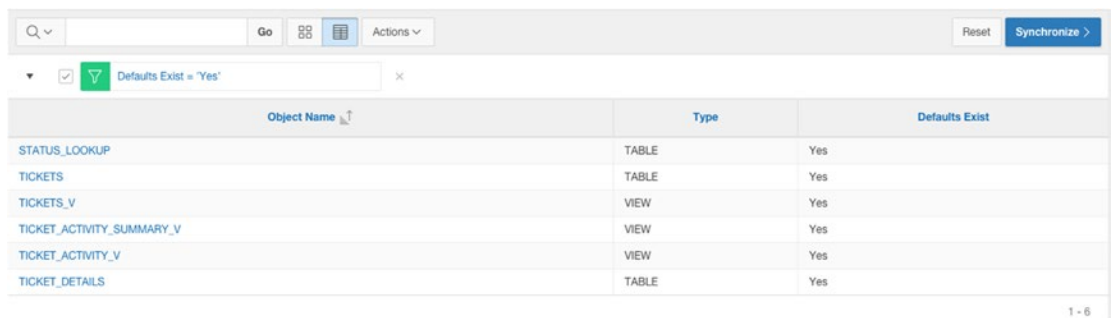
You're taken to the UI Defaults dashboard, where things likely look pretty sparse. This is because you haven't actually created any UI Defaults yet. The first step in creating UI Defaults is to synchronize the Table Dictionary with the database so it knows what tables are in your schema.

2. Click the **Table Dictionary** tab along the top of the page, and then click the **Synchronize** button on the screen that appears.

This initiates the Synchronization Wizard. This wizard shows you the number of tables with defaults defined and the number without. In this case, you should have zero objects with defaults and six objects without.

3. Click the **Synchronize Defaults** button to begin the synchronization with the database. This may take a little time.

Once the Table Dictionary is synchronized with the definitions in the database, you're presented with the report seen in Figure 4-25, which shows each table that now has base UI Defaults. If you have other tables in your schema, they also appear in this report.
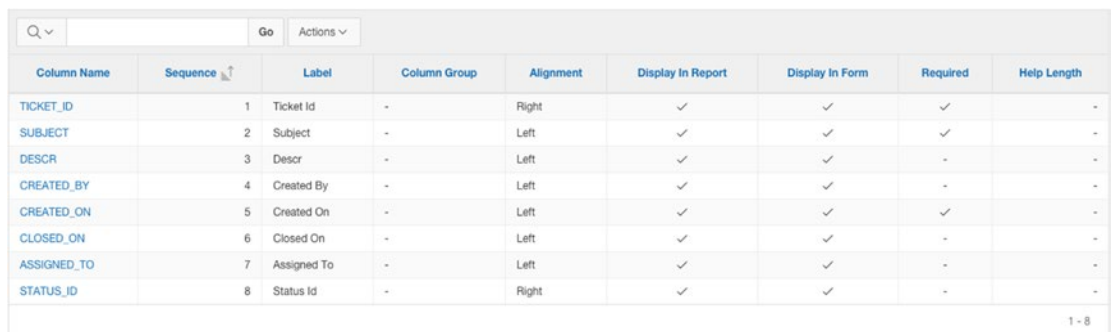
Object Name	Type	Defaults Exist
STATUS_LOOKUP	TABLE	Yes
TICKETS	TABLE	Yes
TICKETS_V	VIEW	Yes
TICKET_ACTIVITY_SUMMARY_V	VIEW	Yes
TICKET_ACTIVITY_V	VIEW	Yes
TICKET_DETAILS	TABLE	Yes

Figure 4-25. *List of objects with UI Defaults defined*

You can now view or edit the UI Defaults for each of these tables. Start by viewing the UI Defaults for the TICKETS table:

4. Click the **TICKETS** link in the report. You should see the results shown in Figure 4-26.

Column Name	Sequence	Label	Column Group	Alignment	Display In Report	Display In Form	Required	Help Length
TICKET_ID	1	Ticket Id	-	Right	✓	✓	✓	-
SUBJECT	2	Subject	-	Left	✓	✓	✓	-
DESCR	3	Descr	-	Left	✓	✓	-	-
CREATED_BY	4	Created By	-	Left	✓	✓	-	-
CREATED_ON	5	Created On	-	Left	✓	✓	✓	-
CLOSED_ON	6	Closed On	-	Left	✓	✓	-	-
ASSIGNED_TO	7	Assigned To	-	Left	✓	✓	-	-
STATUS_ID	8	Status Id	-	Right	✓	✓	-	-

Figure 4-26. *The table and column UI Defaults overview*

On the page in Figure 4-26 you can see an overview of the UI Defaults for the TICKETS table. In the upper portion of the report are the table-level definitions, including what the Form and Report regions based on this table will be called. In the lower portion is a list of the table's columns, the labels that will be used, how they will be aligned when used in a report, whether they will be displayed in a report or a form, whether their REQUIRED attribute will be set in a form, and whether they have any help text.

Next, edit both the table-level and column-level attributes:

5.   Click the **Edit Table Defaults** button in the upper-right portion of the report. This allows you to edit how Form and Report regions based on this table are named.

6.   Enter Manage Tickets for the **Form Region Title**, leave the **Report Region Title** as it is, and click **Apply Changes**.

Clicking any of the column names takes you to a page that allows you to set UI Defaults for that specific column. As you peruse the column UI Defaults, notice that several things have been set for you, including the REQUIRED attribute. When APEX synchronized with the database, it saw that certain fields were marked as NOT NULL at the database level and translated those constraints into UI Defaults for you.

APEX also makes some decisions based on the column's data type, such as how to align the column when it's displayed in a report. Use the following information to alter the UI Defaults for the indicated columns by clicking the link in the column name:

Column: SUBJECT

Label: Subject

Help Text: A brief title for the issue.

Column: DESCR

Label: Description

Help Text: Describes the ticket in detail. Please be as complete as you can.

Resizable: YES

Width: 50

Height: 5

Column: STATUS_ID

Label: Status

If you wish, you can go ahead and set the UI Defaults for any of the other columns and/or tables. Just remember, what you do now will affect what the wizards create for you later, so if something doesn't look exactly like what is shown in this book, check what you set for UI Defaults.

Summary

SQL Workshop may not measure up to some of the more popular GUI tools, but it certainly has the power to do most things you need to do relating to the creation and management of tables and data. You've also seen that SQL Workshop has a few built-in but hidden gems like the Create Lookup Table Wizard. Finally, among the many useful utilities is the UI Defaults manager, making your job as a developer just a bit easier.

Sure, this chapter hasn't covered SQL Workshop in its entirety, but you've definitely gained a fair amount of insight as to what it's capable of. You will use SQL Workshop for a number of other things throughout this book, but don't wait. Go poke around in some of the dark nooks and crannies and see what you find!

CHAPTER 5

■ ■ ■

Applications and Navigation

With some basic data created, you can now create the shell for your application. APEX provides a wizard for creating applications. Several options are available within the wizard to assist with generating a starting application. Based on how much prior planning has been done, the result of running the initial application wizard may vary. You will start this chapter by walking through the steps of the wizard, as I highlight the most common features.

For the example application, you will create the most basic shell of the application with only one page. In other scenarios, you could create an initial draft of all your pages. To illustrate the individual wizards for creating pages, they will be explored in more detail in later chapters.

After the example application has been created, you'll add shared components to it. Shared components are items and structures that are common across all the pages in the application. You will prepare breadcrumbs, lists, and lists of values (LOVs) for use; you will also learn how the Global Page concept works. By the end of the chapter, you'll have some basic components for the application and a starting outline for the remaining pages.

The Create Application Wizard

Applications in APEX are created through application imports, by copying an existing application, or by running the Create Application wizard. The Create Application wizard is the first step in creating an application from scratch. This chapter will walk you through the process of creating the Help Desk application using the Create Application wizard.

To begin, navigate to the Application Builder in APEX. You can do this from the APEX Home page by clicking either the Application Builder menu item or the Application Builder icon shown in Figure 5-1. The Application Builder shows a list of the current applications. At the top of the list is a highlighted Create button, shown in Figure 5-2. Click the button, and the wizard starts.
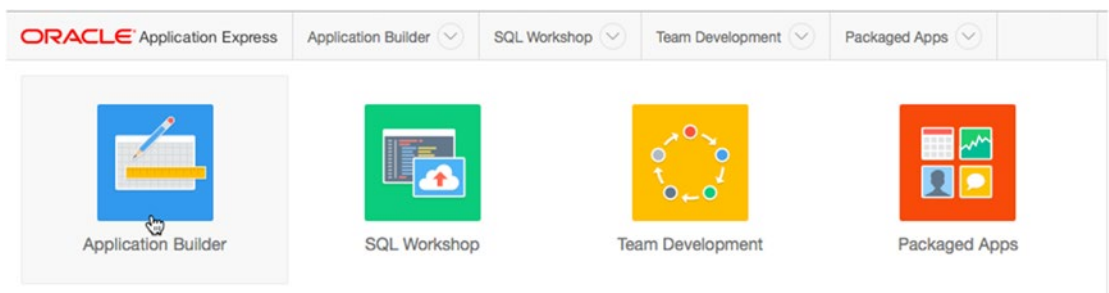
Figure 5-1. *The Application Builder icon on the APEX Home page*
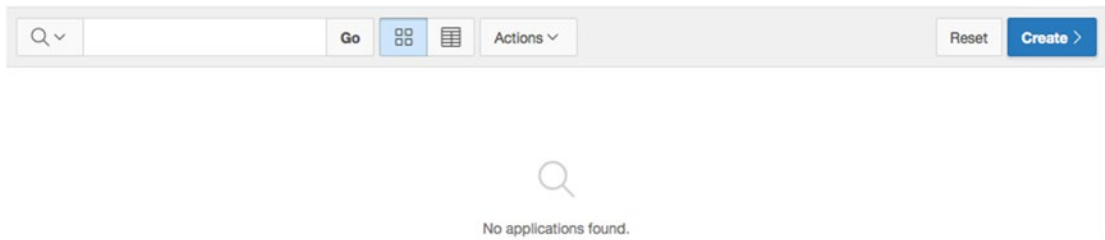
Figure 5-2. *The Create button*

You're presented with four choices for application type: Desktop, Mobile, Websheet, and Packaged Application. The Application Builder will quickly become very familiar to you when you're working with APEX. Because of this, the shortcut menu in Figure 5-3 is also available to assist with quick navigation even when you're in other sections of APEX.
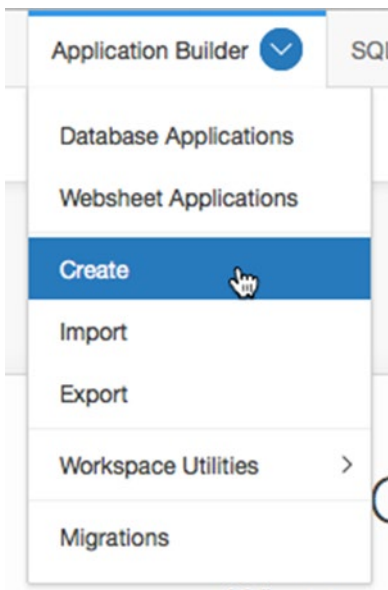
Figure 5-3. *The shortcut to creating an application*

Sample and Packaged Applications

If this is a new workspace, there may or may not be a sample application that was created automatically when the workspace was provisioned. The automatic installation of sample applications is a feature setting that can be configured by the APEX administrator. If a sample application isn't installed, you can install one manually by choosing Packaged Application in the first step of the Create Application wizard, as shown in Figure 5-4.
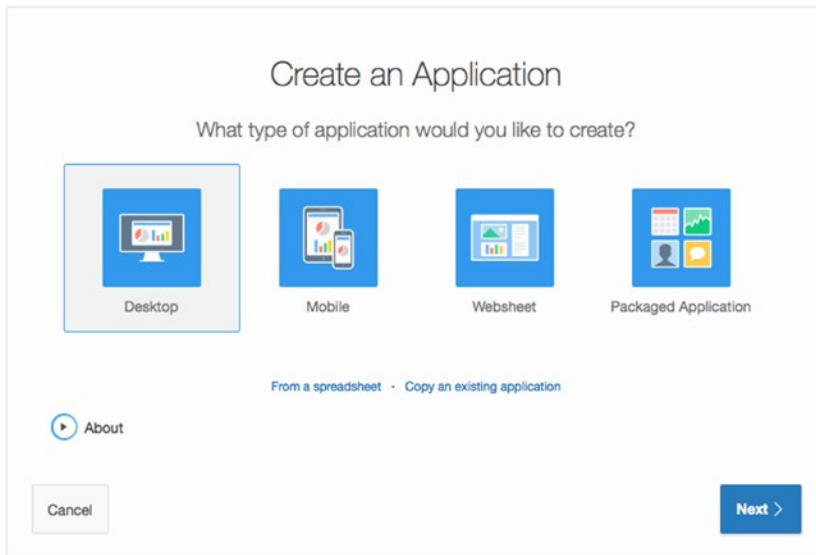
Figure 5-4. *Choosing the type of application*

The Packaged Apps section is where you can install and manage the applications that are bundled with the APEX distribution. The main page, shown in Figure 5-5, shows the Packaged Apps Home page, available from APEX's main navigation menu. From here you can see which applications are installed and can navigate to the three subsections.
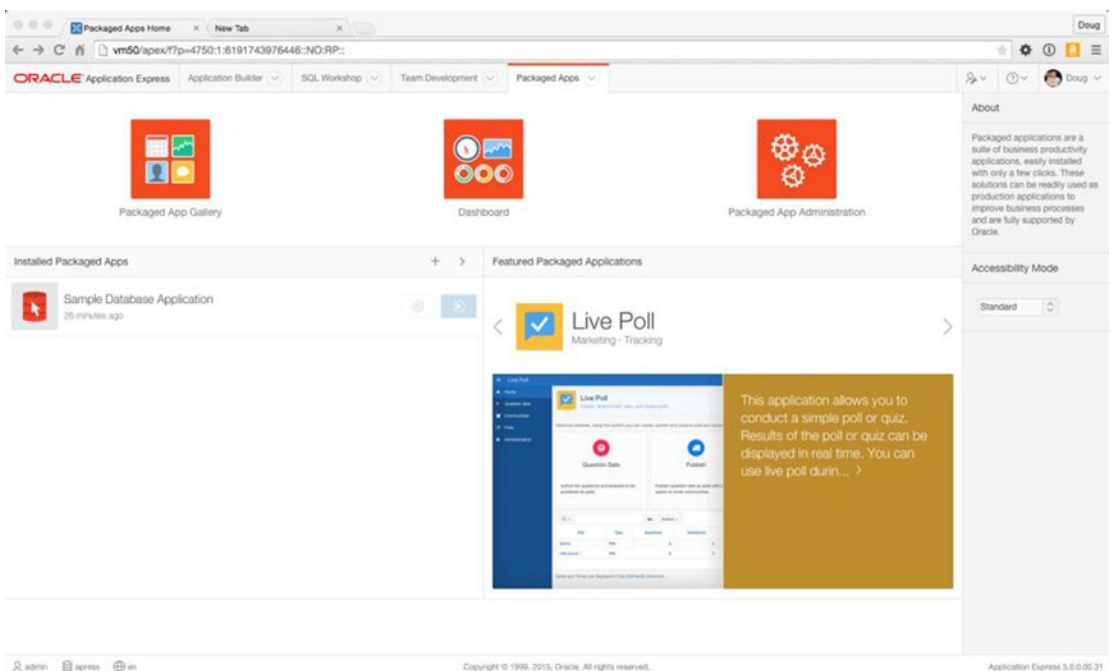
Figure 5-5. *The Packaged Apps Home page*

Packaged App Gallery

The Packaged App Gallery presents all of the applications that come bundled with the APEX distribution. There are 35 separate packaged apps that can belong to several different categories, including Software Development, Tracking, Team Productivity, Marketing, Knowledge Management, IT Management, Project Management, Sample Application, and Sample Websheet.

Clicking on the icon of an application takes you to a detailed information page for that application. Here, you'll be able to see a screenshot of the application, read its full description, and see version information for the app, as seen in Figure 5-6.
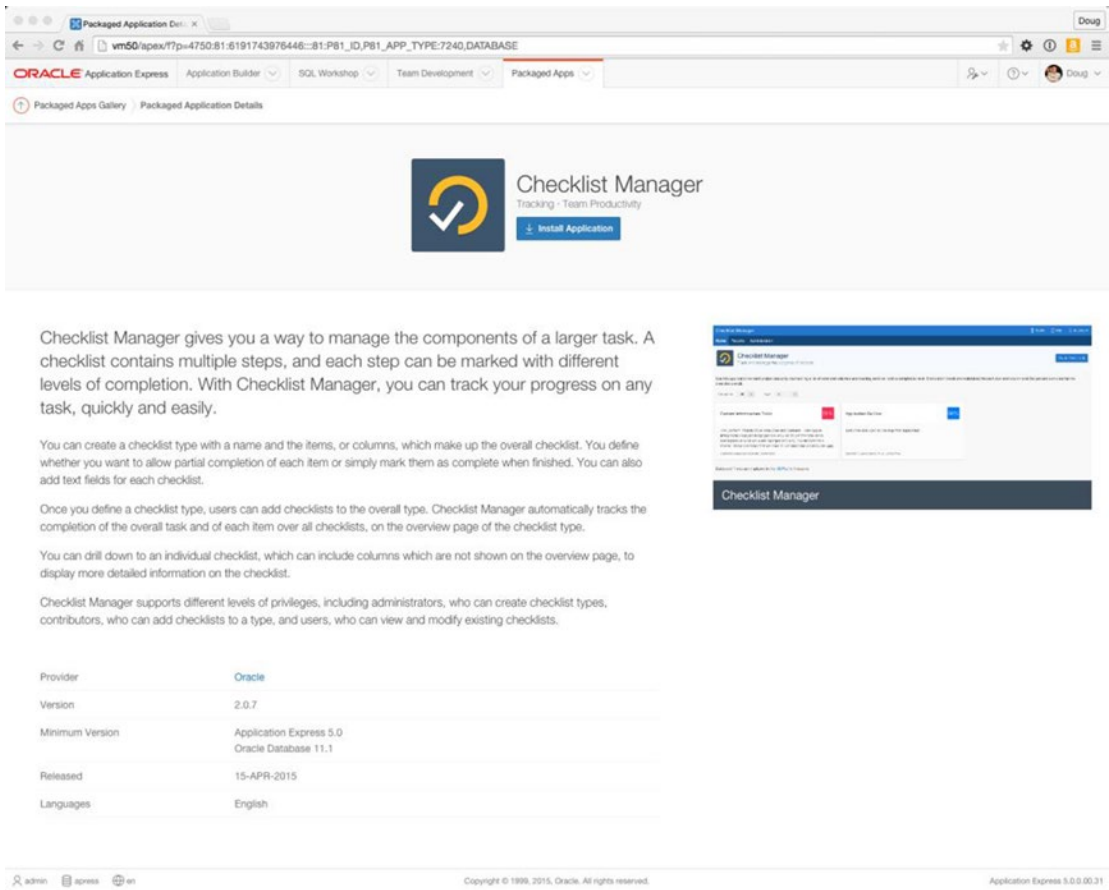
Figure 5-6. *The Checklist Manager Packaged Apps information page*

Clicking the Install Application button will step you through the process of installing the selected application in your current workspace. A pop-up installation wizard will present you with a choice of which authentication method to use; it normally defaults to Application Express Accounts. Clicking the final Install Application button in the wizard will install the application and any of its supporting objects, including any required database objects. Once the installation is complete, you're taken back to the application's information page where, as shown in Figure 5-7, you can see the application has been successfully installed, and you're given the options to manage and run the application.
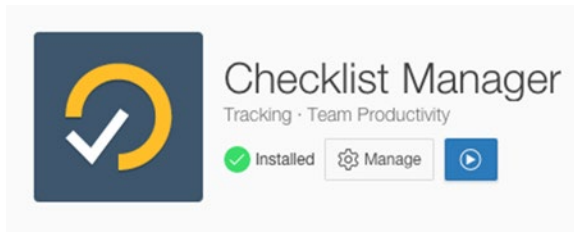
Figure 5-7. *See that the Checklist Manager App has been successfully installed*

There are a few things that you should know about Packaged Applications:

1. Any application that has "Sample" in the name is there to demonstrate functionality available in APEX and therefore will be installed in an *unlocked* state by default. This means that developers will be able to edit the application and see how the APEX team developed the app.

2. Applications that *do not* have "Sample" in the name are provided as production ready and will be installed in a *locked* state that does not allow any editing until the app is specifically *unlocked*. This can be done via the Manage button, as shown in Figure 5-7. Any of these applications that are installed and remain locked are fully supported by Oracle in a production environment. These locked applications can also be upgraded to more current versions that may come with future versions of APEX. The moment you unlock them, all support from Oracle ceases, upgradeability expires, and there is no way to relock an application.

3. As of APEX 5.0, all Packaged Applications are installed into the workspace's default "parse as" schema. Currently there is no direct way to install them in a secondary "parse as" schema without first unlocking and exporting the application, thereby voiding any support.

Even though the sample apps were written as learning aids, there is a lot to be learned from many of the production-ready applications as well. I heartily suggest your first act after finishing this book is to take a look at the inner workings of some of the packaged apps.

Packaged App Dashboard

As shown in Figure 5-8, the dashboard page presents an overview of the utilization of Packaged Apps in the current workspace. You're presented with the total number of available apps, the number installed, and whether there are any applications that can be upgraded. You can also see who has installed the apps and how frequently they are used.
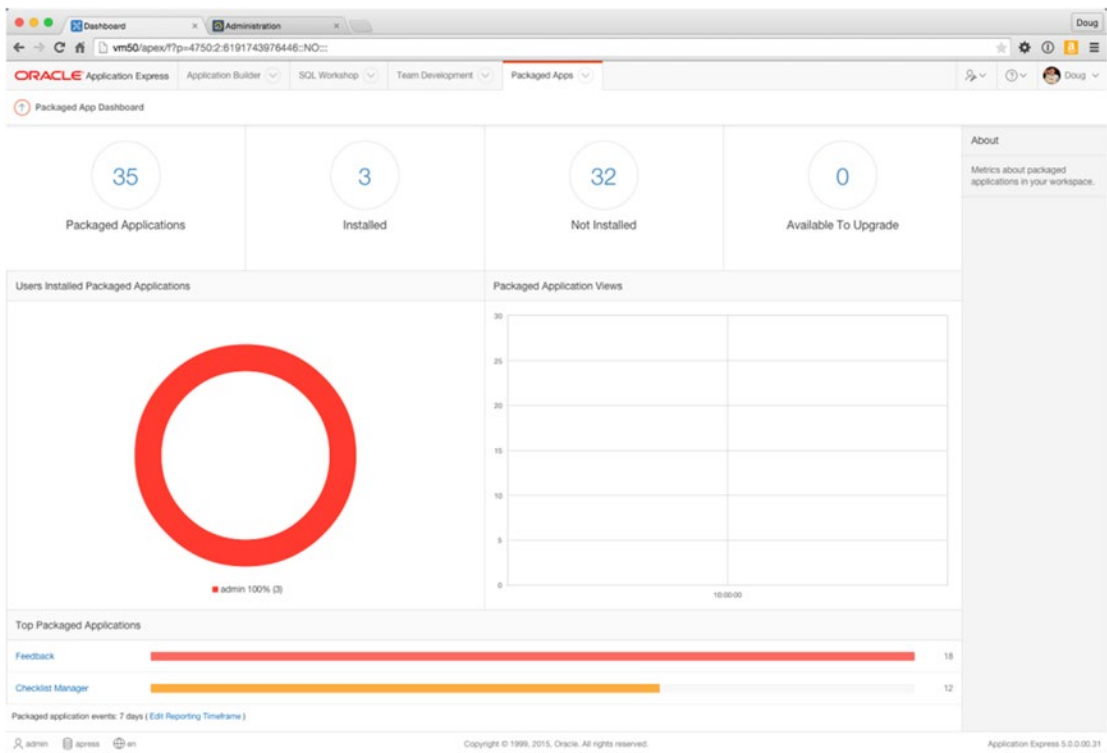
Figure 5-8. *The Packaged Apps dashboard*

Packaged App Administration

The Packaged App Administration page provides a list of administration tasks specifically related to the Packaged Applications installed in the current workspace. If you are logged in as a developer, you'll only see options relating to managing Interactive Report settings and Activity reports. However, when logged in as a workspace administrator, you'll see a section called *Manage Services* that shows a small subsection of what is available to you in the full Administration section.

Websheet Applications

This book will cover websheet application features in Chapters 11 and 12. The starting point for creating a websheet application is the same as for a database application. The primary difference is the creation of predefined database objects that support websheet applications.

Database Applications from Spreadsheets

When creating desktop applications from the wizard, you're quickly faced with a question: where is your data coming from? One of the links listed in Figure 5-4 lets you create an application based on data from an existing spreadsheet. If you choose this option, the Create Application wizard provides steps for loading data into a single table, and at the same time creates an application that allows you to manage and manipulate that data. The application is very simple, using a report and form combination such as that

shown in Figure 5-9. Creating a database application from a spreadsheet is a fast and easy way to get from a single-page spreadsheet to a working online application that can be expanded with additional tools and functionality.

Page	Name	Updated	Updated By	Page Type	Group	User Interface	Lock	Run
1	Report Page	Now	admin	Home	Unassigned	Desktop		
2	Insert Form	Now	admin	DML Form	Unassigned	Desktop		
3	Update Form	Now	admin	DML Form	Unassigned	Desktop		
4	Success Page	Now	admin	Static HTML	Unassigned	Desktop		
101	Login Page	2 seconds ago	admin	Login	Unassigned	Desktop		

1 - 5

Figure 5-9. *The application pages from a spreadsheet application*

Applications from Scratch

When you create an application from scratch, the wizard offers many interesting options. You can create any number of pages, and link pages to different tables of data. Additional steps give advanced options that, when planned for, are very powerful. Creating an application from scratch is the method used in the ticketing-application exercise. Here is what to do to begin the creation process:

1. Navigate to the Application Builder, and click the **Create** button to initiate the Create Application wizard.

2. Select **Desktop** as the application type, and click **Next**.

The following subsections describe the remainder of the creation process in detail. Each subsection contains one or more subsequent steps in the creation process. Read the descriptions and follow the steps as described.

Naming the Application

After selecting the Desktop application option, you're prompted for details of the application, as shown in Figure 5-10. The Schema select list exists for workspaces that have been granted access to more than one database schema, and it allows you to choose which schema you want your application to use as its "parse as" schema. The Name value is what you use to identify the application inside the builder and is used as the title of the application.

Figure 5-10. *Entering the application properties*

Application IDs must be unique across the entire instance of APEX, so it's best to leave the ID set to the number APEX has assigned.

The next options reference the APEX Theme and Theme Style. APEX *themes* are groupings of templates that are used to establish the look and feel of pages, reports, buttons, and other graphical components. As APEX and web standards evolve, so do the premade themes in APEX. Version 5.0 offers a groundbreaking new theme option called the Universal Theme, as well as includes a number of HTML5/CSS3-compliant themes and a responsive theme, along with legacy themes, some of which have been around for quite a long time.

Although APEX currently comes with 27 desktop themes of varying looks, it's always possible to customize an existing theme or to create a completely new one. The APEX administrator also has the ability to create themes that are specific to their instance of APEX. Choosing a theme as part of the Create Application wizard is an easy way to apply a default theme. As you might expect, you can change your mind later and apply a different theme. Additional themes can be added, modified, and tested as part of the shared components of APEX.

Figure 5-11 shows the APEX theme chooser. The select list at the top of the region dictates which themes to show. Your choices are as follows:

- *Standard Themes*: Currently, for APEX 5.0 this only shows the Universal Theme. All other themes within APEX are now considered "legacy themes."

- *Custom Themes*: Shows any custom themes that have been installed by the workspace or instance administrator. By default, there are no custom themes.

- *All Themes*: Shows all available themes across all the previous sets, including the 26 legacy themes.
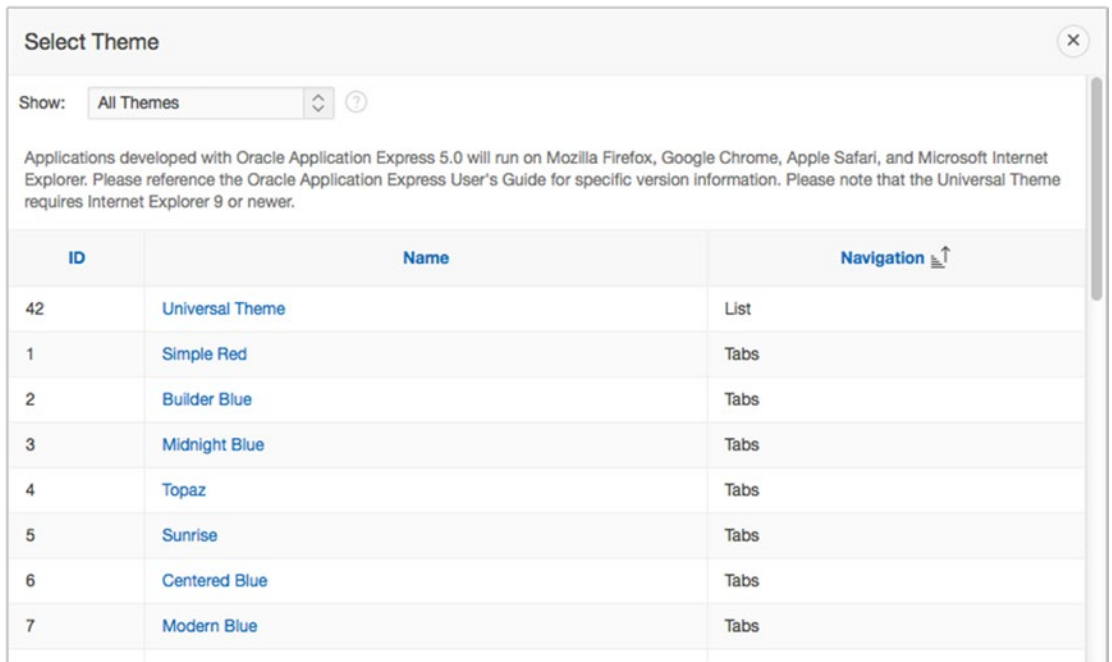
Figure 5-11. *Theme selection*

The Theme Style option is a subselection of the chosen theme and will present the styles available under that theme. Currently, only the Universal Theme has related theme styles.

Having reviewed the themes, continue the creation process by choosing the Universal Theme for your example application. Follow these steps:

3. Enter Help Desk for the **Name**, making sure your **schema** is set.

4. Select **Universal Theme (42)** from the **Theme** select list, and then choose **Vita** as the **Theme Style** for your application.

5. Click **Next**.

Laying Out Pages

The next step in the wizard is to decide which pages you need for your application. The wizard requires at least one page to be created, but Figure 5-12 shows that you have the option to create as many pages as you like.
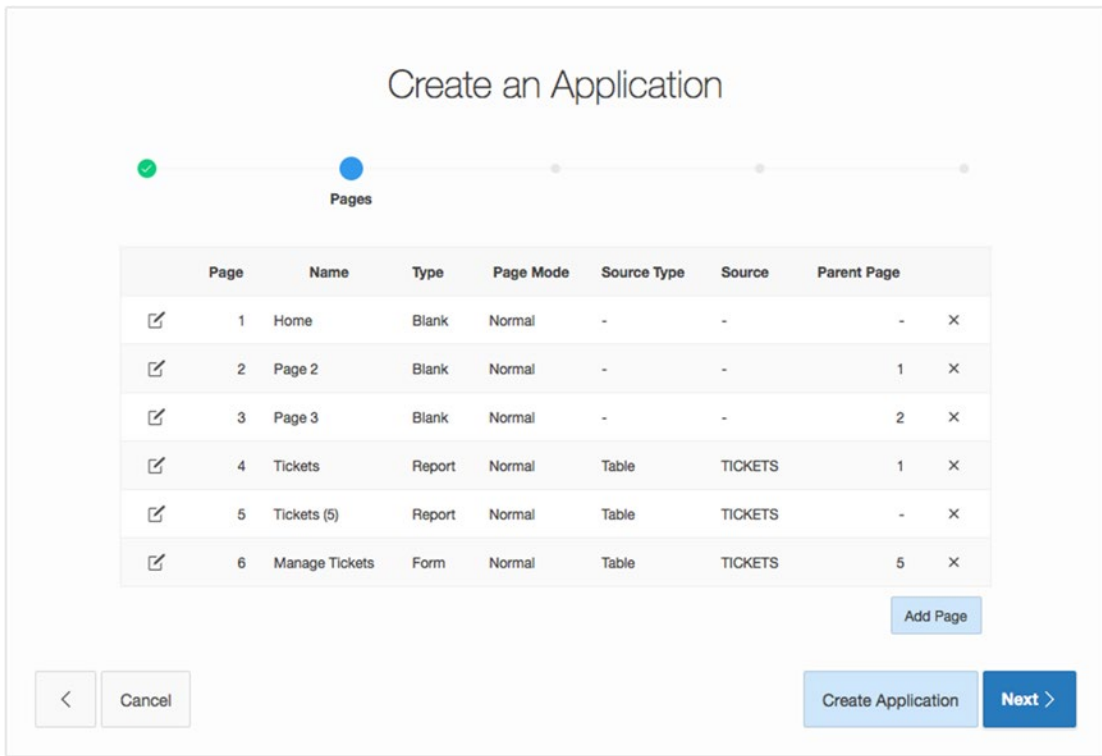
Figure 5-12. *Multiple pages defined in the Create Application wizard*

The Add Page Button at the bottom of this page invokes a pop-up wizard that allows you to define pages of varying types. Each page type calls for different information to be provided. For instance, adding a report page prompts you to select either a table name or a query on which to base the report. Choosing a chart requires a chart type and a query for the initial data series.

For now, you'll stick with the blank home page and create the rest of the pages later as needed. Thus, the next step is simple:

6. An application home page has already been created. Accept the defaults on this page and click **Next**.

Copying Shared Components

The next screen asks whether you wish to copy shared components from another application. This comes in handy if you have a template application that houses components that are shared across applications in the same workspace. Copying shared components isn't an advanced procedure, but it does lend itself to a

controlled and mature development process. This step in the wizard is a convenience, because the same objects can be copied in other ways after the application has been created. You don't need this step, because you're creating an application from scratch. Skip the step as follows:

    7.   Select No for **Copy Shared Components from Another Application**, and click **Next**.

Application Attributes

The next step in the wizard allows you to set some of the application-level attributes, such as the type of authentication to use and globalization attributes, including from where to derive the primary language, date formats, and so on. Let's look at each of these individually so you can gain a full understanding of the ramifications of each.

Selecting an Authentication Method

With every application, you need to make a choice about authentication, even if that choice is no authentication at all. This topic is discussed further in Chapter 9. By default, the APEX Create Application wizard provides three options for authentication:

- *Application Express Accounts*: Users and passwords are local to the APEX workspace. These users are managed in the same way the developer accounts are managed inside the APEX workspace, and users only work inside the current workspace.

- *Database Account*: This option uses the Oracle Database schema user names and passwords for credentials. Some organizations use this type of database-driven authentication to keep track of users. The application still executes as the chosen "parse as" schema, not as the individual user in the database.

- *No Authentication*: This is like a public website. Users aren't prompted for any type of authentication. This is useful for informational applications where the question "Who are you?" isn't important.

    For simplicity, the default is to use the Application Express authentication scheme. This is the one setting that provides login security; by default, the developer writing the application can log in without any additional work.

■ **Note**   Many organizations have an existing method of authenticating users. If an LDAP server is currently available (such as Oracle Internet Directory, Microsoft Active Directory for network domain authentication, or even an Oracle E-Business suite), you may want to use this system for APEX authentication. The number of options and methods is beyond the scope of this book. Simply know that with the Oracle Database technology and the technology of your application server, it's possible to use many of the most common authentication infrastructures.

Selecting Tab Options

Tabs are a common navigational structure for web applications and have been supported by traditional APEX themes since very early versions. They provide an intuitive interface for switching subjects or general areas in an application. Three options are available:

- *No Tabs*: This is a basic page style where no tabs are generated by the wizard and no tabs are displayed by the page template. This is often selected for small applications or applications where navigation is managed by a different method, such as lists, buttons, or other template constructs.

- *One-Level Tabs*: This is the most common style of tab layout; it's useful for small- to mid-sized applications where functionality needs to be separated yet easily accessible. This is also the easiest type of tab style to manage.

- *Two-Level Tabs*: The construction of two-level tabs uses a parent tab construct and breaks the standard tabs into tab sets. It's similar to having a controlling tab.

Legacy themes within APEX support up to two-level tabs in the display templates provided, and the wizard builds the shared components for the tab set as part of the wizard. If you know your application's page outline and can lay it out during the creation of the application, the wizard will do most of the tab setup. Designing the page at creation time can be a big timesaver if the application design calls for a significant number of tabs. In any case, you can create and modify the shared component after the initial run of the Create Application wizard.

■ **Note**   The new Universal Theme does not use tabs for navigation, but instead uses nested static lists. Therefore, when you choose to use the Universal Theme, the Tabs option on the Application Attributes page of the wizard will be absent.

Globalization Options

The authentication step in the wizard also includes six additional settings, as shown in Figure 5-13. A few of the settings have to do with the ability to translate the application to other languages. Multilingual applications are beyond the scope of this book, but for completeness the general usage descriptions of these options are included.

Figure 5-13. *The Attributes page of the Create Application wizard*

These settings are as follows:

- *Language*: This is the language the application uses by default. It's also used as the basis for any internationalization and translation in the case of multilingual applications.

- *User Language Preference Derived From*: For multilingual applications, this setting determines how the application derives the translation that is necessary.

- *Date Format*: This option sets the default of how date elements are formatted within the application. Different regions of the world have assumptions about how dates are formatted, especially when they're strictly numeric values. A common format that is used to try to alleviate this issue is the DD-MON-YYYY format. This style of format makes it clear which portion represents the day, month, and year (for example, 01-JAN-2010).

- *Date Time Format*: This option sets the default formats of dates that include a time dimension.

- *Timestamp Format*: This option specifies the format used for timestamp datatypes used throughout the application.

- *Timestamp Time Zone Format*: This option specifies the format used for timestamp datatypes with time-zone data used throughout the application.

The wizard uses these settings as starting values. You can alter them as needed in the shared components of the application.

The language, date format settings, and time zone handling are classified as globalization settings. After the application is created, you can turn on automatic time-zone detection; this setting is found on the Globalization tab of the Application Settings. Automated time-zone detection is especially useful for applications whose users span different time zones.

Continue creating the example application as follows:

8. Set **Authentication Scheme** to **Application Express**, **Language** to **English** (**en**), and **User Language Preference Derived From** to **Application Primary Language**.

9. Choose 12-JAN-2004 (returns **DD-MON-YYYY**) for **Date Format** and 12-JAN-2004 14:30:00 (returns **DD-MON-YYYY HH:MI:SS**) for **Date Time Format**, and leave the last two options blank.

10. Click **Next**.

Completing the Create Application Wizard

The last step of the wizard is a simple confirmation dialog. Clicking the Create Application button seen in Figure 5-14 commits all the settings and generates the application. The Previous button lets you walk backward through the wizard to make any additional changes before you complete the process.
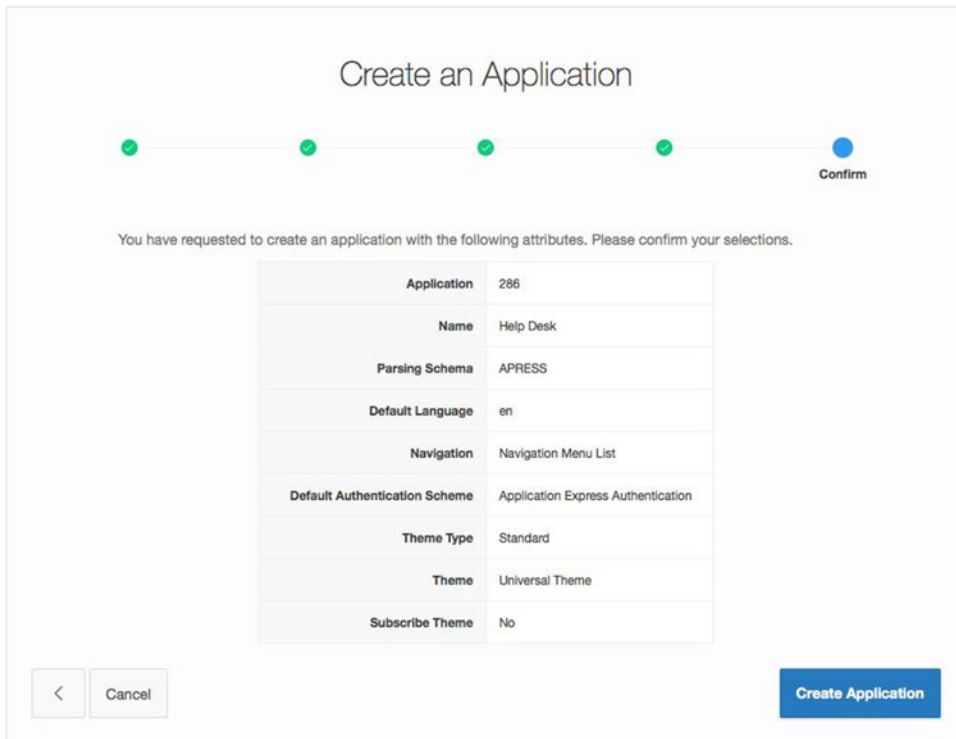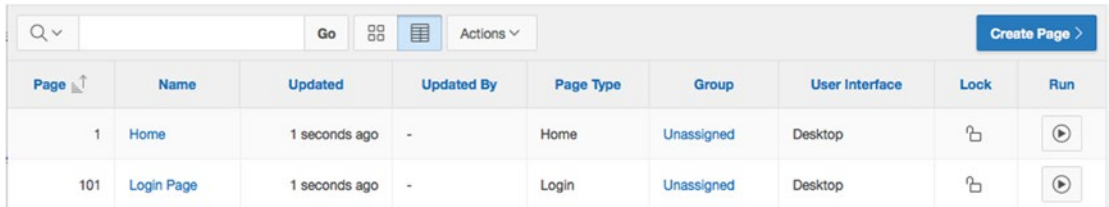
Figure 5-14. *Completing the Create Application wizard*

Complete your creation of the example application by executing the final step in the process:
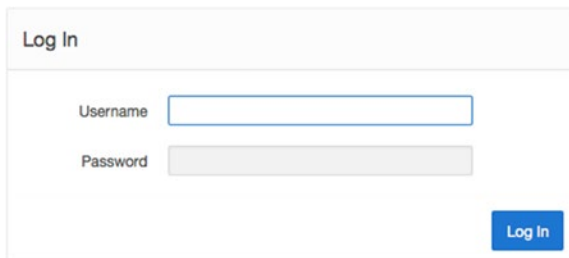
11. Review the wizard's summary page and confirm the choices you've made by clicking **Create Application**.

You now have a simple application with only two pages, as shown in Figure 5-15 (View Report view). Run that application, and you should see the login page shown in Figure 5-16. That login page takes your normal APEX developer user name and password. Once logged in, you'll be taken to the application's Home page, as shown in Figure 5-17.

Page ↑	Name	Updated	Updated By	Page Type	Group	User Interface	Lock	Run
1	Home	1 seconds ago	-	Home	Unassigned	Desktop	🔓	▶
101	Login Page	1 seconds ago	-	Login	Unassigned	Desktop	🔓	▶

Figure 5-15. *Resulting pages for the Help Desk application*

Figure 5-16. *Login prompt when running the application*
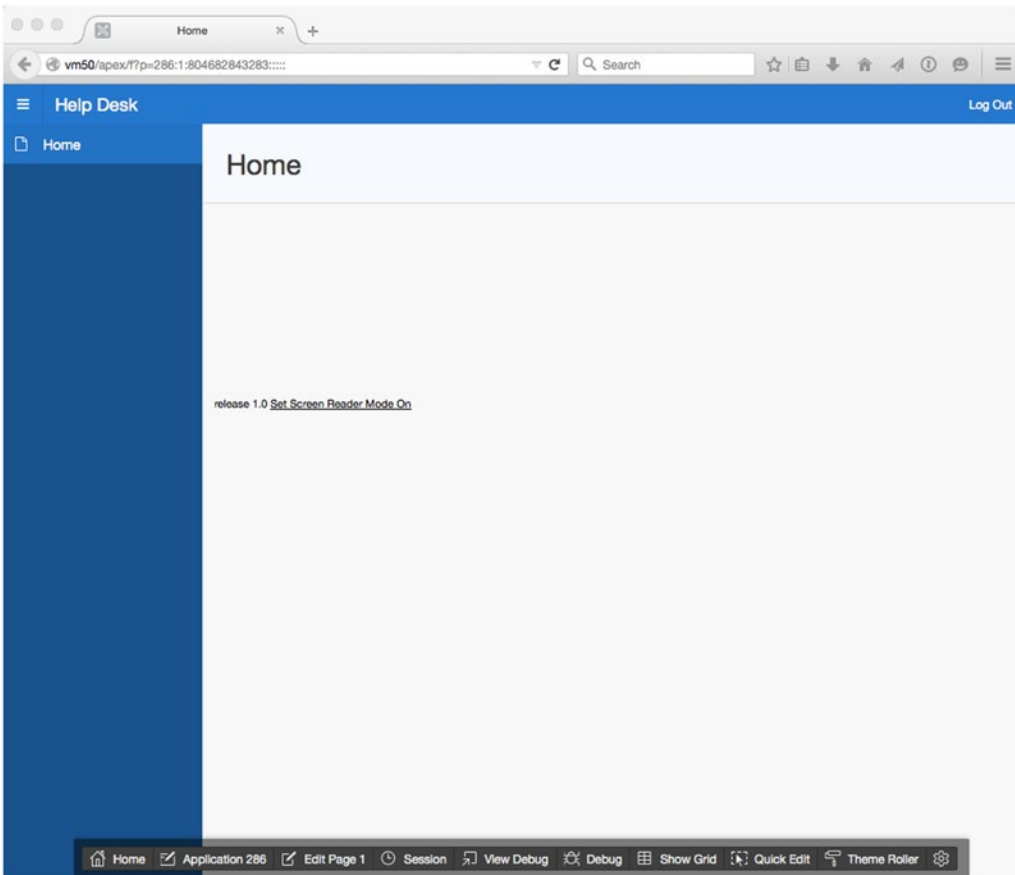
Figure 5-17. *The application after you've logged in*

Now that you have the shell of the application created, you can move forward in extending it by adding other pages, regions, and items.

Static Content Regions

The Static Content region type is one of the most basic and yet most flexible types of region. By manipulating the attributes of a Static Content region, as shown in Figure 5-18, you can control how that region is displayed:

- *Output As*: Selecting HTML will interpret any markup as entered in the region source as HTML and will render the resulting output. Selecting Text (escape special characters) will escape (not interpret) special characters such as <, >, & when emitting the region source to the page. Example: <br /> will show up exactly as the code <br /> rather than being interpreted as a break or return.

- *Expand Shortcuts*: Enables or disables support for shortcut technology. This technology includes a shared component object that can be used for managing a type of variable using SHORTCUT_NAME syntax.
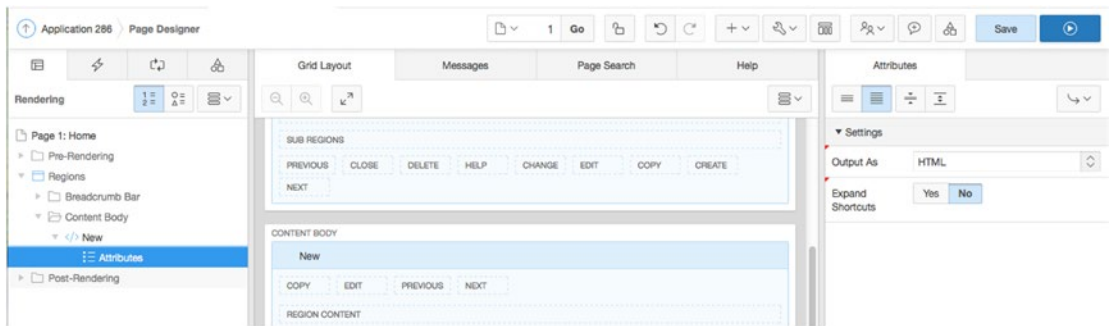
Figure 5-18. *Viewing the attributes of a Static Content region*

With the Static Content region's simplicity comes a wide variety of uses. A Static Content region is a container that can have its own value, embedded JavaScript, or CSS definitions, or it can contain other page items. Any valid HTML entered in the source is rendered on an APEX page. Substitution-string syntax, such as &ITEM_NAME., can also be used to display item values in the source text.

Continuing with the Help Desk application, add some content to the first page:

1. Navigate to the Application Builder, and **Edit** the Help Desk application. Depending on how you're viewing the applications report, you may need to click the icon as shown in Figure 5-19 or click the name of the application as shown in Figure 5-20.
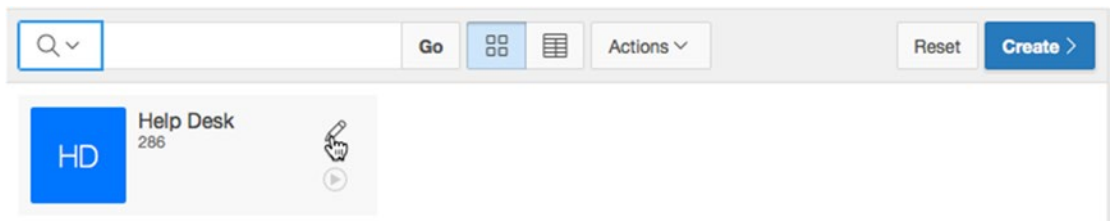
Figure 5-19. *Edit the Help Desk application from the icon view*

Figure 5-20. *Edit the Help Desk application from the report view*

2. Edit the **Home** page by clicking the link for the page name in the report, as shown in Figure 5-21.

Figure 5-21. *Editing the Home page*

    3.   From the Gallery in the lower center of the screen, select **<u>Regions</u>** for the component type.

    4.   Click and drag the **Static Content** icon from the Gallery into the Grid Layout section of the screen and drop the component in the **Content Body** content area, as shown in Figure 5-22.
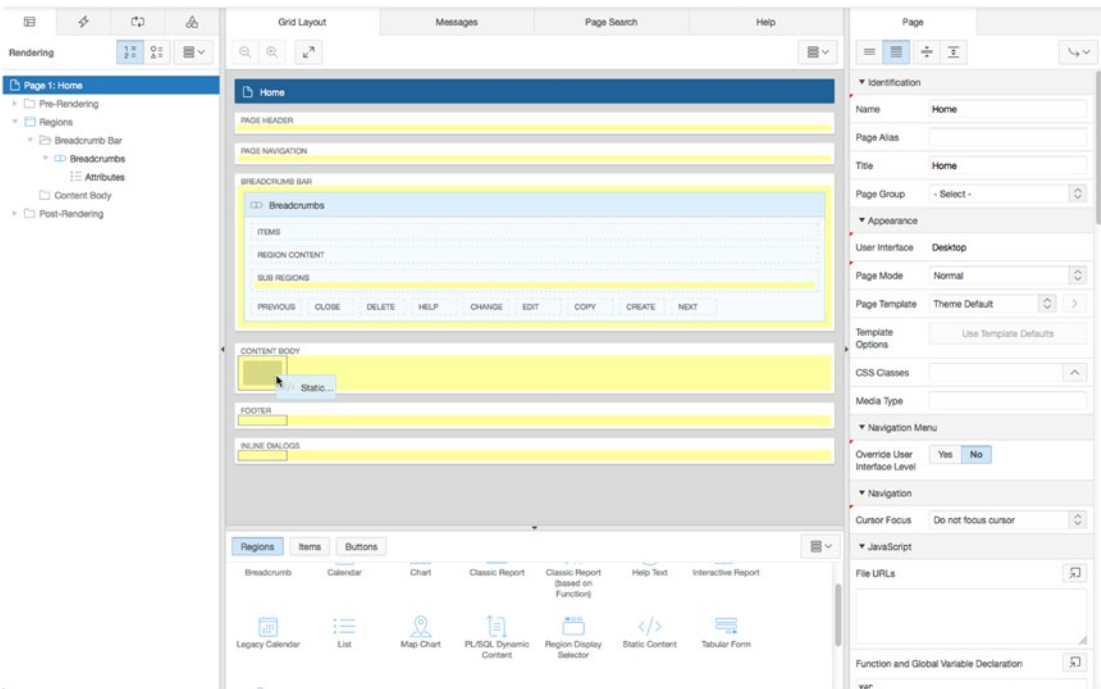
Figure 5-22. *Dragging the Static Content component to the Content Body area*

Once you've dropped the component, the view will change, showing the new region in place within the Grid Layout and selected as current within both the Tree Pane and the Grid Layout, as shown in Figure 5-23.
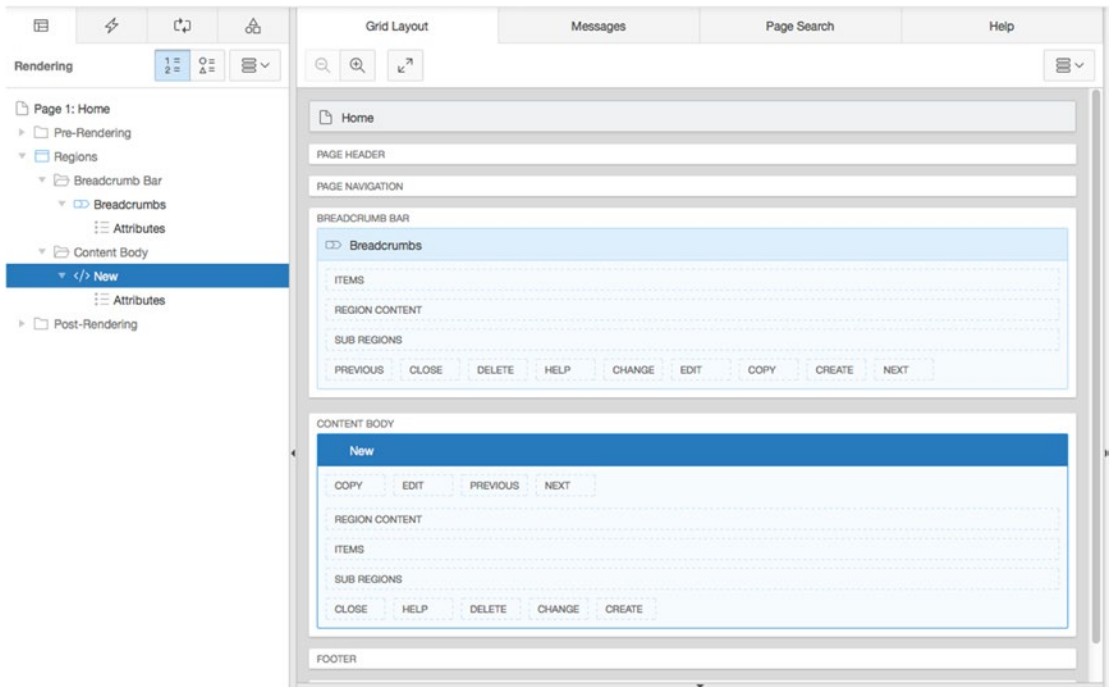
Figure 5-23. *The new Static Content region shown in the Page Designer*

Now, edit the attributes of the new region:

5. In the Attributes Pane, under the **Identification** section, set the **Title** to APEX Issue Tracker.

6. In the Attributes Pane, under the **Source** section, enter the following for the **Text** and then click the **Save** button at the top of the page. See Figure 5-24.

```
<h1>Welcome to the APEX Issue Tracking System</h1>
<br />Select an option from the list
```
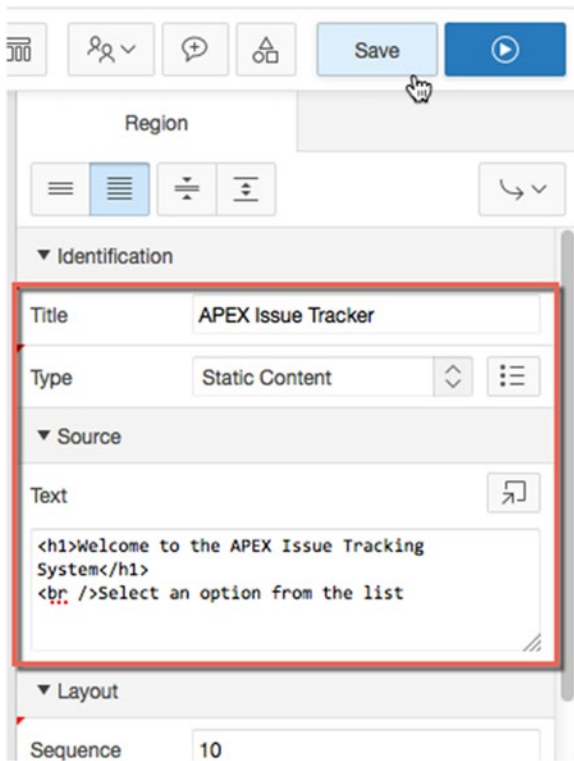
Figure 5-24. *Entering the Title and Text and saving your work*

Run the page by clicking the Run button at the top of the Page Designer. You should see the changes you just made indicated by a new region with a friendly welcome message. Your results should be similar to those shown in Figure 5-25.
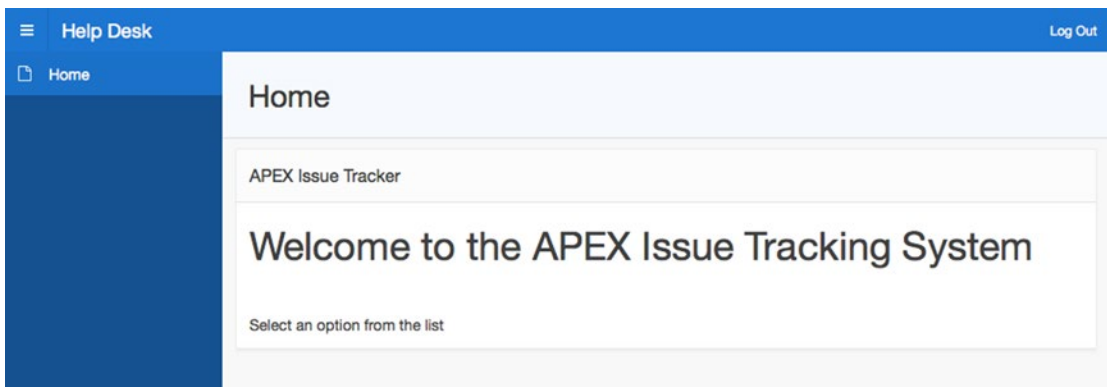
Figure 5-25. *Results after adding the Static Content region*

Public Pages

As mentioned, it's possible to allow the entire application to use no authentication scheme. But what if you want some of the pages to require authentication, and others to be public? How can you make a page that doesn't require a login in order to view it?

If any of the pages in an application require authentication, an appropriate authentication scheme must be applied to the whole application. APEX lets you define individual pages as Public or Requires Authentication using a defining property of the page. Each page can have different security requirements (authorization), but only one authentication mechanism can be applied to an application. Public pages are useful for introductory landing pages, login pages, and information pages.

In the Help Desk project, you want to have the main page available to all visitors. To accomplish this, you can modify the first page of the application to allow it to be seen by anyone without requiring authentication. Do that via the following steps:

1. In the Help Desk application, navigate to and edit page 1.

2. In the Tree Pane, edit the page attributes by clicking the page name (Home) in the **Rendering Tree**. The page name appears as the root node of the tree, as shown in Figure 5-26.
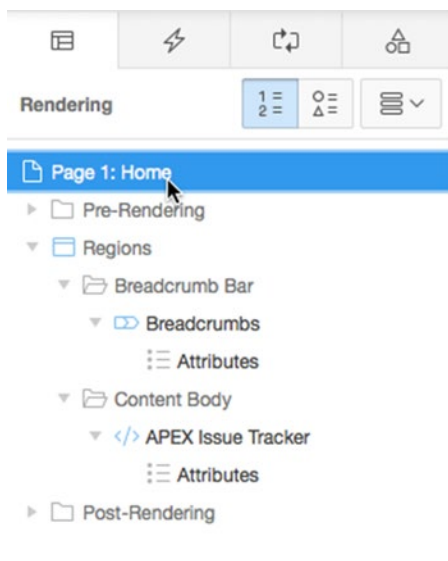
Figure 5-26. *Selecting the page node in the Rendering Tree*

3. In the Attributes Pane, scroll to the **Security** section, shown in Figure 5-27. In this section, change **Authentication** to **Page Is Public**.
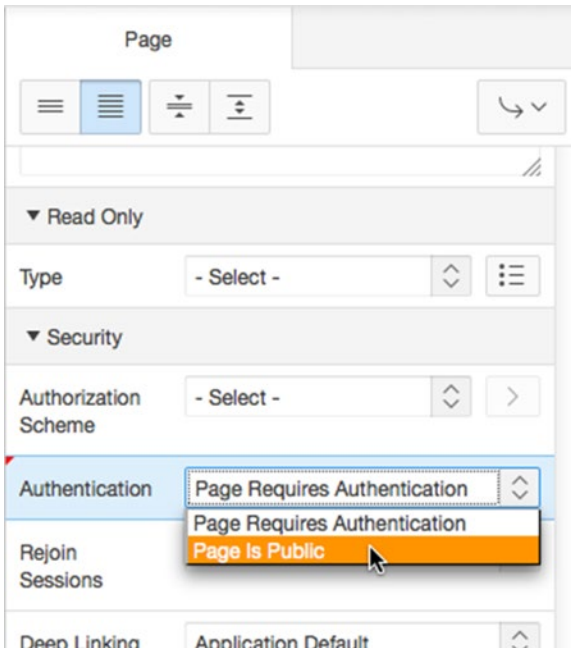
Figure 5-27. *Changing a page's authentication setting*

>   4.   At the top of the page, click **Save**.

Now, when the page is run, the authentication screen doesn't appear when page 1 is requested. You will learn more about authentication and authorization in Chapter 9. For now, just know that the change you've made allows users to see the first page of the application without being logged in.

Navigation Bar Entries

Each APEX application has one navigation bar that may contain multiple entries. Examples of links typically displayed on every page are Login, Logout, Help, and My Account. As a developer, you can create and modify navigation bar entries depending on the application and need. The navigation bar can also go beyond standard link text; it can be modified to include images. Entries can be based on conditions, authorization schemes, and build options. Placement of navigation bars is dictated by the page template substitution variable #NAVIGATION_BAR#. In most applications, the navigation bar is placed either at the upper right or upper left of the page.

The example application already has a very simple navigation bar that has been created for you, as shown in Figure 5-28. It currently contains only a Log Out link.

Figure 5-28. *The basic navigation bar*

Because you've modified the Home page to be a publicly viewable page, you need to add a navigation bar entry that allows users to log in. At the same time, you need to make both the Login and Log Out links context sensitive so they're only displayed when it makes sense. (For instance, the Log Out link should only be displayed when a user is actually logged in.)

Navigation bars are part of an application's shared components, so they're created and maintained from the Shared Components section of the Application Builder. Create one in the example application as follows:

1. From the Page Designer, click the Shared Components icon in the upper-right section of the page, next to the Save button, as shown in Figure 5-29.
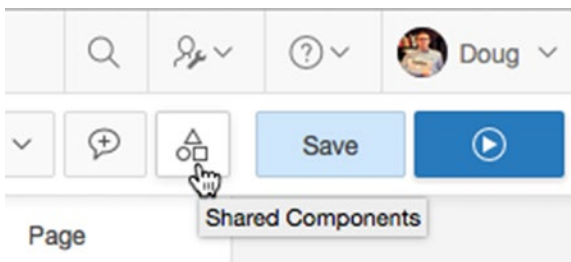
Figure 5-29. *Navigating to the Shared Components screen from the Page Designer*

2. Under the **Navigation** section, click **Navigation Bar List**, as shown in Figure 5-30.
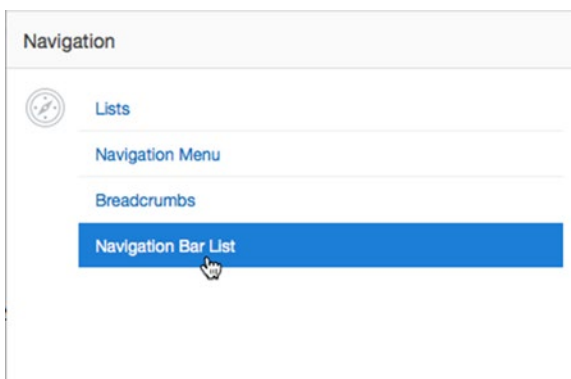
Figure 5-30. *Navigation components in the Shared Components screen*

You'll see a report showing that a Navigation Bar List called *Desktop Navigation Bar* already exists. You will need to edit this list to add your Login entry and to edit the Log Out entry.

   3. Click the **Desktop Navigation Bar** link in the report.

   4. Click the **Create List Entry** button in the upper right of the screen.

   5. In the **Entry** section of the page enter Login in the **List Entry Label** field.

   6. In the **Target** section, set **Target Type** to **Page in This Application**.

   7. For **Page**, enter 101. This will send the user back to the login page after they've logged out. See Figure 5-31.

Figure 5-31. Navigation bar settings

   8. In the Conditions section, set **Condition Type** to **User is the Public User (user has not authenticated)**, as shown in Figure 5-32.

Figure 5-32. Navigation bar conditions

9. Click **Create List Entry** at the top of the page.

Run the application now. If you're logged in, you only see the Log Out navigation bar entry. Click the Log Out link. Once you're logged out, you see the new navigation item, as shown in Figure 5-33. This identifies a small problem: the Log Out link can still be seen even though you've already logged out.
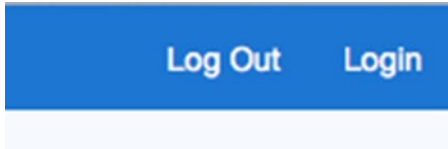
Figure 5-33. *Login and Log Out links both showing*

Clearly, it's a problem to show the Login and Log Out choices at the same time. After all, only one of those two choices can apply. Let's tackle that problem:

1. Navigate back to the **Shared Components** section for the Help Desk application.

2. Edit **Navigation Bar List**, and then edit the **Desktop Navigation Bar** list.

3. Edit the Log Out navigation bar entry by clicking on its name in the report.

4. In the **Conditions** section of the page, set **Condition Type** to **User is Authenticated (not public)**, as shown in Figure 5-34.

Figure 5-34. *Navigation bar condition type*

5. Click **Apply Changes**.

Run the application again. You should see that the Login and Log Out navigation items are mutually exclusive. When you created the new navigation item, you applied the condition to allow it to be seen only by the public user. The Log Out navigation item was created as part of the Create Application wizard; no condition was placed on the Log Out item by default. You will learn more about conditions in Chapter 8.

Global Pages

A *Global Page* is a special type of page that acts as a "master page" for your application and can be added one per user- interface type (that is, you may have one Global Page for the Desktop UI and another for the Mobile UI).

Items placed on a Global Page are rendered on every page in its related UI for that application unless conditionally told to do otherwise. This is particularly useful when you identify the need to display the same region on multiple pages or even on all pages in your application. Simply move a region to your Global Page, and it's rendered with every page.

A good example of usage is a breadcrumb region or a region that contains custom JavaScript code that needs to be available to every page. Region contents from a Global Page are included on every page of that UI, even when a region doesn't render visibly.

Although you can assign any page number to a Global Page, the default page number for a Global Page related to a desktop interface is zero (0). In fact, Global Pages take the place of what used to be called Page Zero in previous versions of APEX.

You may notice when looking at the definition of a Global Page in the APEX Page Designer (Figure 5-35) that there are no nodes in the Tree Pane for the Processing tab. Global Pages are only used during page rendering. Regions that are added to a Global Page are included even on the Login page. You need to consider the different page types in an application when adding content to a Global Page.
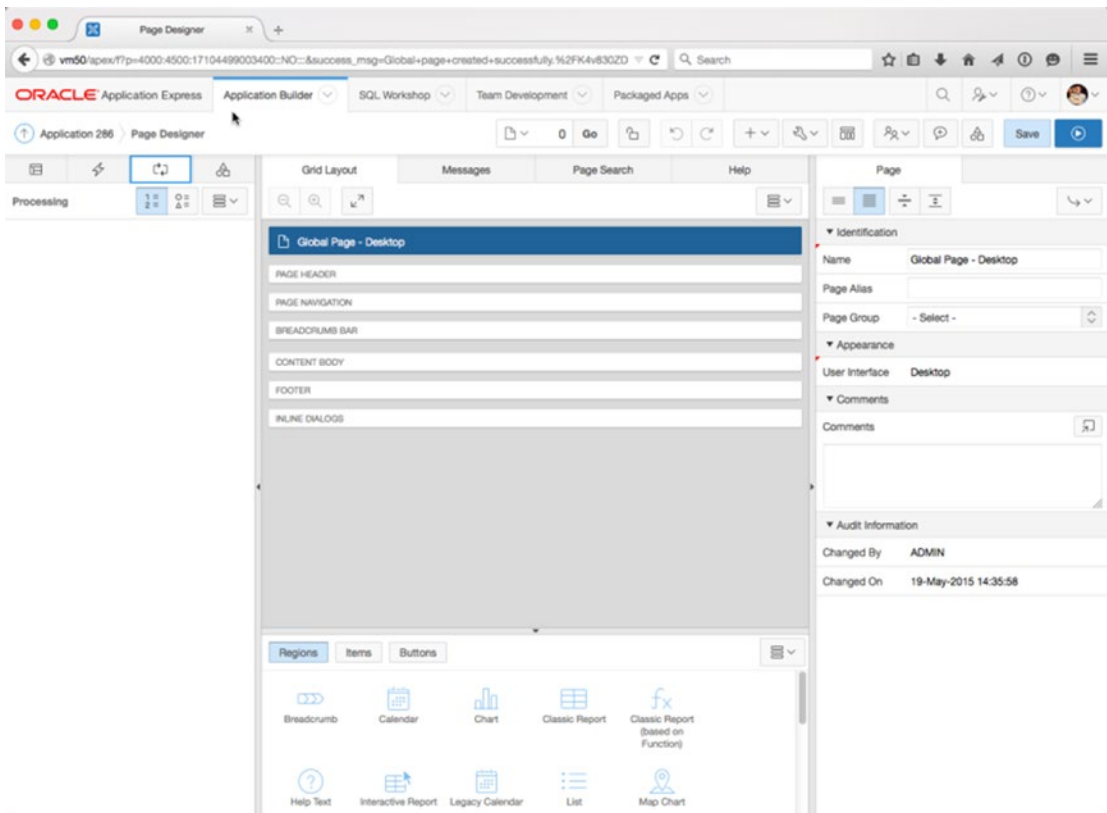
Figure 5-35. *There are no Processing nodes for a Global Page, as shown in the APEX Page Designer*

Creating a Global Page is like creating any other page in an APEX application. However, once it's created, it's no longer available in the Creation Options list for that UI type. Let's create a Global Page for the desktop interface:

1. From the **Application page list**, click the **Create Page** button.

2. In the resulting pop-up wizard, select **Global Page** from the **Page Type** list and click **Next**. Figure 5-36 shows the Global Page option, which should be near the bottom of the list.
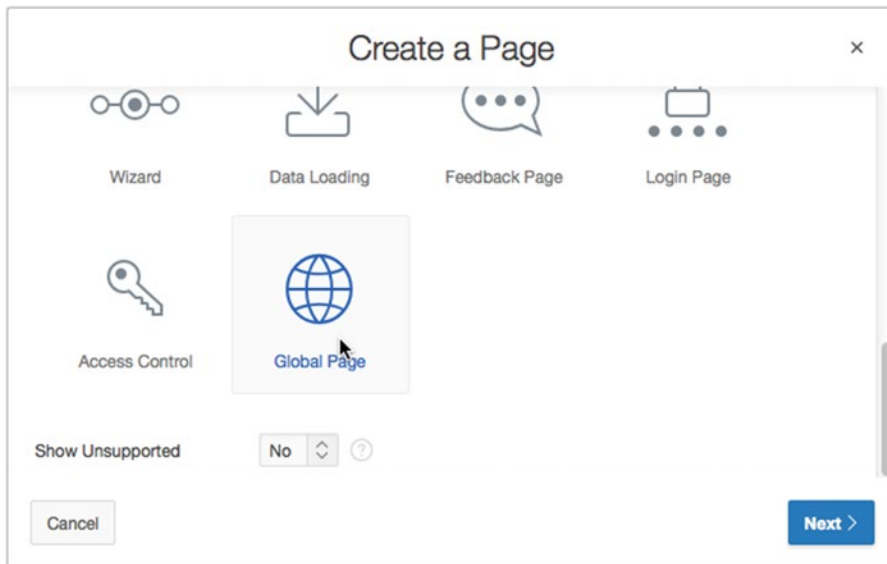
Figure 5-36. *Choosing to create a Global Page*

3.   Leave **<u>Page Number</u>** set to **0** (zero), and click **Create**.

You should now see your Global Page listed in the pages for the application. Currently, there is no content on the Global Page. You will use this Global Page to contain and display the breadcrumb region covered in the next section.

Breadcrumb Regions

Breadcrumbs are a popular navigation structure. They give the user a quick and intuitive representation of the current navigation path with optional functionality to navigate back using the structure. Oracle Application Express uses the structure in the builder shown in Figure 5-37.

Figure 5-37. *Example of breadcrumbs in the Application Builder*

In APEX, breadcrumbs are a declarative structure with built-in behavior. They're managed as shared components and have their own region type and template. When you ran the Create Application wizard, the pages that the wizard created automatically included a region to contain the breadcrumbs. Figure 5-38 shows the Breadcrumbs region in the Breadcrumb Bar section of the page.
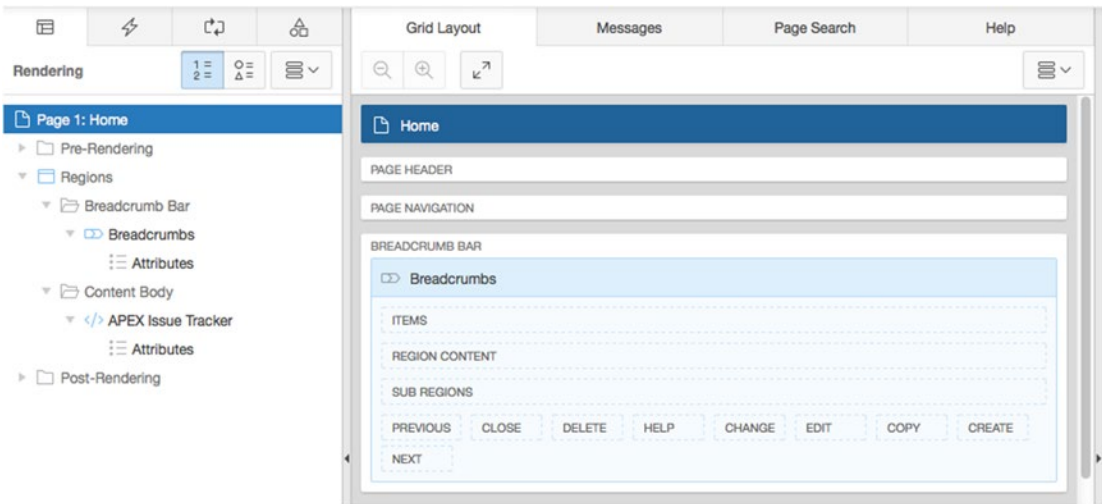
Figure 5-38. *The Breadcrumbs' position in the page-rendering hierarchy*

When you're creating new pages for an application, the Create Page wizard has an option to assist in creating new breadcrumb entries. When you use this option, child pages receive a copy of the breadcrumb region from the parent, as well as an automatic entry in the Breadcrumb group. When a breadcrumb region doesn't exist, nothing is copied, but the entry in the breadcrumb shared component is still created. An issue with this approach is that if you need to make any changes to the region's display or other layout considerations, they have to be done manually on every page that contains a breadcrumb region. Adding the region to a Global Page to make it appear on all pages can be helpful, because it gives you one point of change instead of many.

Continuing with the Help Desk application, the design is supposed to have a breadcrumb region that appears on all pages. It isn't necessary to re-create the region manually. Because the Create Application wizard created the region for you, you can use the Copy Region feature in APEX to duplicate the region to your Global Page. Do the following:

1. Edit **Page 1** using the Page Designer.

2. Right-click the **Breadcrumbs** region in the Rendering Tree Pane to show the context menu, as shown in Figure 5-39.
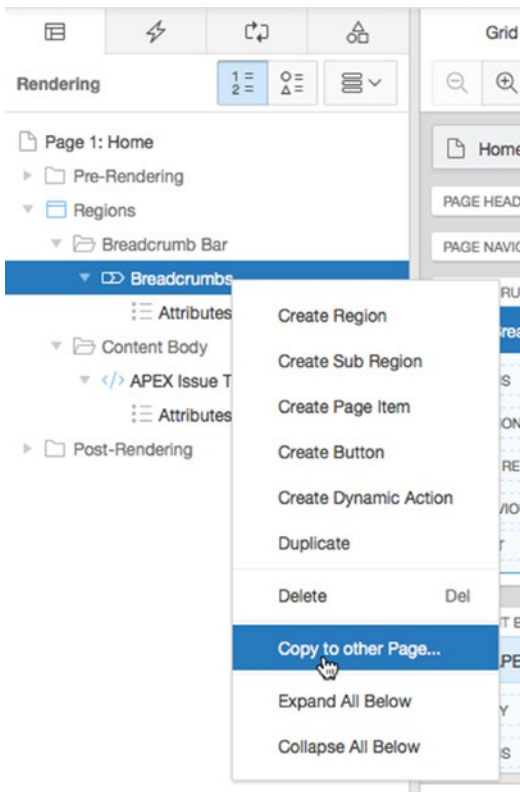
Figure 5-39. *Context menu for the Breadcrumbs region*

3. Select the **Copy to other Page...** option.

4. Change the page number for the new region to **0**, as shown in Figure 5-40, and click **Next**.
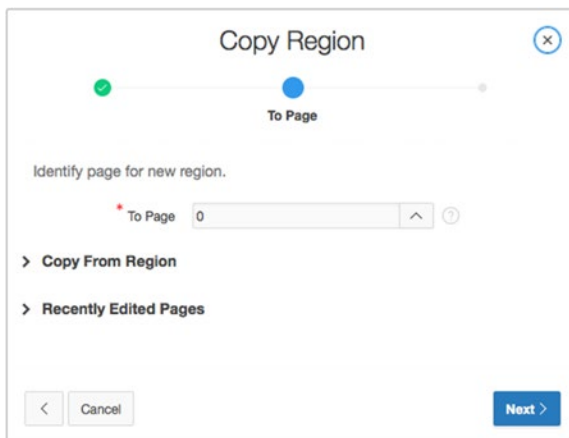
Figure 5-40. *Setting the destination page*

The Copy wizard allows the modification of what is copied in a limited fashion. Options that don't apply are disabled. In the current example, you could modify the region name and sequence as well as some display-placement options. For now, leave them with their default values:

    5.   Confirm the settings shown in Figure 5-41. Click **Copy** to complete the wizard.
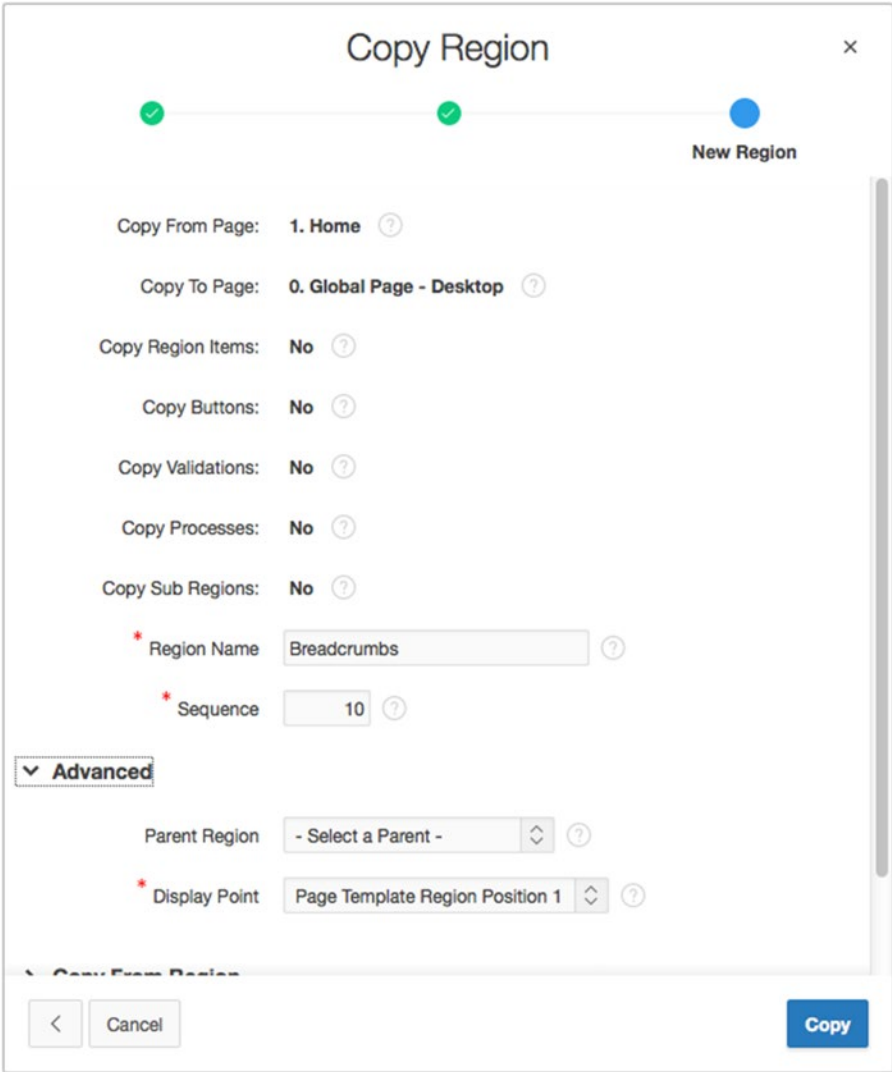
Figure 5-41. *Confirming the copy operation*

Reviewing the change in the Page Designer, notice that the Global Page now has the new breadcrumb region, but the original breadcrumb region still remains on page 1. In running the application, you can see the two breadcrumb regions shown in Figure 5-42. Note that the Copy feature doesn't remove the existing breadcrumb region.
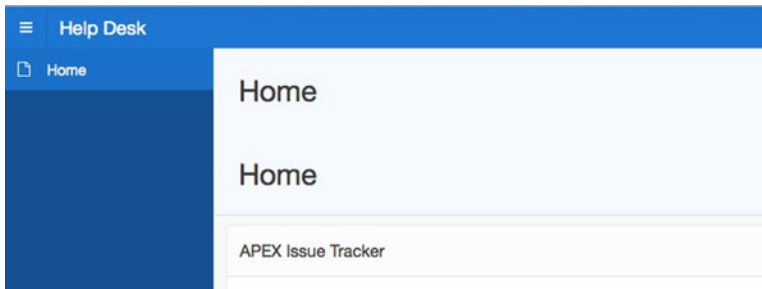
Figure 5-42. *Redundant breadcrumb regions*

To correct this duplication, do the following:

1. Edit **Page 1**.

2. Right-click the **Breadcrumbs** region name in the Rendering Tree and select **Delete** from the context menu, as shown in figure 5-43.
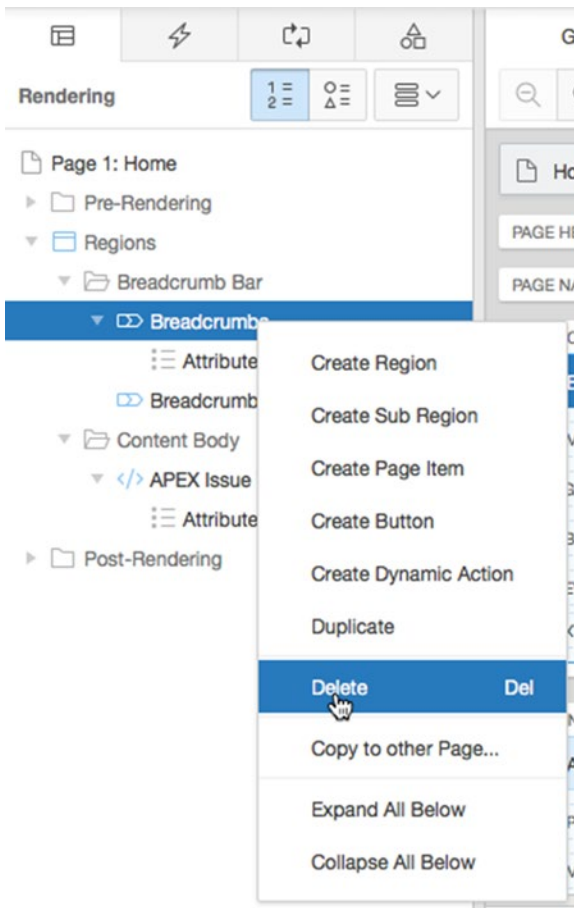
Figure 5-43. *Preparing to delete a redundant breadcrumb region from Page 1*

3. Click the **Save** button in the upper-right area of the Page Designer.

Now, re-test the application. You should just see the Global Page version of the breadcrumbs region, as shown in Figure 5-44.
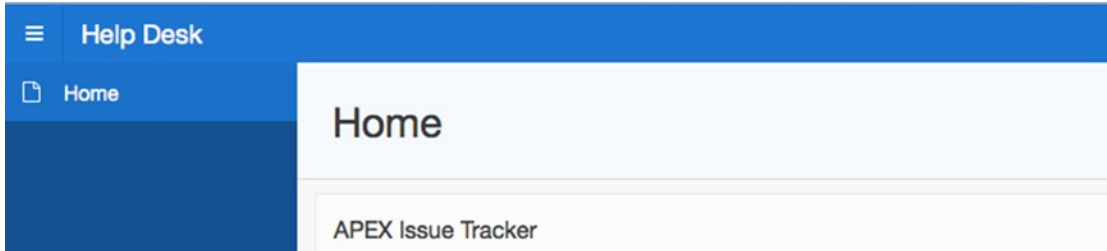
Figure 5-44. *Completed migration of the breadcrumb region to the Global Page*

Effectively, you have moved the management of the breadcrumb region to the Global Page. Any setting changes to that region done on the Global Page are seen on all pages of the application without requiring any additional work.

Breadcrumb Entries

As additional pages are added to the application, the page-creation wizard prompts for optional breadcrumb settings. If they weren't set at the time the page was created, or if they need to be modified from their existing settings, you can modify the data that drives the breadcrumbs in the Shared Components section of the application.

It's possible to have several breadcrumbs in one application. A default breadcrumb with the name Breadcrumb is created as part of the APEX Create Application wizard. This is the name of the grouping of breadcrumb entries. APEX provides some utilities to see where breadcrumbs are used as well as easy methods of editing entries.

To see the breadcrumb groups created, navigate to the Shared Components section and click the Breadcrumbs option in the Navigation section. Figure 5-45 shows the main screen for listing the different breadcrumb groups.
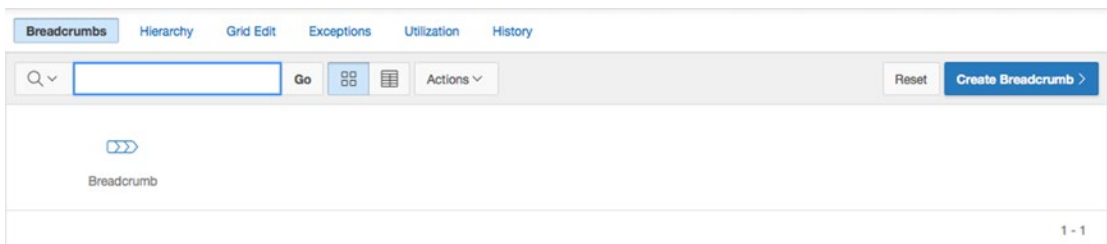
Figure 5-45. *Breadcrumb groups available in the application*

Clicking the group name displays the detailed entries in that group, as shown in Figure 5-46. The entries can be modified independently here. As an application becomes larger, you may need to arrange the entries into different breadcrumb groups.

Figure 5-46. *Detail of entries in a breadcrumb group*

Lists

As the name implies, a *list* is a structure that APEX uses to keep a collection of data for links. The list structure allows menus to be displayed consistently across numerous application pages, with easy maintenance performed in the Shared Components area of an application. Don't confuse navigation lists, which we are discussing here, with lists of values (LOVs). Lists are a navigational structure with built-in templates for displaying information in different ways. LOVs are used to support data entry, limiting the options a user can enter.

There are two types of lists: static and dynamic. Static lists are made up of list items that aren't data driven but are instead entered at design time by the developer. Dynamic lists are data based, and the values returned into the list are based on an SQL query.

List templates have a lot of capability. They support hierarchical lists, graphical bullets, dynamic HTML, and highlighting for the current page. Lists can contain data in a parent-child relationship; some list templates are specifically designed to display parent-child data. APEX themes contain a variety of templates for lists, but if the behavior you're looking for isn't already available, it's possible to modify or create your own list template to display and behave as desired.

As briefly mentioned earlier in this chapter, the new Universal Theme uses static lists instead of Tabs for navigation. A list named **Desktop Navigation List** is created to hold the navigation for the site, and a special List template built into the Universal Theme is used to display the list, depending upon some application-level attributes.

Whether using lists or tabs, as you navigate through the page-creation wizards, APEX will ask if you wish to create a navigation entry for the page you are creating. If you choose to create a new navigation entry while using the Universal Theme, a new list entry is created for you.

Although we could let the page-creation wizard create all of the list entries for us, you're going to create two entries in the list for pages that don't exist yet to show you how it can be done manually. Don't worry–you'll create those pages in the next few chapters. Here's the process to follow:

1. Navigate to the **<u>Shared Components</u>** section of your application.

2. Locate and click the **Lists** entry under **<u>Navigation</u>**.

3. Locate and click the **Desktop Navigation Menu** list in the resulting report.

4. To create a new list entry, click the **Create List Entry** button shown in Figure 5-47.
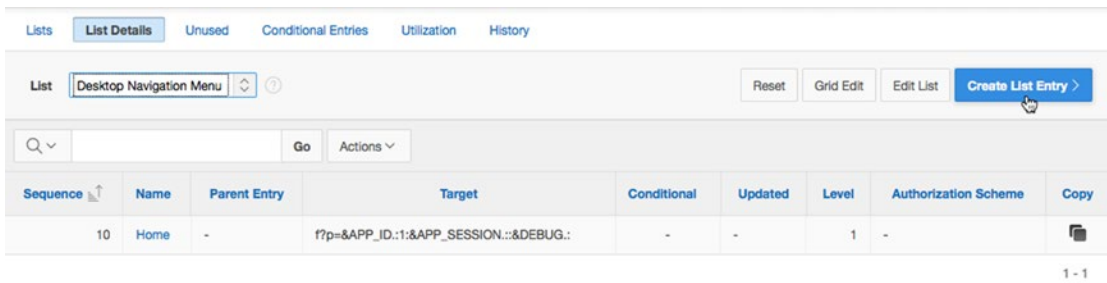
Figure 5-47. *Creating a new list entry*

The resulting page presents all the options available for a list entry. A lot of functionality is built into the lists structure. The key items you're interested in are shown in Table 5-1. Fill out the page shown in Figures 5-48 and 5-49 using the values from Table 5-1. Leave any other values at their defaults.

Table 5-1. *Values to Use for the First List Entry*

Section	Value	Entry
Entry	Parent	Home
	List Entry Label	Submit a Ticket
Target	Page	2
	Clear Cache	2
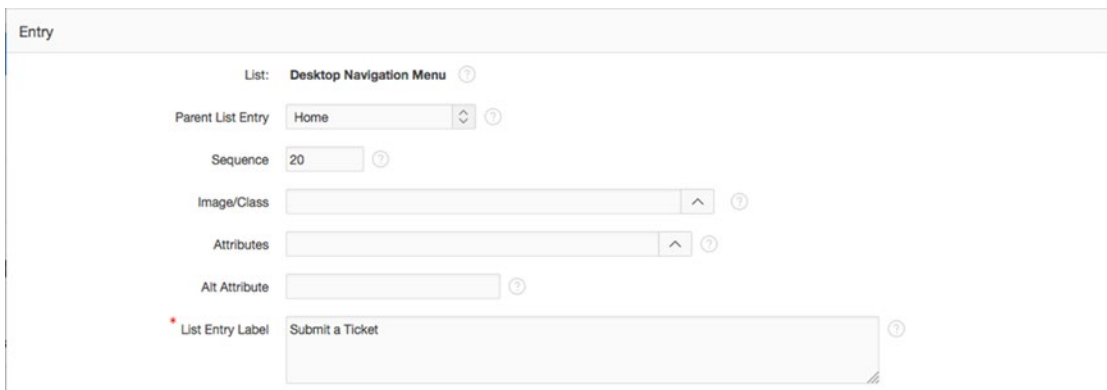
Figure 5-48. *Choosing a parent list entry and setting the label*

Figure 5-49. *Target definition*

5. Once you've finished your entries for the first list item, scroll to the top of the page and click the **Create and Create Another** button. This brings you back to the same page and allows you to add another list entry. Use the information in Table 5-2 to create the second list entry.

Table 5-2. *Values to Use in the Second List Entry*

Section	Value	Entry
Entry	Parent	Home
	List Entry Label	Contact Us
Target	Page	3
	Clear Cache	3

6. Click **Create List Entry** to save your changes.

You should now have a list with three entries in it, as shown in Figure 5-50. The List Details tab shows some important information in a single view. The Sequence value identifies the order in which the items are listed when using an unordered list type. Some list types are classified as ordered, in which case they're sorted by name alphabetically. The Target value is the construction of a URL that includes the page to navigate to as well as a clear-cache instruction. Several of the declarative forms construct a URL based on the inputs provided, in the same way as for the list entry.

Figure 5-50. *List entries at a glance*

A list, as a shared component (unless it is designated as the default navigation list for the UI), doesn't display in an application directly. Normally, a list region must be configured on a page in order for the list to be seen by the user. APEX has a template type defined specifically to support lists. The list templates contain all the intelligence required for dynamic lists and options for display. When you're creating a list region, the template choice can be set, and it can be modified through the region settings.

In our case, the list entries you created will be displayed as part of the navigation list region that is part of the Universal Theme.

Running your application now should result in the list entries you created appearing on the left side of the screen as part of the Navigation menu (Figure 5-51). Clicking either link generates an application error. This is expected: you've asked the application to link to pages that don't exist yet.
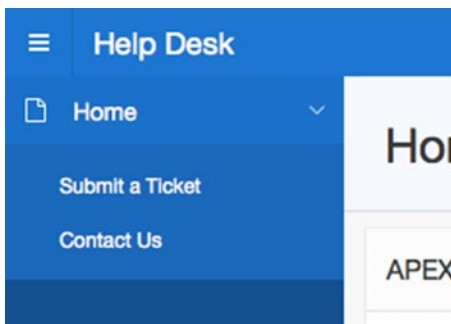
Figure 5-51. *Navigation menu with new list entries*

Lists of Values

One of the fundamental benefits of writing an application on top of a database architecture is the ability to enforce data quality. LOVs are an APEX component that can be mapped to different item types, including Select Lists, Multiple Select Lists, Checkboxes, and Radio Groups. These types of structures help ensure that data collected through transactions is consistent. As with lists, there are two types of LOVs in APEX:

- *Static*: A set list of options in APEX

- *Dynamic*: Based on data in the database returned via SQL

LOVs can be defined as shared components either at the application level or at the item level. Figure 5-52 shows an item-level attribute definition for a static LOV. An LOV used more than once should be written as a shared component. This allows the maintenance of that LOV to be centrally located with the Shared Components. If an LOV is created at the item level, it's easy to convert it to a shared LOV by using a utility that APEX provides. When viewing the LOV Shared Components, an LOV that is locally defined can be edited and converted to a Shared Component LOV.
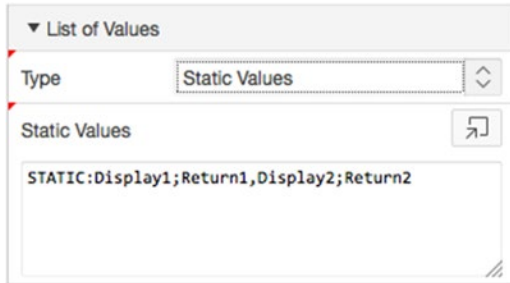
Figure 5-52. *An item-level LOV with static options*

Static List of Values

A static list of values is simply a set of display and return value pairs. This type of list is normally short and unchanging. When you define a static list of values at the item level, there are two types of data options:

- STATIC: Entries are automatically alphabetized.

- STATIC2: Entries render in the order in which they're entered.

The syntax for specifying a static LOV is as follows:

```
TYPE:DISPLAY;RETURN,DISPLAY;RETURN,...
```

The TYPE may be either STATIC or STATIC2.

If you wish the display value and the return value for a given entry to be the same, omit the semicolon and specify only one value. For example, the second item in the following example is a single value for both display and return:

```
TYPE: VALUE1,VALUE2,VALUE3,...
```

The return value in an LOV is saved as the value of the associated form item. In static lists, using the semicolon as the value of an entry may cause issues with parsing the list.

The following is an example of a static list. Commas separate the list items. Each list item is composed of a display value and a return value, with a semicolon separating those two values:

```
STATIC:C;1,A;2,D;3,B;4,
```

When you display the values in this list, you see only the display values. Because the list is type STATIC, the values are displayed in alphabetical order:

```
A
B
C
D
```

Next is an example of a STATIC2 list. Notice that the entries are specified in the same order as before:

```
STATIC2:C;1,A;2,D;3,B;4,
```

However, this time the values are displayed in their order of definition. They are *not* sorted alphabetically:

```
C
A
D
B
```

Shared-component static LOVs have more options than item-level static LOVs. Due to their shared nature, conditions and build options can be configured. These can be edited after the list has been created. Because the lists are stored differently as shared components, it's possible to use a semicolon in the item value.

Dynamic List of Values

As with static LOVs, dynamic LOVs have a display and return value pair requirement. The difference is that the values are obtained through an SQL query. The SQL query you write must return two columns. If the columns are the same, you need to use aliases to distinguish a display value and a return value. You must also use an alias if you're using a concatenated string as a column. Dynamic LOVs can also use session variables or values currently being used in the application. This gives dynamic LOVs flexibility to dynamically change what is offered during runtime.

The example application needs two LOVs to support the selection of user names. In preparation for building your form pages, create an LOV to support the names of the users and the technicians in your Help Desk system:

1. Navigate to the **Shared Components** section of the Help Desk application, then go to the **Other Components** section shown in Figure 5-53, and click the **Lists of Values** link.
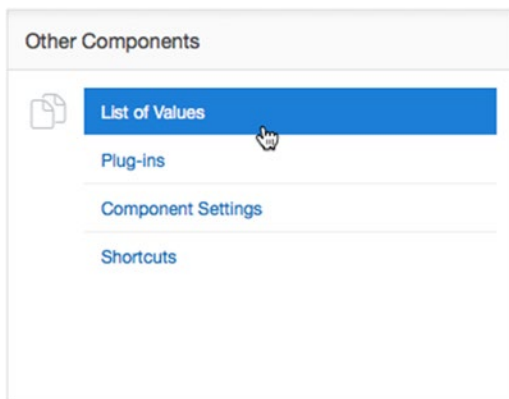
Figure 5-53. *Other Components options*

2. Click the **Create** button to create a new LOV.

3. Choose **From Scratch** as the method of creating your LOV, as shown in Figure 5-54.

Figure 5-54. *Creating an LOV from scratch*

4. Click Next.

5. Enter TECHS as the **Name** value and choose **Static** as the **Type**, as shown in Figure 5-55.

Figure 5-55. *Specifying a list as static*

6. Click **Next**.

7. Enter the values shown in Table 5-3 into the form. Add your own name to the list!

Table 5-3. Display Attributes for the LOV

Display Value	Return Value
Scott	SCOTT
Doug	DOUG
Karen	KAREN
Martin	MARTIN
Patrick	PATRICK
Tim	TIM
(Your Login Name)	(YOUR LOGIN NAME)

8. Click **Create List of Values** when you're finished.

Now that you've created a static LOV, let's include a second one that uses an SQL query to derive the list of values:

9. Repeat steps 1 through 4.

10. Create a second list named **USERS**, selecting the **Dynamic** option. Click **Next**.

11. Locate the book supplemental file ch5_lov.txt that includes the SQL query text. Enter the SQL query for the LOV source.

12. Click **Create List of Values**.

You should now have two LOVs. Don't worry if you made a mistake. All the settings can be modified—simply click the name of the LOV you want to modify.

Summary

In this chapter, you created the basic shell of an application and several of the supporting objects that you will use in the upcoming chapters. These items have been created as a result of planning that was done prior to starting to create the application. Depending on your situation, the amount of planning you do for your own application will vary. The shared components outlined here can be created at any time during the development process. In the next section, you will start using some of the key structures outlined here.

■ ■ ■ ■

Forms and Reports: The Basics

Now that you have the database objects and the base application in place, you can get to the real work of building pages in your application. Most applications contain a series of forms, reports, charts, and other elements designed to display, edit, and collect data.

This chapter focuses on basic forms and reports. These are the simplest, most standard types of forms and reports in APEX. They're most often created by using the APEX wizards, which create all the elements of a form or report for you.

In the sections that follow, you will learn how to use the APEX wizards to add pages to your Help Desk application. You will create some basic forms and reports on the Tickets table; you will also look at the elements created by the wizards for your working forms and reports.

APEX Forms

Forms are used to display, edit, and collect data, which is then sent back to the database for processing. Forms can interface with tables, views (via "instead of" triggers), procedures, and web services.

An APEX form is actually a collection of APEX objects acting together as a single, cohesive unit to perform insert, update, and delete operations on data elements. An APEX form generally consists of a region, one or more items, one or more buttons, and one or more processes that handle interactions with the database. The APEX form wizards create all the objects necessary for a fully operational form.

■ **Note** Once a form is generated, the objects in it aren't logically associated in any way except that they collectively make a complete working form. Although it's possible to alter or delete individual elements, doing so may cause the form to not work properly if an error is introduced; thus, doing so is not recommended.

The APEX form wizards listed in Figure 6-1 are the fastest, most effective, and most accurate way to create APEX forms. The wizards guide you through a series of steps, collect the information required for the form type, and then generate all the required items, processes, and buttons. Using the wizards frees you from the tedious and error-prone task of individually creating each component. After a wizard creates a form, you can, and likely will, make modifications and enhancements to the resulting components to tailor the form to your specific requirements.
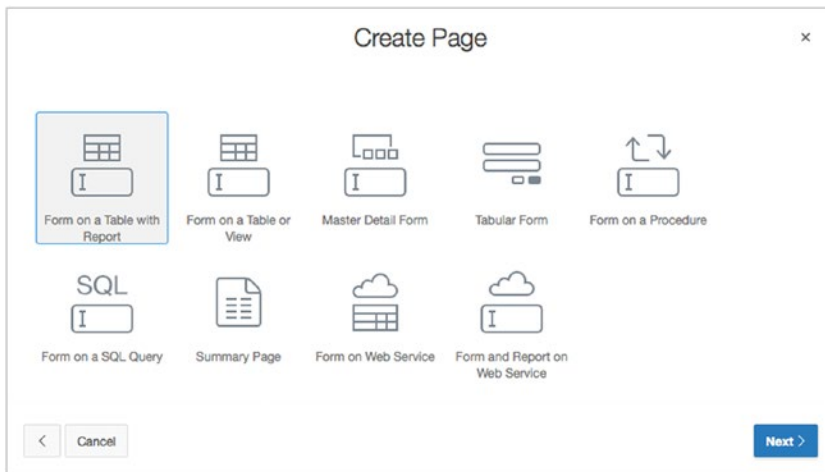
Figure 6-1. *APEX Create Page wizard showing form options*

The following are some of the form types that you can create using the wizards listed in Figure 6-1:

- *Form on a Table with Report*: A form built on the columns of a table or view, having one item for each table column and processing a single row of data at a time, plus a report on the contents of the table or view, with navigational elements between the report and form pages.

- *Form on a Table or View*: A form built on the columns of a table or view, having one item for each table column and processing a single row of data at a time.

- *Master Detail Form*: A form on a pair of tables having a master–detail relationship. The APEX Master Detail Form Wizard creates all the data, processing, and navigational elements required for managing master–detail data.

- *Tabular Form*: A multi-row, multi-column form (like a spreadsheet) that allows the editing of multiple rows and columns of data at once.

- *Form on a Procedure*: A form based on the parameters of a procedure, typically to collect values for passing in to a procedure for subsequent processing.

- *Form on a SQL Query*: A form built on the results of a SQL query. This is a very powerful form construct due to its flexibility.

- *Summary Page*: A display-only form showing selected items from an existing input form page. A summary page is often used in building a confirmation page for a wizard.

- *Form on Web Service*: A form on the arguments of a web service.

- *Form and Report on Web Service*: A single-row form on the arguments of a web service with a corresponding report of all rows of data, including navigational elements for moving from report to form and back.

If you look at the available APEX form wizards, you can see that several of them create accompanying reports (the Form on a Table with Report and Form and Report on Web Service Wizards). It's a common practice to use a report on a table, view, or web service to locate a particular row of data and then edit that data in a form on the same table, view, or web service. Some wizards simply create both the report and the form for you, including all navigation elements and database-transaction processes required to make everything work.

Form on a Table

One of the most common types of form in APEX is the form on a table. The APEX Form on a Table wizard automatically creates and maps APEX items to database columns, making it trivial to quickly create forms for database table entry and updates. As a developer, you can then modify the different types of controls for each column. All of the supported HTML widgets (text fields, text areas, select lists, radio groups, check boxes, and so on) are available, as well as several APEX-specific ones. The best way to understand just what the APEX Form on a Table wizard does is to use it, so let's dive in and create a form on a table.

Creating a Form on a Table

In this section you will create Page 2 of your Help Desk system and add a form to it. This form will allow the user to create a new ticket by inserting a row into the TICKETS table. You can limit which DML operations a form in APEX can perform. In this case, you restrict it to only performing inserts.

The Form on a Table wizard walks through all the steps required to generate a form on a table: selecting the parsing schema, selecting the table on which to base the form, selecting the columns to include and edit, assigning region and form titles, and specifying column headings. Begin as follows:

1. **Navigate** to your application's development home page. This is the page that lists all of the pages in your application.

2. Click the **Create Page** button in the upper right of the screen.

3. Select **Form**, and click **Next**.

4. Select **Form on a Table or View**, and click **Next**.

5. Set **Table/View Owner** to your schema, and select **TICKETS (table)** for **Table/View Name**, as shown in Figure 6-2. Click **Next**.

Figure 6-2. *Entering the schema and table name*

The next step allows you to set some details about the page and region that will be created as a result of the wizard. The Page Number can be set to anything you wish, but it must be unique within an application. The Page Name sets the text that appears in the browser tab when the application is run, and the Region Title sets the text that displays in the region's title area.

The Page Mode dictates whether the page you are creating will be a normal APEX page or one of the two types of dialogs now built in to APEX 5.0: modal or non-modal. Modal dialogs disallow interaction with the page underneath the dialog, while non-modal ones allow the user to see and interact with the underlying page.

The region template dictates how the region container is visually rendered. Each APEX theme has a number of templates available, but you'll find that you use the Standard template the most. Continue as follows:

6. Enter 2 for **Page Number**, as shown in Figure 6-3. Enter `Create a Ticket` for both **Page Name** and **Region Title**. Set the **Page Mode** to **Normal**. Set **Breadcrumb** to **Breadcrumb**. When the page refreshes, set **Parent Entry** to **Home** and click **Next**.

Figure 6-3. Specifying page, region, mode, and breadcrumb information

Next, you get to choose how this page relates to the menu system you've already defined, if it does at all. We've already created the entries in the Navigation List for these pages, so we'll use those as we create the page:

7.   For **Navigation Preferences** (Figure 6-4), select **Identify an existing navigation menu entry for this page**. When the page refreshes, set **Existing Navigation Menu Entry** to **Home**, and then click **Next**.

Figure 6-4. *Specifying navigation options*

APEX 4 introduced the ability to use ROWID as a primary key. This comes in handy when you're dealing with a table that has a multi-column natural primary key, but the table already has a single-column primary key defined, so you'll use that:

8.   Set **Primary Key Type** to **Select Primary Key Column(s)**, ensure that **Primary Key** is set to TICKET_ID, and click **Next**.

The primary key of the table is based on a sequence within the database, and there is already a trigger in place that fills the primary key with the next sequence value, if the primary key for the incoming record is null, as follows:

9. Set **Source Type** to **Existing Trigger**, as shown in Figure 6-5, and click **Next**.
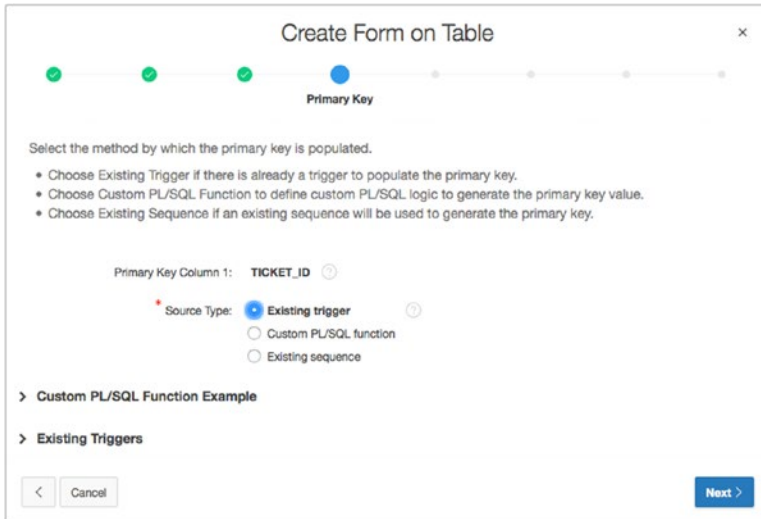
Figure 6-5. *Specifying the primary key population option*

Next, specify the columns that will be visible and editable on the form. By default, all the columns in the chosen table appear in the selected column. However, for this simple form, you want to restrict the columns the user can see:

10. Using the shuttle, make sure SUBJECT, DESCR, CREATED_BY, and STATUS_ID are the only columns selected, as shown in Figure 6-6, and click **Next**.
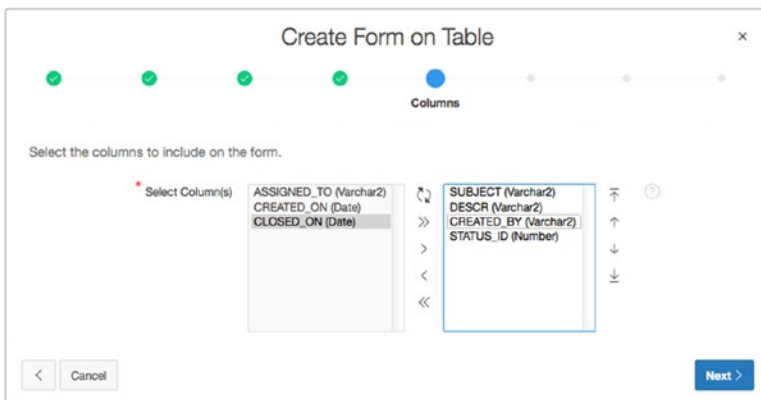
Figure 6-6. *Selecting the columns to include*

Not all forms allow people to update or delete data. Some are simply data-entry forms. In this case, you want un-authenticated users to be able to submit a ticket, but you don't want them to be able to edit or delete those tickets. The next step of the wizard allows the developer to choose which actions are available to the end user and to name the buttons related to those actions.

Every form should have a Cancel button that allows the user to abort any actions or data entry. But the rest of the buttons are optional:

- *Create button*: Saves a new record

- *Save button*: Saves updates to an existing record

- *Delete button*: Deletes an existing record

Continue now with creating the form:

11. Enter Cancel for **Cancel Button Label** and Create a Ticket for **Create Button Label**. Set **Show Save Button** and **Show Delete Button** to **No**, as shown in Figure 6-7, and click **Next**.

Figure 6-7. *Specifying the buttons to display*

When the user enters a ticket and clicks a button to either cancel data entry or create the new ticket, you need to specify what happens next. Does APEX stay on the same page? Does it return to the home page?

In this instance, you want the user to be redirected to the home page no matter which choice they make:

12. Set both **<u>Branch here on Submit</u>** and **<u>Branch here on Cancel</u>** to **1**, and click **Next**. See Figure 6-8.
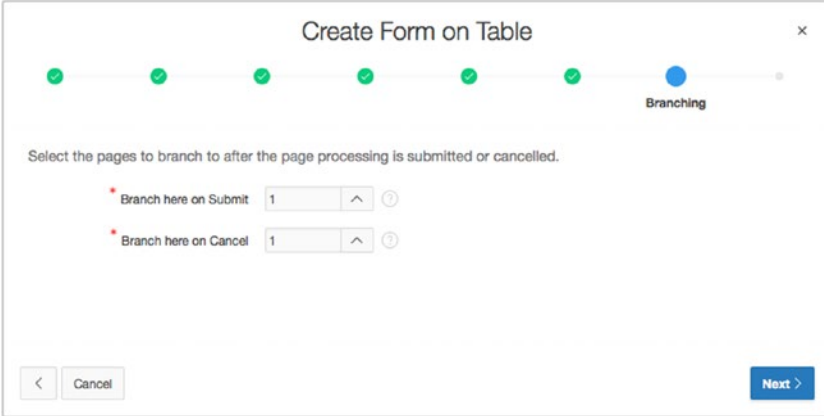
Figure 6-8. *Specifying branching for Submit and Cancel*

As with most wizards, you're presented with a final page that summarizes your choices. At this point you can use the Previous and Next buttons to work your way back and forth through the wizard steps to alter any of your choices. Then do the following:

13. Click **Create** to complete the wizard.

14. **Run** your application.

Congratulations! You've just created a fully operational form on the TICKETS table. The form should look similar to that in Figure 6-9.
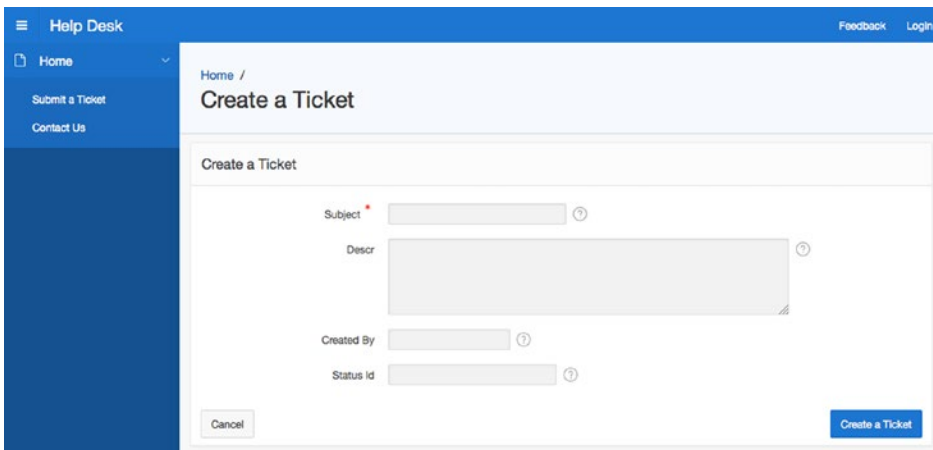
Figure 6-9. *Running the form on the TICKETS table*

Notice that the form region is labeled as you specified in step 6, the form contains fields for the four columns you selected in step 10, and the Create a Ticket button is labeled as you specified in step 11. Also notice that the four fields are each created as the default element type specified in the UI defaults for the TICKETS table that you created in Chapter 4. The help text you specified for each column is there, and it pops up in a new window when you click the question mark icon at the end of the field. The Cancel button brings you to the home page—page 1, as you specified in step 12. APEX did a lot of work for you!

Modifying a Form on a Table

The APEX wizards handle most of the work of creating a form for you. However, it's rare that you won't have to make some minor changes to what the wizard creates. Now that you have the Create a Tickets form on page 2 of your application, you can make a few changes to polish it up a bit.

Changing the Label Templates

You'll change the label templates for P2_SUBJECT and P2_CREATED_BY (the items that correspond to the SUBJECT and CREATED_BY table columns) to Required with Help. Use of the Required with Help label template indicates to the end user that this is a required field on the form. However, it doesn't make the field itself mandatory. You will do that later.

You'll also reduce the width of P2_CREATED_BY so it doesn't take up as much space. Begin as follows:

1. Edit **Page 2** of the application.

2. Edit the item **P2_SUBJECT** by clicking its name in the **Rendering** tab of the **Tree Pane**.

3. In the **Property Editor**, scroll to the **Appearance** attribute group, as shown in Figure 6-10, and set **Template** to **Required**, and click **Save**.

Figure 6-10. *Modifying the label templates*

4.   Edit the item **P2_CREATED_BY** by clicking its name in the **Rendering** tab of the **Tree Pane**.

5.   In the **Property Editor**, scroll to the **Appearance** attribute group, as shown in Figure 6-11. Set **Template** to **Required**, and the **Width** to 20, then click **Save**.
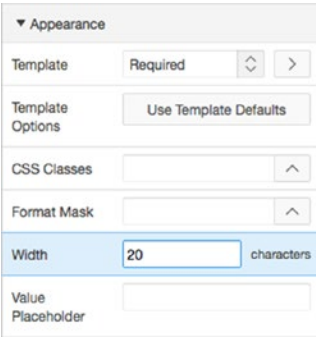
Figure 6-11.   *Setting the display attributes*

Next, you want to hide the P2_STATUS_ID item from the user, because you don't want the user to change this value. You do, however, want all new tickets to be created with a default value of OPEN. Because you can't guarantee which STATUS_ID maps to which STATUS, you can call a simple function and pass in the STATUS. This function, in turn, returns the corresponding STATUS_ID, which is set as the default value for P2_STATUS_ID:

1.   Edit the item **P2_STATUS_ID** by clicking its name in the **Rendering** tab of the **Tree Pane**.

2.   In the **Identification** attribute group, set **Type** to **Hidden**.

3.   In the **Default** attribute group shown in Figure 6-12, set **Type** to **PL/SQL Function Body** and set the default value to RETURN get_status('OPEN');. This function was created as part of the script run in Chapter 4.

4.   Click **Save**.

Figure 6-12.   *Specifying a default value*

Next, you want to set page 2 to be a public page. You want any user—authenticated or not—to be able to access this page:

1. Edit the page attributes for **Page 2** of your application by clicking its name (Page 2: Create a Ticket) at the top of the **Rendering** tab of the **Tree Pane**.

2. Set **Page 2** to be a public page, and click **Apply Changes**. Refer back to Chapter 5 for detailed steps.

Finally, you need to make sure users enter values for the Subject and Created By fields. There are two ways to make a field mandatory in APEX. For demonstration purposes you'll use a different method for each field.

Making the Fields Mandatory

For the Subject field, you'll create a validation. Although a validation takes more steps, it gives you more control over how and when it's performed. Here's what to do, first for the Subject field and then for the Created By field:

1. Edit **Page 2** of the application.

2. Create a new validation by switching to the **Processing** tab of the **Tree Pane** then right-clicking the **Validating** node in the tree and selecting **Create Validation**, as shown in Figure 6-13.
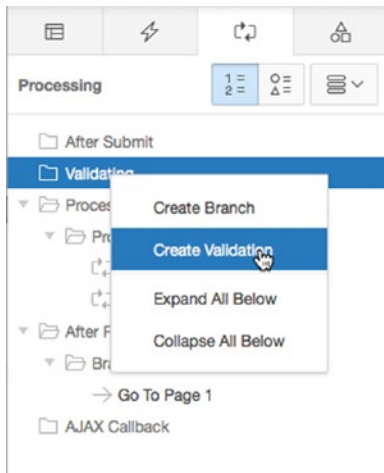
Figure 6-13. *Choosing to create a new validation*

You will see a new node in the validation tree that is highlighted and has a red X next to the name. This indicates that the validation has been created, but that there are attributes that must be filled in. Looking at the Property Editor, you'll see a number of attributes highlighted in red, as seen in Figure 6-14.
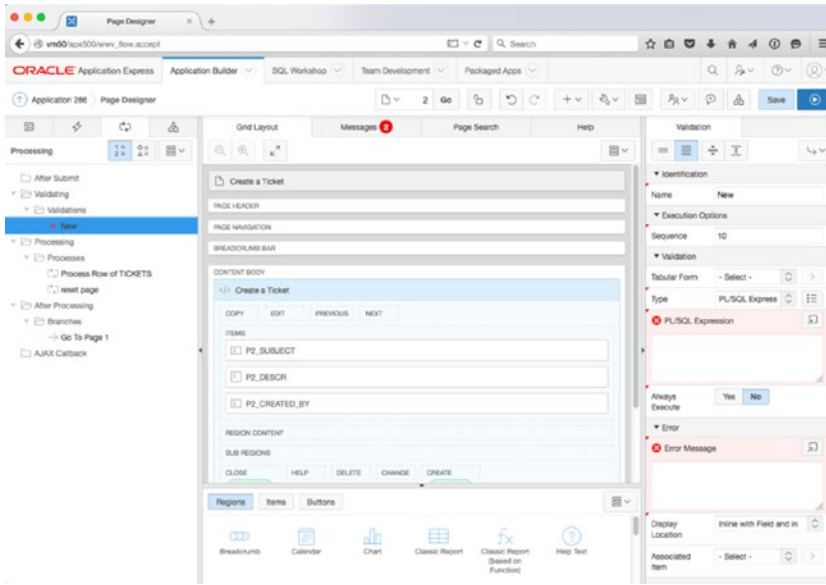
Figure 6-14. *A new validation that needs to be completed*

We'll now fill out the required attributes for our validation, as seen in Figure 6-15:

3. In the **Identification** attribute group, set the name to **P2_SUBJECT is NOT NULL**.

4. In the **Validation** attribute group, set the **Type** to **Item is NOT NULL**, then using the pop-up select list, set the **Item** to **P2_SUBJECT**.

5. In the **Error** attribute group, set the **Error Message** to #LABEL# must have some value and set the **Associated Item** to **P2_SUBJECT**.

6. Click **Save**.

Figure 6-15. *Entering the details for a new validation*

Next, use the second method to make the Created By field mandatory. To do this, simply set an attribute of the input item:

7. Switch to the **Rendering** tab of the **Tree Pane** and edit **P2_CREATED_BY**.

8. In the **Property Editor**, navigate to the **Validation** attribute group shown in Figure 6-16, set **Value Required** to **Yes**, and click **Save**.

Figure 6-16. *Making a value required*

If you check the Processing tab of the tree pane, you'll see that no new validation has been created. That is because you used the item-level attribute instead of creating a full validation. The main difference between an item-level and a full validation is that with the item-level validation, you can't conditionally control when the attribute is applied, and you don't have direct control over the error message that is displayed.

Go ahead and run the application again. At this point, you should be able to enter new tickets into the system but not see them anywhere outside of SQL Workshop.

Looking Behind the Scenes

Now that you have a working form, let's look at just what the APEX Form wizard built in order to understand a bit more about how your form works. If you have installed the Web Developer Toolbar add-in (available for both Chrome and Firefox), you can use the Form ä Display Form Details option to display the form details. Figure 6-17 illustrates the Create a Ticket form with the form details exposed.

Figure 6-17. *Form on the TICKETS table with form details exposed*

■ **Note** The Web Developer Toolbar add-in is a free web-development tool, written by Chris Pederick, that lets you inspect various aspects of a web page. To learn more about Web Developer, visit http://chrispederick.com/.

The highlighted input tags display the input identifier and name for each field of the form. Both are unique for each form field. The input identifier is the column name prepended with the page number. The input name identifies the element names that APEX uses internally to process data in the form. Note that the columns you didn't choose to display in the form, TICKET_ID and STATUS_ID, are still present in the page's HTML.

A look behind the scenes tells you more. Edit Page 2 to view the elements that make up the new form.

Figure 6-18 shows three of the tabs from the tree pane: Rendering, Processing and Shared Components. The Rendering tab contains APEX objects required for page rendering. The Processing tab contains objects required for page processing, such as validations, processes, and branches. The Shared Components tab contains APEX objects that are shared across pages, such as tabs, lists of values, breadcrumbs, templates, and security schemes.
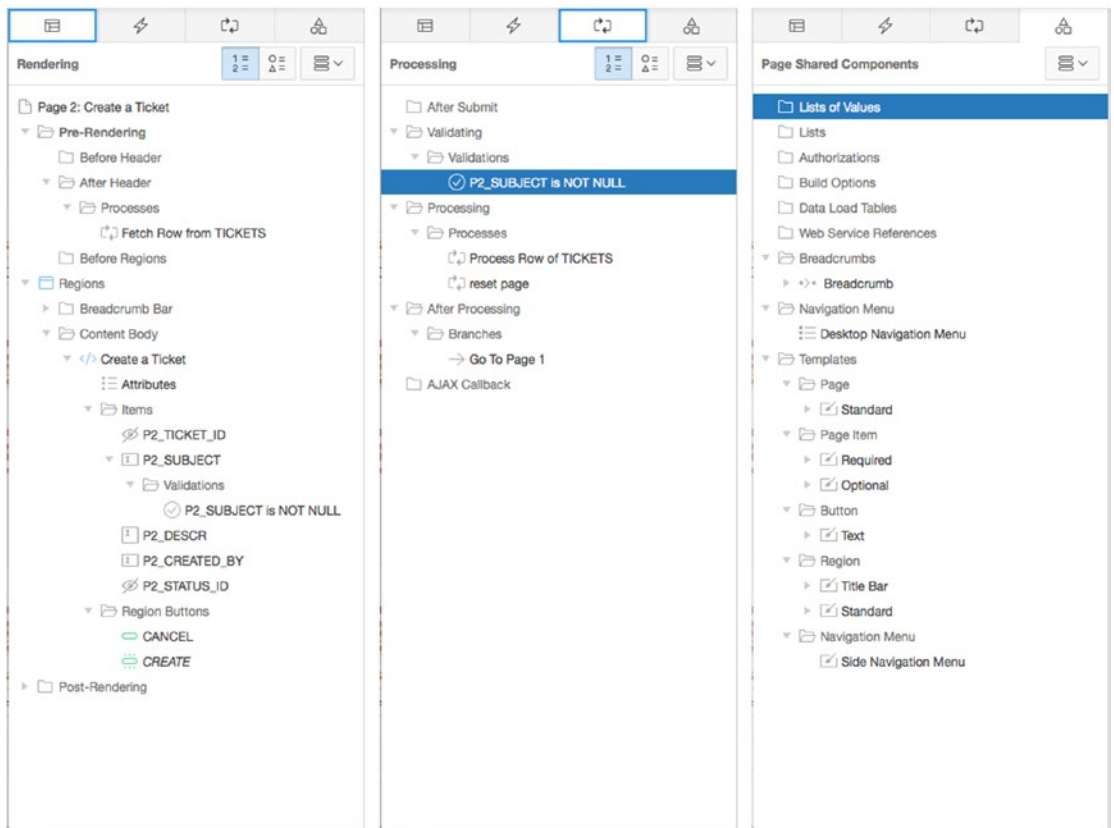
Figure 6-18. *Elements of a form as viewed from the Page Builder's various tree panes*

For your new Create a Ticket form, in the Rendering tab, you see that the wizard has created one item for each of the columns from the TICKETS table that you selected via the wizard. There are also two buttons called Cancel and Create, and a Fetch Row from TICKETS process. This process is an Automated Row Fetch process, which does exactly what its name says: it fetches a row from the designated table into the current form. The attributes of the Automated Row Fetch process specify the table owner, the table name, the primary key column(s), success and failure messages, and a condition.

Notice that the TICKET_ID item is present in the tree but isn't rendered on the form, nor is it present in the Grid Layout. It's visible in the Display Details view (Figure 6-17) of the form as the first element on the page, with no visible element associated with it. TICKET_ID is a hidden item. APEX hidden items exist to hold a value, but although they're rendered on the page, they aren't visible to the user. In this case, the hidden TICKET_ID column holds the primary key value for the TICKETS row. As the primary key, TICKET_ID is used by the APEX processes to pull data from the database and to process inserts, updates, and deletes on a TICKETS row. Because you don't want the end users to edit the primary key, APEX automatically hides it for you.

In the Processing tab, you have a Process Row of TICKETS process, a Reset Page process, and a Go To Page 1 branch. The Process Row of TICKETS process does just that: it processes one row of the TICKETS table using the values from the items that correspond to the columns of the TICKETS table. This process fires when the user clicks the Create button. The Reset Page process clears the items on the page. It fires when the user clicks the Cancel button.

In the Shared Components tab, you need to expand the Navigation Menu tree node to see that this page uses the Desktop Navigation Menu. Expanding the Breadcrumbs region shows the Breadcrumb object. Under Templates, you see that your form uses the Standard page template, the Title Bar and Standard templates, two different Page Item templates, and the default Button template.

All APEX form wizards create items, buttons, and processes, but in different combinations to suit the specific needs of the form type. The other APEX form wizards perform essentially the same way, with slight differences in process types and navigation objects so as to accommodate the underlying data source: table or view, procedure, query, or web service. Next, let's look at a form on a procedure.

Form on a Procedure

Another way to create a form in APEX is to create it based on the parameters of a PL/SQL procedure. Instead of the traditional DML processes, APEX calls the associated procedure and executes whatever logic is embedded within it. This method is also referred to as *using table APIs*, because this is the option to use if all access to tables in your workspace schema must be done through a table API.

Creating a Form on a Procedure

The process to create a form on a procedure is almost identical to that of a form on a table. You create a new page containing a form on the CONTACT_US stored procedure that was created as part of the exercises in Chapter 4, which enables users to contact you through the Help Desk application:

1. **Navigate** to your applications development home page.

2. Click the **Create Page** button in the upper right of the screen.

3. Select **Form** and click **Next**.

4. Select **Form on a Procedure** and click **Next**.

5. Set **Procedure Owner** to your schema, enter CONTACT_US for **Stored Procedure Name**, as shown in Figure 6-19, and click **Next**.

Figure 6-19. *Creating a form on a stored procedure*

6. In the top section of the page, enter 3 for **Page Number**, enter Contact Us for both **Page Name** and **Region Name**, and set **Breadcrumb** to **Breadcrumb**. When the region refreshes, select **Home (Page 1)** to set it as the **Parent Entry**, as shown in Figure 6-20, and click **Next**.

Figure 6-20. *Selecting the breadcrumb parent entry*

7. For **Navigation Preferences**, select **Identify an existing navigation menu entry for this page**. When the page refreshes, set **Existing Navigation Menu Entry** to **Home**, then click **Next**.

8. Leave **Invoking Page** and **Button Label** blank, and click **Next**.

9. Enter 1 for both **Branch here on Submit** and **Branch here on Cancel**, as shown in Figure 6-21. Then click **Next**.
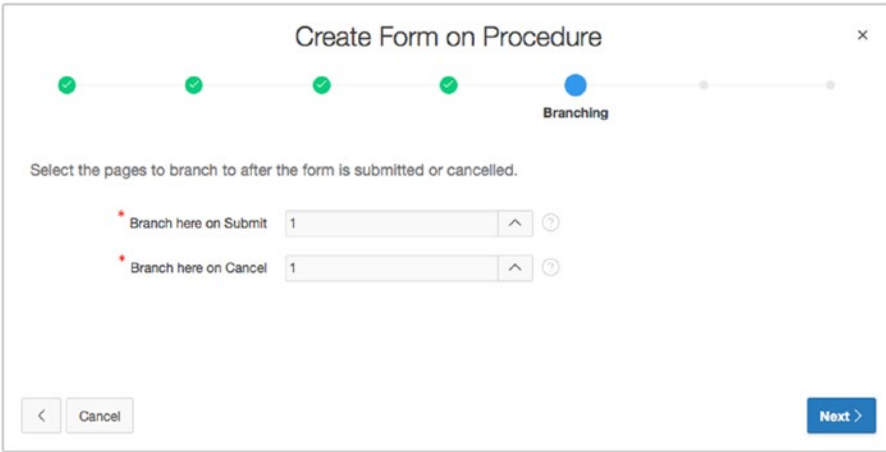
Figure 6-21. *Specifying branching options*

10. In the dialog in Figure 6-22, set the **Label** for P_FROM to **From**. Set the **Label** for **P_BODY** to Body. Set the **Display Type** for **P_BODY** to **Textarea**, and then click **Next.**
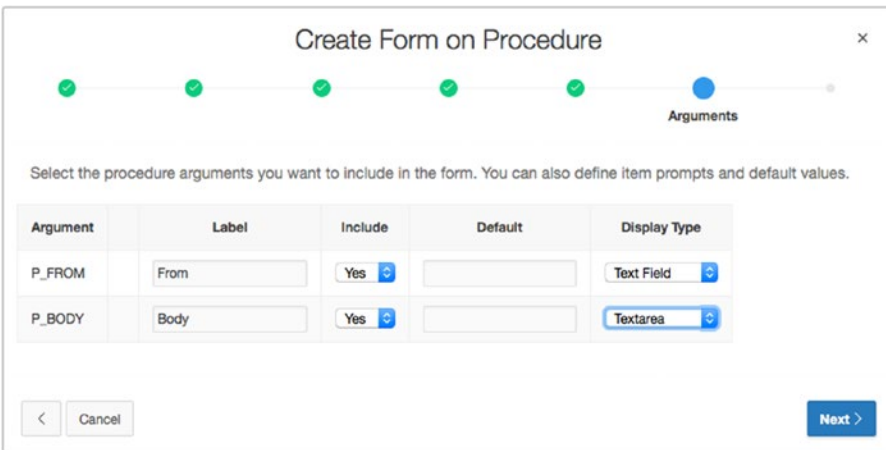
Figure 6-22. *Specifying procedure arguments*

11. Click **Create**.

Modifying a Form on a Procedure

Once again, the wizard has done most of the work, but you have a few minor changes to make before your form on a procedure is complete. You want both the From and Message values to be required, so you need to change their label templates and set their Value Required attribute to Yes. This time we're going to use a new feature in APEX 5.0 that allows us to edit the properties of multiple components at once. Do the following:

1. Edit **Page 3** of the application.

2. Select **P3_FROM** by clicking its name.

3. Holding the CTRL key (or Command key on Mac) select **P3_BODY** by clicking its name. At this point both elements should be selected.

Now, if you look at the Properties Editor, you'll see that there are several options that have a blue background and a grey Delta (Δ) symbol just to the left of the label. This indicates that the components you have selected have different values for these options. It's simply a visual clue so that you know that the fields may not in fact be blank, but that APEX can't display the varying values between the components.

We'll continue by changing the common attributes as follows:

4. In the **Appearance** attribute group, change **Template** to **Required**.

5. In the **Validation** attribute group, change **Value Required** to **Yes**.

6. We now want to change a few attributes only for P3_BODY. So we'll have to de-select everything else so we don't accidentally change their attributes as well. Remember, as long as you haven't clicked the Save button, you can always use the Undo button to step back through what you've done. To select P3_BODY and de-select everything else:

7. In the **Rendering** tab of the **Tree Pane**, click on **P3_BODY**.

8. In the **Appearance** attribute group, set **Width** to 80 and **Height** to 5.

Next, set page 3 to be a public page. You want any user—authenticated or otherwise—to be able to send you a message through the Contact Us page:

9. Set **Page 3** to be a public page. Refer back to Chapter 5 for detailed steps.

Finally, modify the process that was created to include a success message:

10. Switch to the **Processing** tab of the **Tree Pane**.

11. Edit the process **Run Stored Procedure** by clicking its name.

12. In the **Success Messages** attribute group, enter the following for the **Success Message**:

    Your message has been sent.

13. Scroll to the top and click **Save**.

Run your application and test the Contact Us form. Each time you submit a record, an email is sent to info@example.com. If you want to change the destination address for the email, you can use the SQL Workshop's Object Browser to edit the CONTACT_US procedure.

Looking Behind the Scenes

From the user perspective, there is no indication that the form you've just created was created on a procedure. Looking in the Page Builder, the objects in the Page Rendering sections are similar to what you saw in your form on a table on page 2, but not exactly. Let's take a look to see what makes your form on a procedure different from the form on a table. Edit page 3 of your application. The different tabs of the tree pane are represented in Figure 6-23.
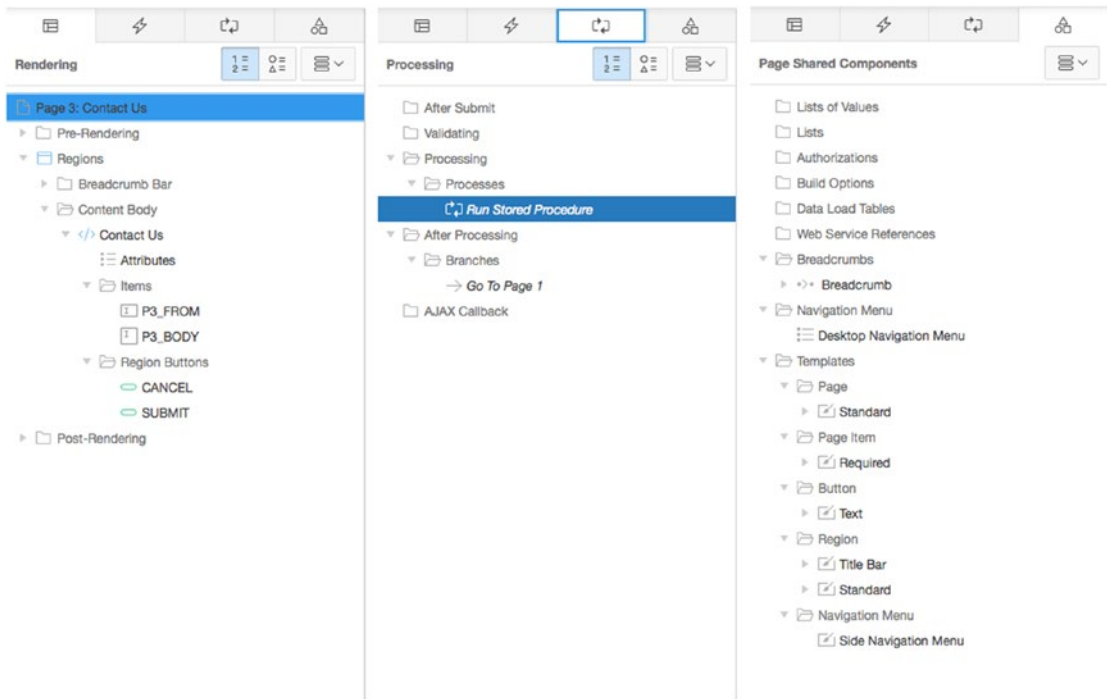
Figure 6-23. *Elements of a form on a procedure as viewed from the Page Builder's various tree panes*

In the Rendering tab, you have two items, P3_FROM and P3_BODY, corresponding to your two form fields, From and Body. There are two buttons, CANCEL and SUBMIT.

In the Processing tab are a process and a branch. However, the process is a different type—a PL/SQL anonymous block. This powerful type of process executes the PL/SQL procedure specified in the Source element. The PL/SQL procedure can be a stored PL/SQL procedure or an anonymous PL/SQL block, as long as the code is syntactically correct between a BEGIN statement and an END statement. In this case, the process calls the CONTACT_US procedure using the P3_FROM and P3_BODY item values as input parameters. The body of the CONTACT_US procedure is what creates and sends an email. Thus, the key difference between the form on a table and the form on a procedure is in the page-processing process that is executed on a click of the Create button. The APEX wizard has automatically provided the process type required for the selected form type.

The Shared Components region contains the standard entries for the table, breadcrumb and page, tab, region, label, and button templates, the same as for the form on a table. Again, it was nice of the form wizard to create all these elements for you.

Master–Detail Report and Form

One of the most popular features in APEX is the Master Detail Form wizard. With a single, simple wizard, you can quickly create a report and corresponding forms to manage data stored in a master–detail fashion. Let's use this wizard to create a report and forms for the TICKETS and TICKET_DETAILS tables.

Creating a Master–Detail Report and Form

First, you create the report and form on application pages 200, 210, and 220. Because you don't yet have those pages created, the wizard does that for you.

1. Navigate to the Application Builder home page for your application.

2. Click the **Create Page** button in the upper right of the screen.

3. Select **Form** and click **Next**.

4. Select **Master Detail Form** and click **Next**.

5. See Figure 6-24. Set **Table/View Owner** to your schema. Set **Table/View Name** to **TICKETS (table)**. When the page refreshes, all the columns from the table are selected by default. Click **Next**.

Figure 6-24. Creating the master page

When dealing with a master–detail relationship, you normally have a foreign key between the detail and master tables. However, that may not always be the case. At the detail table step, the wizard allows you to choose whether to show only tables that are related via a foreign key.

In this case, the tables are indeed linked, so you can leave Show Only Related Tables set to Yes.

6. Select **TICKET_DETAILS** for **Table/View Name**. When the page refreshes, make sure the following columns are moved to the Selected area to the right. You should end up with results like those in Figure 6-25.

    - TICKET_DETAILS_ID
    - TICKET_ID
    - DETAILS
    - CREATED_BY
    - CREATED_ON
    - ATTACHMENT
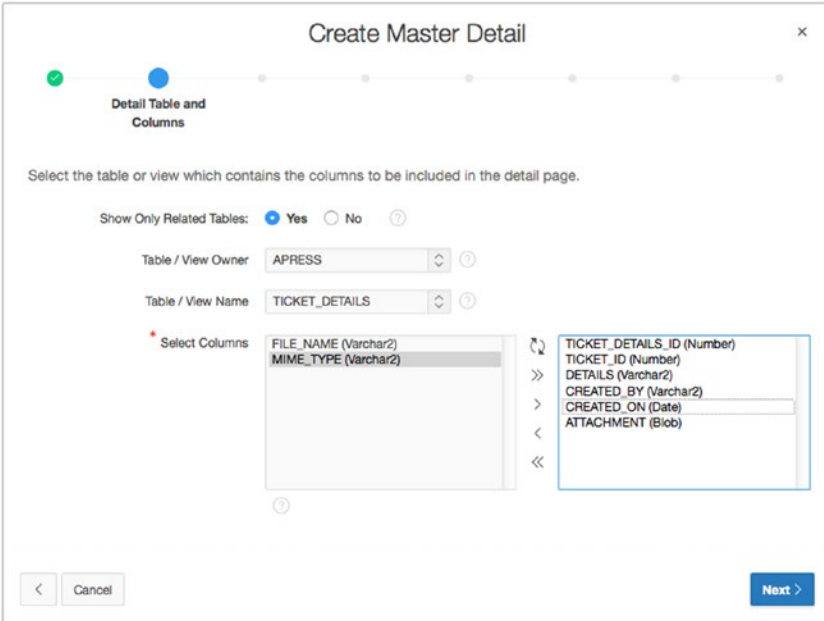
Figure 6-25. *Defining the detail table*

7. Click **Next**.

8. Set <u>**Primary Key Type**</u> for the master table to **Select Primary Key Column(s)**. For <u>**Primary Key Column 1**</u>, select **TICKET_ID (Number)**.

9. Set <u>**Primary Key Type**</u> for the detail table to **Select Primary Key Column(s)**. For <u>**Primary Key Column 1**</u>, select **TICKET_DETAILS_ID (Number)**.

10. Click **Next**.

11. Set <u>**Primary Key Source**</u> to **Existing Trigger** for the master table, and click **Next**.

12. Set <u>**Primary Key Source**</u> to **Existing Trigger** for the detail table, and click **Next**.

13. Set <u>**Include master row navigation?**</u> to **Yes**, as shown in Figure 6-26. Set <u>**Master Row Navigation Order**</u> to **CREATED_ON**, and click **Next**.

    Do not click Finish at this point.

Figure 6-26. *Defining master-row navigation options*

14. Set **<u>Build Master Detail with</u>** to **Edit Detail on Separate Page**, and click **Next**.

15. On the next page, set the items to the values shown in Figure 6-27.

Figure 6-27. *Specifying page attributes*

16. Set **<u>Breadcrumb</u>** to **Breadcrumb**.

17. Once the region refreshes, in the **<u>Create Breadcrumb Entry</u>** section, set the items to the values shown in Figure 6-28.

Figure 6-28. *Creating a breadcrumb entry*

18. Click **Next**.

19. Set the **<u>Navigation Preference</u>** in Figure 6-29 to **Create a new navigation menu entry.** When the page refreshes, enter Tickets for **<u>New Navigation Menu Entry</u>** and leave the **<u>Parent Navigation Menu Entry</u>** set to - No parent selected - then click **Next**.

Figure 6-29. Setting navigation options

20. Confirm your selections, and click **Create**.

When the wizard completes, you have a working master–detail form on the TICKETS and TICKET_ DETAILS tables, plus a report on the TICKETS table. This is perhaps one report more than you expected, but APEX knows that in most cases, you need the report to select the master–detail record to be edited, so that report is created at the same time for convenience. The Master Detail Form wizard created one report and two forms, plus the links and branches for navigation and the processes for performing database transactions. The Tickets report has a link to the Tickets form, which allows editing of ticket master data and lists ticket details. The Ticket Details region on the Manage Tickets page has an Edit link to the Ticket Detail modal dialog, where the user can add, update, or delete ticket detail information. All the items, buttons, processes, and even the column links were created by the Master Detail Form wizard.

Again, although you can build a master–detail form and report manually, the wizard is much faster and certainly more efficient. Now, let's make some adjustments to the report and the forms to suit your requirements.

Modifying a Master-Detail Report

Next, let's modify the report to add CSV export capabilities, change the sorting options, and modify the date format mask. Then we'll clean up the two edit forms. Here are the steps:

1. Edit **Page 200** of your application.

2. In the **Rendering** tab of the **Tree Pane**, highlight all columns of the **Tickets** report except for TICKET_ID and DESCR, as shown in Figure 6-30.
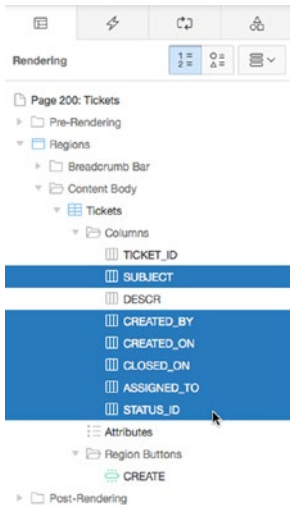
Figure 6-30. *Choosing to edit report attributes*

3. In the **Properties Editor**, navigate to the **Sorting** attribute group and set **Sortable** to **Yes**.

4. Select the DESCR column of the report and in the **Properties Editor**, change the **Type** to **Hidden Column**. See Figure 6-31.
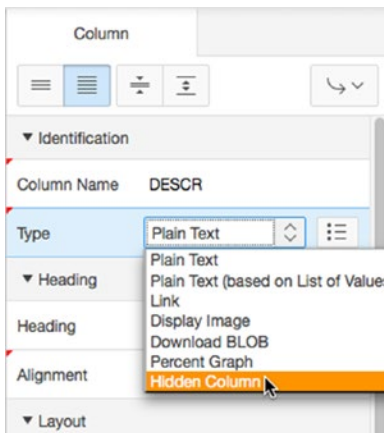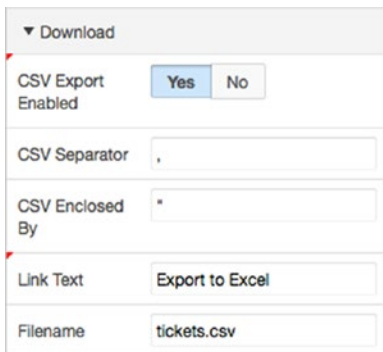
Figure 6-31. *Editing column attributes*

132

5. In the **Rendering** tab of the **Tree Pane**, click the **Tickets** report's **Attributes** child node.

6. In the **Properties Editor**, navigate to the **Pagination** attribute group and set **Partial Page Refresh** to **Yes**.

7. In the **Download** attribute group, set **CSV Export Enabled** to **Yes**. Once the section refreshes, set the following options, which you can also see in Figure 6-32:

    - Separator: **,**

    - Enclosed By: **"**

    - Link Text: `Export to Excel`

    - Filename: `tickets.csv`

Figure 6-32. *Setting report export options*

8. In the **Rendering** tab of the **Tree Pane**, edit the `CREATED_ON` column by clicking on its name.

9. In the **Appearance** attributes group, use the pop-up list of values to select **Monday, 12 January, 2004** as the **Format Mask**. Selecting it returns **fmDay, fmDD fmMonth, YYYY** into the **Number/Date Format** field, as shown in Figure 6-33.

Figure 6-33. *Selecting a date format mask*

CSS and HTML formatting directives entered in the Column Formatting properties group are applied to the report column when the page is rendered:

10. In the **Column Formatting** properties group, enter `font-weight:bold` for the **CSS Style** field (Figure 6-34).
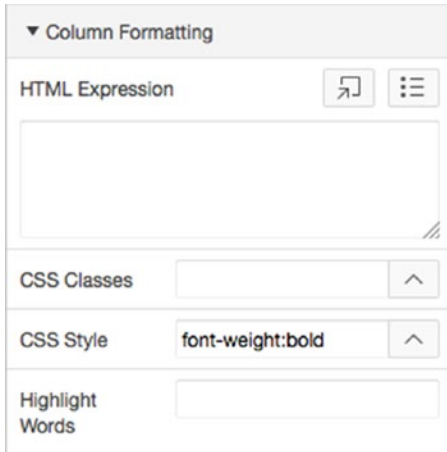
Figure 6-34. *Choosing column formatting options*

11. Edit the `TICKET_ID` column by clicking its name.

12. In the **Export/Printing** attributes group, set **Include in Export / Print** to **No** and click **Save**.

13. Run the page to view your changes.

Note that when you sort an APEX report column by date, the report sorts based on the value of the actual date, not the displayed value. This is a built-in feature of APEX. Also, when you export to Excel, the `TICKET_ID` column isn't part of the resulting CSV file, which is the result of your setting the Include in Export option to No.

Next, remove `STATUS_ID` and replace it with the corresponding value, pulled into the report by a slight adjustment to your query:

1. Edit **Page 200** in your application.

2. Edit the **Tickets** report by clicking the region's name in the **Rendering** tree.

3. In the **Source** attributes group, click the **Code Editor** button in the upper right, near the SQL Query definition. This will expand an editor window, allowing you to better edit the SQL statement.

4. Locate and open the file `ch6_add_status_to_report.txt`, which you can find where you extracted the class files earlier, and copy the contents into Code Editor, replacing all text that is currently there, and click OK to dismiss Code Editor. See Figure 6-35.
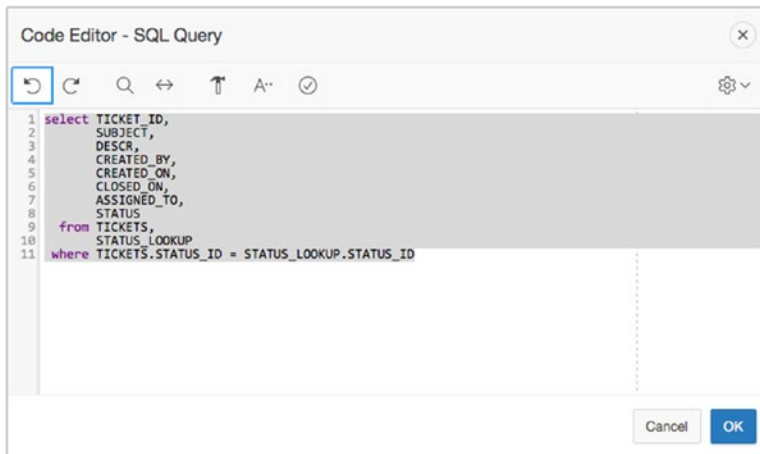
Figure 6-35. *Pasting the new query text into Code Editor*

Now, we'll reorder the columns in the report by using simple drag and drop. In this case, we want to move the STATUS column just between the TICKET_ID and SUBJECT columns. You can do this as follows:

> **5.** In the **Rendering** tree, click and drag the STATUS column from the bottom of the list and drop when the indicator shows its position to be between the TICKET_ID and SUBJECT, as shown in Figure 6-36.
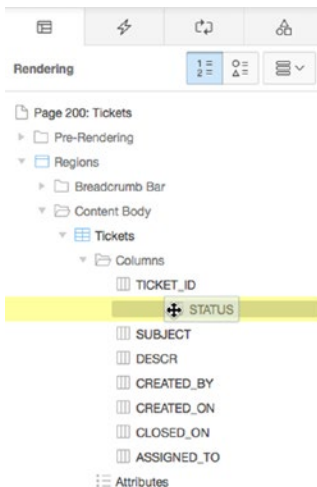
Figure 6-36. *Using drag and drop to re-order columns in a report*

> **6.** **Save** your changes.

Run the application to see the changes to the Tickets report. You should see results like those in Figures 6-37 through 6-39. The Created On and Status values are now more readable, and you can sort by column by clicking the column heading.

Figure 6-37. *The Tickets report*

Figure 6-38. *The Manage Tickets form*

Figure 6-39. *The Ticket Details form*

Session State

Next, let's add a Search field to the report to allow users to filter for a specific ticket they may be interested in. Before we do, here's a brief explanation of session state to help you understand how APEX keeps track of the values associated with a user's session.

Understanding Session State

Session state is what allows APEX to keep track of all the values that belong in a particular user's APEX session. Session state is particularly useful for keeping track of values as a user moves from page to page in the application.

Unlike a stateful database application, where a connection is maintained continuously and all values are retained until changed or removed or until the session ends, an APEX application doesn't maintain a continuous connection to the database. APEX is a *stateless* system—the APEX engine generates HTML pages based on directives stored in the APEX repository. Each page-rendering is a stateless transaction. An APEX session ties the stateless HTML pages together.

137

An APEX session is logically and physically distinct from the underlying database session. A database session is stateful, and an APEX session is stateless. To illustrate the difference, think of a database session as a phone call on a land line. The parties are connected for the duration of the conversation. Both parties have to invest resources to carry on a conversation. Even if no one is talking, the connection—and the link between the two parties—remains, as shown in Figure 6-40.
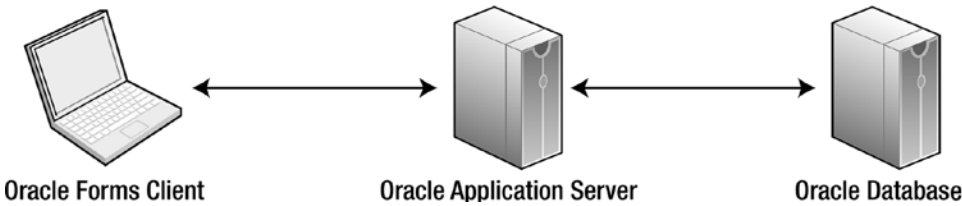
Figure 6-40. *Database session communication*

Think of an APEX session as a text message. The parties aren't directly connected; they push information in one direction at a time, even if the communication is an entire conversation via a series of texts. Figure 6-41 illustrates APEX stateless session communication.
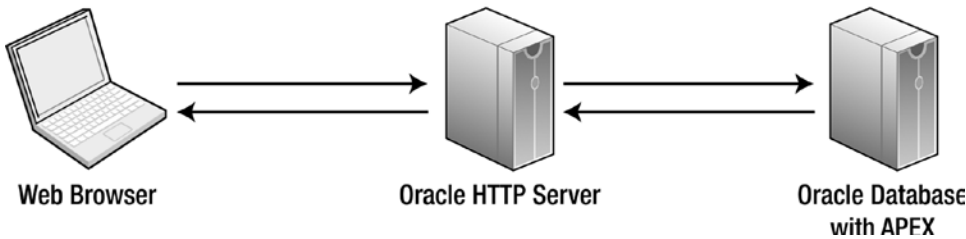
Figure 6-41. *APEX session communication*

Sharing Database Connections

Multiple APEX users can share the same database connection. There is a one-to-many relationship between APEX users and database sessions. This is why APEX can scale as well as it does—it doesn't need dedicated database sessions, only a database session to use to process a request from a user.

APEX, being stateless, must rely on an external mechanism to manage session state. The APEX engine has a built-in session-state management component. This session-state management is an integral part of APEX—it can't be disabled or circumvented.

Each APEX user is assigned a unique session identifier. Session-state management functions the same, regardless of how the user authenticates to the system—APEX authentication, database authentication, custom authentication, or public user. Yes, even unauthenticated users are assigned a session identifier. By default, APEX purges inactive sessions older than 24 hours every 8 hours. APEX session-state values are stored in a table in the database. The APEX engine recognizes the user by their session identifier and retrieves the appropriate set of session-state values for the user's session.

The values of all APEX items, both page items and application items, are tied to this unique session identifier. This identifier is referred to as the `APP_SESSION_ID`. You can see the session identifier in the URL of most pages in an APEX application. It's highlighted in Figure 6-42.
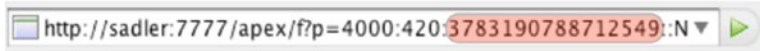
Figure 6-42. *APEX session identifier in an APEX URL*

Setting and Retrieving Session State

Session state is set by user-input items, computations, processes, and PL/SQL code. In PL/SQL, when within an APEX process, you can set an item to be equal to a value, like so:

```
:P1_ITEM_NAME := 'some value';
```

In PL/SQL, when in a stored procedure, you can use the `apex_util.set_session_state` procedure to set a value in session state, as follows:

```
 apex_util.set_session_state( 'P1_ITEM_NAME', 'some value');
```

The syntax to retrieve session state for an item varies according to where you're referencing the item.

In templates or regions, tabs, menus, or lists, use the following substitution-string syntax (and don't forget the trailing dot!):

```
&P1_ITEM_NAME.
```

Use the following syntax in SQL statements:

```
:P1_ITEM_NAME
```

From PL/SQL, use one of the following two options, depending on what type of block or program unit you're in:

```
Anonymous PL/SQL block:          :P1_ITEM_NAME.
PL/SQL Unit Called from APEX:    V('P1_ITEM_NAME')
```

Within conditions, use this syntax:

```
P1_ITEM_NAME
```

■ **Note**   The v function just mentioned is an APEX-provided function that retrieves the session-state value of an APEX item. Exercise caution when using this function, because using it in a stored program unit could introduce performance issues.

Viewing Session State

To view session state, click the Session link on the Developer toolbar. You should see a page like that shown in Figure 6-43. Then use the Page, Find, and Views parameters to view session state for the application. The drop-down View menu shown in Figure 6-44 allows you to view Page Items, Application Items, Session State, Collections, and All of the Above.
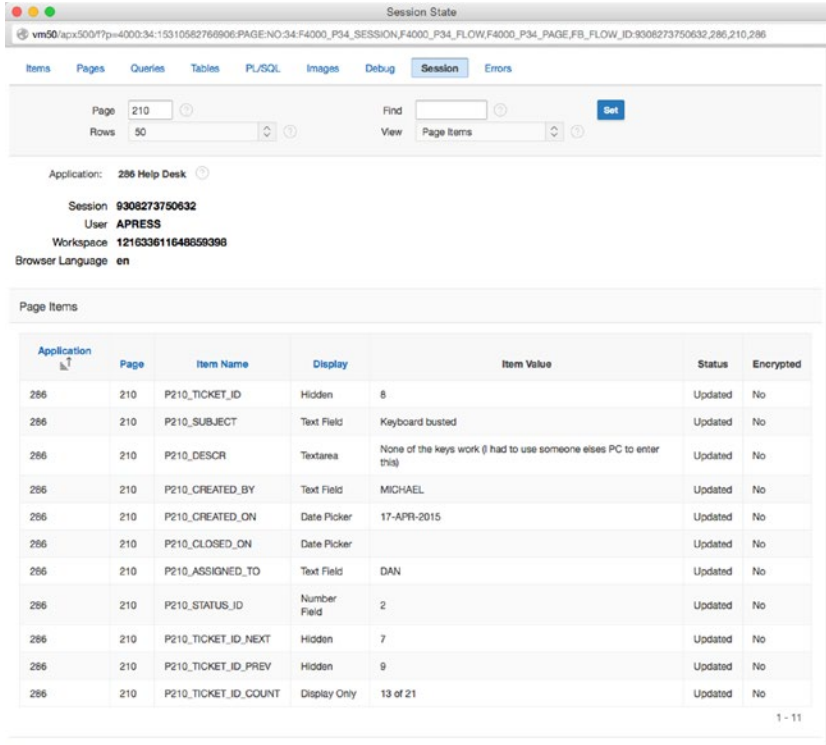
Figure 6-43. *Viewing the session state of Page Items*

Figure 6-44. *Choosing to view session state for all items in the application*

APEX Items

There are two types of APEX items: page items, which are displayed to the user on a page, and application items, which hold values in an application but aren't displayed. When referencing item values of either type in queries, you should use bind variables. You may also want to reference some of the built-in items that are available.

Page vs. Application Items

APEX page items are the UI controls that let users view and enter data—Text Field, Textarea, Select List, Checkbox, and so on. Page items are associated with a specific page and have UI properties associated with them; the item is displayed to the user (or not) according to the UI properties. Figure 6-45 shows the available APEX page item types, as displayed as part of the components gallery. See the APEX documentation for more information on page item types and their attributes.
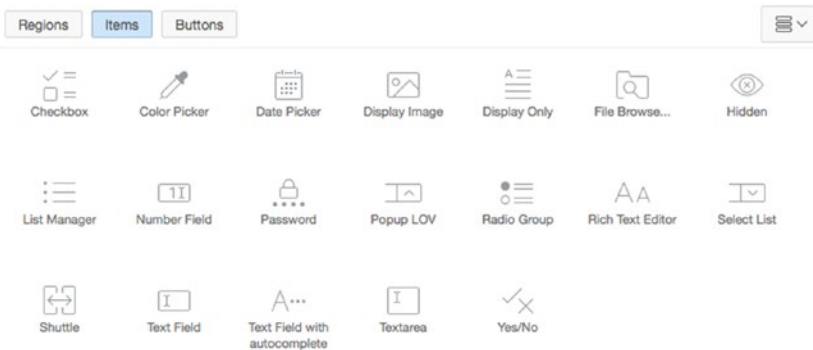
Figure 6-45. *APEX page item types*

Application items aren't associated with a page and have no UI properties. They hold values in an application that are essential but are not necessarily displayed. You can use an application item much like a global variable. For example, you may need to calculate sales tax based on the state the user lives in. You could read that sales tax percentage from a table when the user logs in and keep the value in an application item for use throughout the user's session.

The Importance of Bind Variables

When referencing APEX item values, particularly in SQL queries in your APEX application, it's important to think about SQL security basics, including SQL injection. Consider the example of an online form that allows a user to sign on with a username and password, which ultimately executes this query:

```
SELECT COUNT(*) FROM users
WHERE username = '&username'
  AND password = '&password'
```

If you enter this password

```
I_dont_know OR 'x' = 'x
```

the resulting SQL is

```
SELECT COUNT(*) FROM users
WHERE username = 'SCOTT'
  AND password = 'I_dont_know' OR 'x' = 'x'
```

This SQL statement erroneously returns 1, indicating True, rather than No data found. The user is allowed in! Not good. To prevent the injection of unintended SQL, use bind variables in the SQL query, like so:

```
SELECT COUNT(*) FROM users
WHERE username = :USERNAME
  AND password = :PASSWORD
```

Now try entering the following as your password:

```
I_dont_know OR 'x' = 'x
```

Unless this entire string is specifically your password, the database returns No data found. Your attempt to sneak past the login fails.

We recommend the use of bind variables whenever possible. They prevent SQL injection and improve SQL performance.

Built-In Items

APEX includes several built-in items for referencing key APEX application-wide session-state values. These are set automatically by APEX and are available for reference by the developer throughout APEX. The most common of these are as follows:

- APP_ID: The application identifier of the currently running application

- APP_ALIAS: The application alias of the currently running application

- APP_USER: The currently signed-on user

- APP_SESSION: The session identifier of the currently signed-on user

- APP_PAGE_ID: The currently running page identifier

APEX URL Syntax

Every APEX page is a call to the APEX engine. Every APEX URL is really a call to a specific page and passes various parameters. Figure 6-46 shows the URL syntax.

```
        f?p=
       APP_ID:
    APP_PAGE_ID:
    APP_SESSION:
       REQUEST:
         DEBUG:
     Clear Cache:
     item1,item2:
itemValue1,itemValue2:
    printerFriendly
```

Figure 6-46. *APEX URL syntax*

f?p is the call to the f PL/SQL procedure passing the argument p. The argument is actually a concatenation of nine arguments combined into one, delimited by a colon. The nine elements of the p argument are the same for all APEX page requests. You may omit one or more of the arguments, but you must include the colon delimiters as placeholders.

The elements that form the p argument are as follows:

- APP_ID: The application number or alias

- APP_PAGE_ID: The page number or alias

- APP_SESSION: The APEX session identifier

- REQUEST: The HTML request

- DEBUG: A debug flag, set to YES or NO or omitted to use the current value of the debug flag

- Clear Cache: A list of pages for which to clear the cache

- *Item names*: A list of APEX item names, separated by commas

- *Item values*: A list of APEX item values, separated by commas, that correspond in order to the items specified in the list of item names

- Printer Friendly: A flag that determines whether the page is rendered in Printer Friendly mode

It's easiest to understand the APEX URL syntax by looking at a few examples. Table 6-1 shows several examples and explains them.

Table 6-1. *APEX URL Examples*

f?p=&APP_ID.:10:&APP_SESSION.::::10	Calls page 10 of the current application using the current session and clears the session cache for page 10
f?p=&APP_ID.:5:&APP_SESSION.:::NO::P2_ID:1234	Calls page 5 of the current application using the current session, not in Debug mode, setting the value of P2_ID to 1234
f?p=&APP_ID.:5:&APP_SESSION.::YES	Calls page 5 of the current application using the current session in Debug mode

As you can see, the APEX URL not only supplies directions to the server, but is also your key to what page is being requested, with what request, and with what values. So, how does this URL syntax tie in to your work on the Help Desk application?

APEX applications store all values in an APEX session, which is securely bound to a specific user and user session. Values stored in this user session can easily be set or read by a developer. Any item—application or page—can be easily referenced from anywhere within your APEX application. Values can be referenced and passed to APEX as part of the p parameter so as to control which APEX page is rendered and what values are displayed on that page.

As the volume of data in your system grows, you need a quick way to sort through it and control what data is passed to what page. You can add a page item and then use the value of that item to filter the SQL statement for the report on page 200 of the application. In fact, an item in APEX can be referenced in a SQL or PL/SQL region, as in the predicate of a query, by using the bind variable syntax (:P1_ITEM_NAME), and as part of the APEX URL.

Getting back to the wizard-generated Tickets report, you can apply what you just learned about session state, APEX items, and the APEX URL to add a new item called P200_SEARCH that the user can use to filter the report. After you make these report modifications, take a closer look at the components and attributes of an APEX report.

Searchable APEX Reports

Reports with Edit links let users scan a list of rows and choose one to modify. Scanning works well for reports that are short, but when reports are long, especially more than a page or two, it's time to add some search functionality to help a user quickly zero in on a record to edit.

Creating a Searchable APEX Report

You've already modified the Tickets report generated by the Master Detail Form wizard to add sorting, CSV export capability, and a readable status value. As generated, the report has an Edit link on the first column, which navigates to a Ticket—Ticket Details master–detail form. For the user to find the correct ticket to edit, you need a search function. In the next series of steps you will add a search item and a Go button to activate the search, and you will modify the report query to filter on the search value. We'll use two different methods to place the items in the grid layout. Know that both methods work equally well; which you use depends on which you're more comfortable with. First, let's create the search field:

1. Edit **Page 200** of the application.

2. Create a new item in the **Tickets** region by right-clicking the region name and selecting **Create Page Item**.

3. In the **Properties Editor**, enter P200_SEARCH for the **Name** and set the **Label** to Search, as shown in Figure 6-47.

Figure 6-47. *Setting the attributes of the newly created item*

145

4. In the **Settings** properties group, set the value of **Submit When Enter pressed** to **Yes**.

Although you just set the item attributes so that the page is submitted when the Enter key is pressed, it's still a good practice to provide a way to submit the page using the mouse. Next, you'll use the component gallery and drag and drop to create a new button that, when clicked, processes the item value, stores it in session state, and then reloads page 200:

5. In the Component Gallery at the bottom of the screen, select **Buttons** as the component type. Click and drag the **Text** button so that it is positioned directly beside the P200_SEARCH item you created in the previous steps, as shown in Figure 6-48.
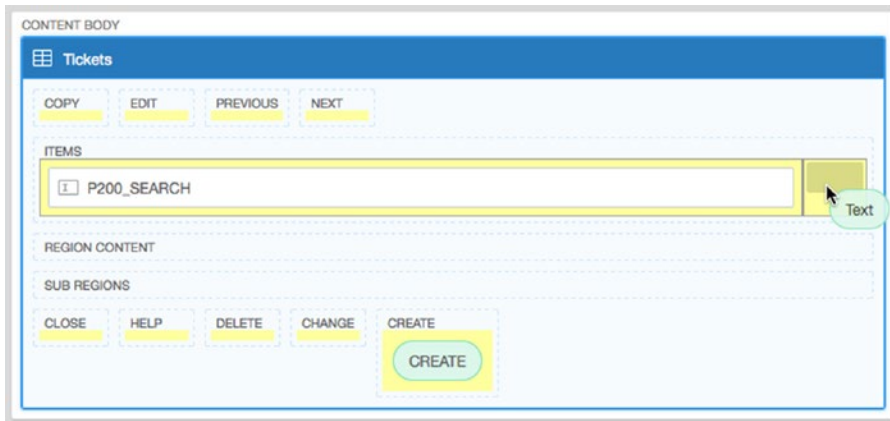
Figure 6-48. *Creating a Go button for the search function*

6. In the Properties Editor, enter P200_GO as the **Button Name** and Go as the **Label**. Leave all the other attributes alone.

Next, you'll adjust the report query to apply the P200_SEARCH filter. You'll add a line to the query predicate that uses the value stored in P200_SEARCH as a filter:

7. Edit the **Tickets** region definition by clicking its name in the **Rendering** tree.

8. Click the **Code Editor** button in the upper-right portion of the **SQL Query** attribute.

9. Append the following line to the end of the query, and click **OK**:

```
AND UPPER(subject) LIKE '%'||UPPER(:P200_SEARCH)||'%'
```

10. **Save and Run** your report. Remember to test both the button and pressing Enter while editing the search field. Both should filter the report correctly.

Adding Reset Pagination

Any time you add a search item to a page, it's a very good idea to also add a Reset Pagination process. This prevents the APEX reporting engine from losing its place in a result set. In this case, there is only one way to create the process, as there are no process components in the Gallery:

1. Edit **Page 200** of the application.

2. **Navigate** to the **Processing** tab of the **Tree Pane**.

3. Right click on the **Processing** node of the tree and select **Create Process** from the context menu.

4. In the **Properties Editor**, set the **Name** to Reset Pagination Process and select **Reset Pagination** as the **Type**, as seen in Figure 6-49.
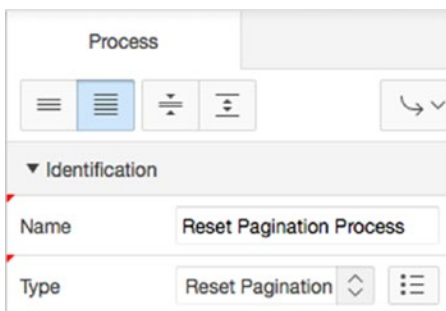
Figure 6-49. *Specifying process options*

5. **Save and Run** the application.

The search function should work both when the user presses Enter and when the user clicks the Go button. But let's go one more step and alter the Subject column so the search term is highlighted in red:

1. Edit **Page 200** of the application.

2. **Navigate** to the Rendering tree and edit the **Subject** column by clicking its name.

3. In the Properties Editor, find the **Column Formatting** attributes group and enter &P200_SEARCH. in the **Highlight Words** element. Make sure you include the period (.) at the end. If you forget it, the variable won't be parsed correctly, and therefore the value won't be highlighted.

This process uses APEX session state to indicate that the value the user entered into P200_SEARCH should be used to highlight that same text in the Subject column. Continue as follows:

4. **Save and Run** your application.

Now, when you enter a search value, the matching rows are returned with the search term highlighted in red. In just a few minutes, you've created a sortable, searchable report for your Help Desk system. Let's look at what the report looks like behind the scenes. Figure 6-50 shows the components as seen from the various tabs of the tree pane.
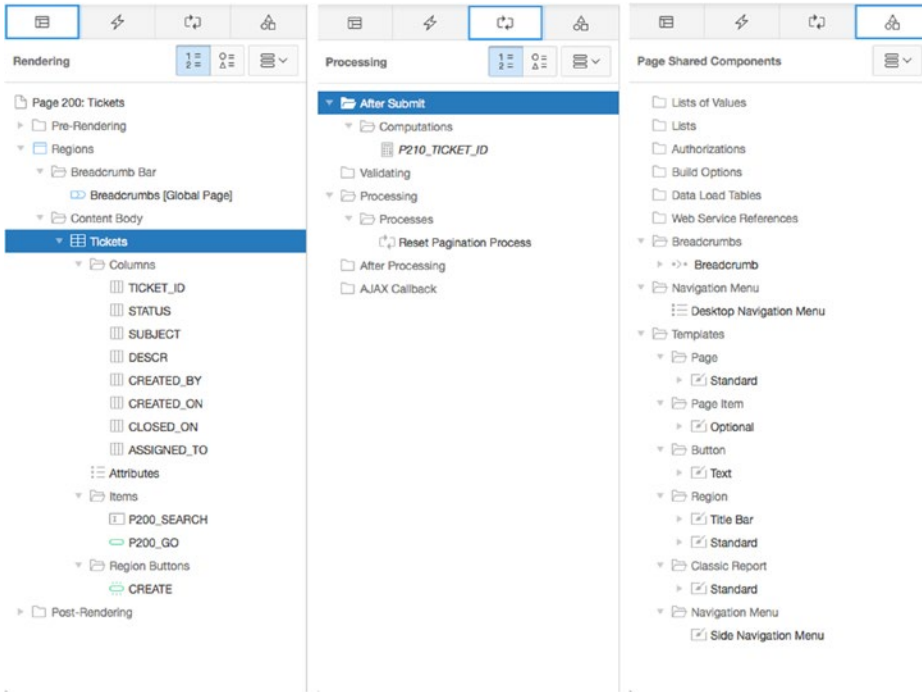
Figure 6-50. *The searchable report as seen from the various tabs of the tree pane*

Looking Behind the Scenes—APEX Report

Let's take a closer look at the components and attributes of the Tickets report. Edit page 200 to view the Rendering, Processing, and Shared Components tabs of the Application Builder. In the Rendering tab, you have a single Tickets region that contains report columns, the two items you just added for search capability, and a Create button. Click the Tickets region name to select it. Now, in the Properties pane, you can see the details shown in Figure 6-51.

Figure 6-51. The Tickets report region source with the search filter

Here you see that the region type is Classic Report. The source for this region is your SQL query on the TICKETS table with the modified WHERE clause to add the filter on the P200_SEARCH item, referencing P200_SEARCH as a bind variable. You can use the Code Editor button to get a better view of the SQL statement if you like.

By clicking the Attributes child node in the Rendering tree at the left of the page, you are able to view and edit the visual attributes of the report region. Also, in the Rendering tree you can see the list of report columns.

By selecting one (or many) of the columns, you can adjust the heading, column width, column alignment, and heading alignment; you can also decide whether the column is shown, whether a sum is required, and whether you want to enable sorting on the column. The columns may be reordered by dragging and dropping them into the order in which you want them.

In the Shared Components tree, you see the expected objects for the navigation, the breadcrumb, and the page, as well as the region, report, label, and button templates. It's nothing new, but be glad the wizard has built these for you.

Next, let's focus on the Tickets and Ticket Details forms, the other components generated by the Master Detail Form wizard.

Looking Behind the Scenes—APEX Master–Detail Forms

Edit page 210 to view the Page Rendering, Page Processing, and Shared Components regions of the Application Builder. You should see results similar to those shown in Figure 6-52.
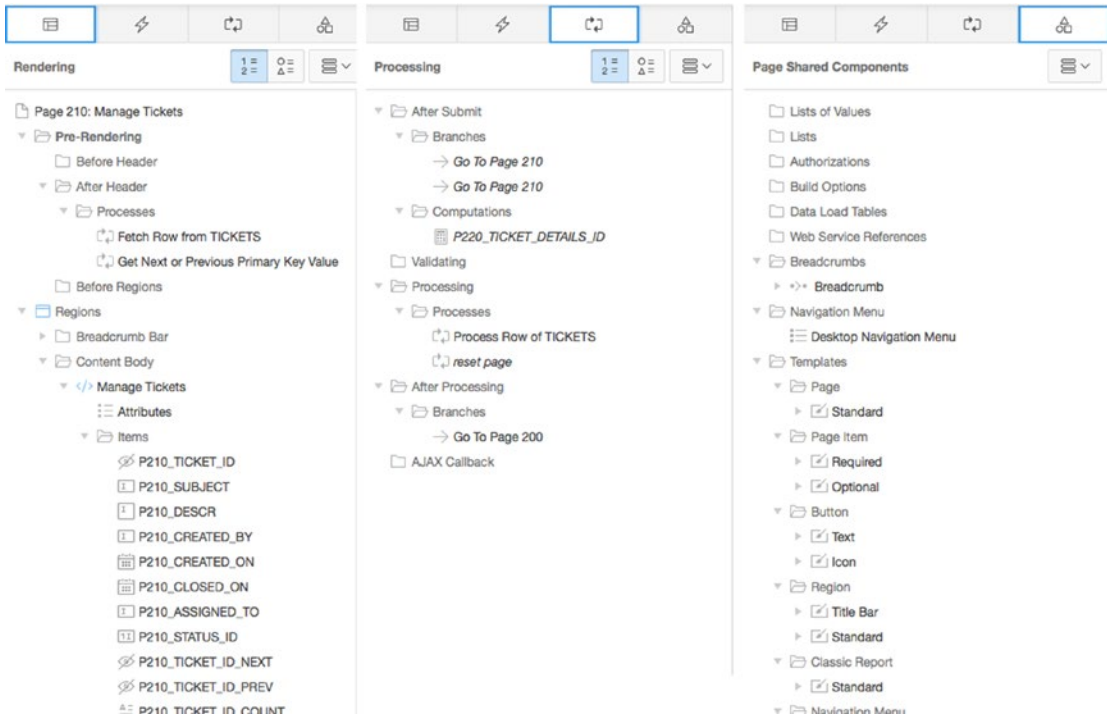
Figure 6-52. *Master Detail page as shown from the various tabs of the tree pane*

In the Rendering tab, you have two After Header processes, a Manage Tickets HTML region that contains your form items, and a Ticket Details report region.

The two After Header processes, Fetch Row from TICKETS and Get Next or Previous Primary Key Value, do exactly what their names imply. The Fetch Row from TICKETS process fetches a row from the TICKETS table for display in the form when the page passes a TICKET_ID. The Get Next or Previous Primary Key Value process gets the next or previous TICKET_ID value in the series and fires in conjunction with the Next and Previous buttons on the master–detail page.

The Manage Tickets region holds an APEX item for each of the TICKETS columns you selected to include in the master–detail form, as well as buttons for cancel, delete, save, create, next, and previous operations.

The Ticket Details region is a report region that displays the ticket details and a Create button, which redirects you to page 220 in order to create additional ticket details.

In the Processing tab, you see two After Submit branches that return you to this same page, an After Submit P220_TICKET_DETAILS_ID computation, two processes (Process Row of TICKETS and Reset Page), and an After Processing branch to page 200. The After Submit computation gets the next TICKET_DETAILS_ID when you click the Create button in the Ticket Details region. The new TICKET_DETAILS_ID is passed to page 220, the Ticket Details form. The Process Row of TICKETS process performs the database DML operations for insert, update, and delete operations on the TICKETS table. The Reset Page process resets (clears) the elements of the page when the Delete button is clicked. The After Processing branch to page 200 redirects the user to page 200, your TICKETS list, on successful processing.

The Shared Components region includes the by-now familiar APEX elements for your page tabs, lists of values, breadcrumbs, and templates.

Moving to page 220, the Ticket Details form, in the Application Builder, you will see elements that look similar to those for the Manage Tickets form on page 210 (see Figure 6-53).
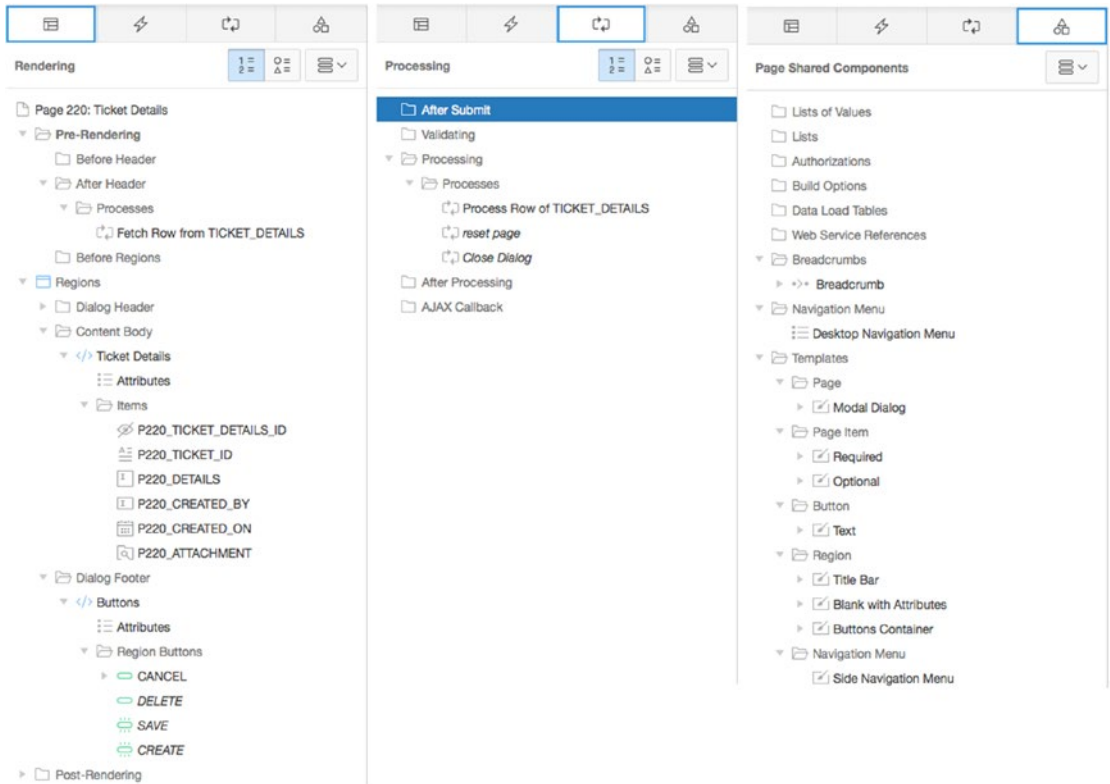
Figure 6-53. *The Ticket Details form as shown from the various tabs of the tree pane*

The Rendering tab includes an After Header Fetch Row from TICKET_DETAILS process, an HTML region that contains items for each of the TICKET_DETAILS columns you selected to include in your master–detail form, and buttons for processing.

The Processing tab includes a Process Row of TICKET_DETAILS process for handling inserts, updates, and deletes on the TICKET_DETAILS table, a Reset Page process to clear the rows on a Delete transaction, and a Go to Page 210 branch that returns the user to the Tickets page upon completion of a Ticket Details transaction.

The Shared Components region on the Ticket Details page includes your page tabs, breadcrumbs, and templates.

Wow! The Master Detail Form wizard created a lot—a fully functional report with master–detail forms, all with no code written on your part. This master–detail example underlines the time-saving value of the APEX wizards in generating APEX components, particularly when creating more complex and multipage components for an application.

More on APEX Forms

When creating forms, the APEX wizards do about 80 percent of what you want them to do. The last 20 percent of fine tuning is up to you, the developer. In this section you will make a number of small changes to the Manage Tickets and Ticket Details forms, with the overall goal of increasing usability.

Item Layout

APEX 5.0 provides two ways to adjust item layout: adjusting certain item attribute settings, and dragging items in the tree view. You'll use both of these methods to adjust the Manage Tickets and Ticket Details forms.

In many of the older themes, APEX laid out form items using standard HTML tables. This was somewhat limiting, as the rows and columns of a table are fairly fixed in terms of layout. Using the new Universal Theme, APEX has introduced the idea of a more loosely defined grid layout. Instead of HTML tables with rows and columns, DIV elements are used to encapsulate each item.

Think of a grid as a coordinate system where items are placed either next to one another or above one another. This grid layout may seem limiting, but you can rearrange items using the grid attributes of items. In this section, you will use the grid attributes of the items on your page to move the Assigned To, Created On, and Created By items to a single row.

Begin adjusting the Manage Tickets form layout by altering the item P210_CREATED_ON so it's automatically populated with today's date. Then, set it so it always displays in read-only mode, preventing users from making any changes:

1. Edit **Page 210** of the application.

2. Edit the item **P210_CREATED_ON** by clicking its name.

3. In the Properties Editor navigate to the **Default** attribute group, as shown in Figure 6-54, set **Type** to **PL/SQL Expression** and enter SYSDATE as the **Default Value**.
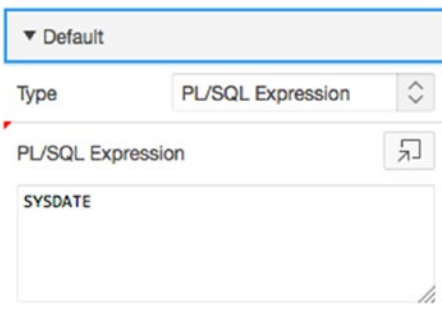
Figure 6-54. *Specifying a default value for a date*

4. In the **Read Only** attribute group seen in Figure 6-55, set **Type** to **Always**.
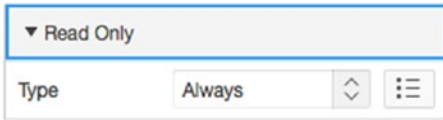
Figure 6-55. *Setting the read-only condition*

You're also going to alter P210_CLOSED_ON. In order to reduce errors, you can use a little-known HTML attribute to make the actual input field read-only. The user is then forced to use the date picker pop-up:

5. Edit the item **P210_CLOSED_ON** by clicking its name.

6. In the **Appearance** attribute group, enter 12 for **Width**.

7. In the **Advanced** attribute group, add the following text immediately after the existing text in the **Custom Attributes** field (as shown in Figure 6-56):

```
readonly="readonly"
```
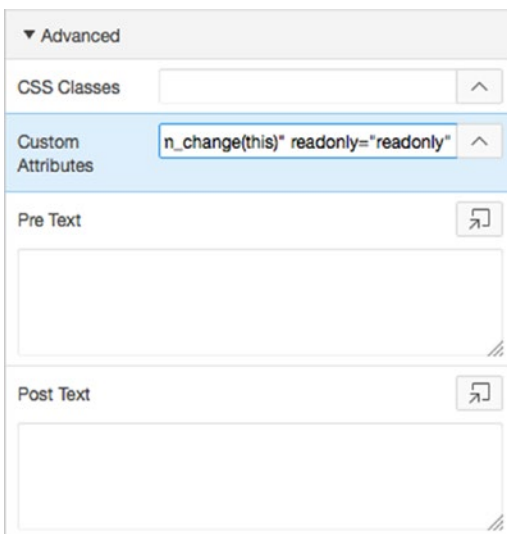

Figure 6-56. *Setting the width and adding an HTML form element*

8. Click **Save**.

Placing Multiple Items in the Same Row

Now, let's rearrange the items on the page so they aren't in a single column but rather are arranged with multiple items are in the same row:

1. Edit **Page 210**.

2. Using your mouse, click and drag **P210_CREATED_BY** in the grid layout so it will be placed in a new grid position to the right of P210_CREATED_ON. As you drag a component around the grid, a yellow box indicates an area where it can be dropped. There is also a position indicator, in the form of a grey box, that indicates the current drop position of the component, as shown in Figure 6-57.
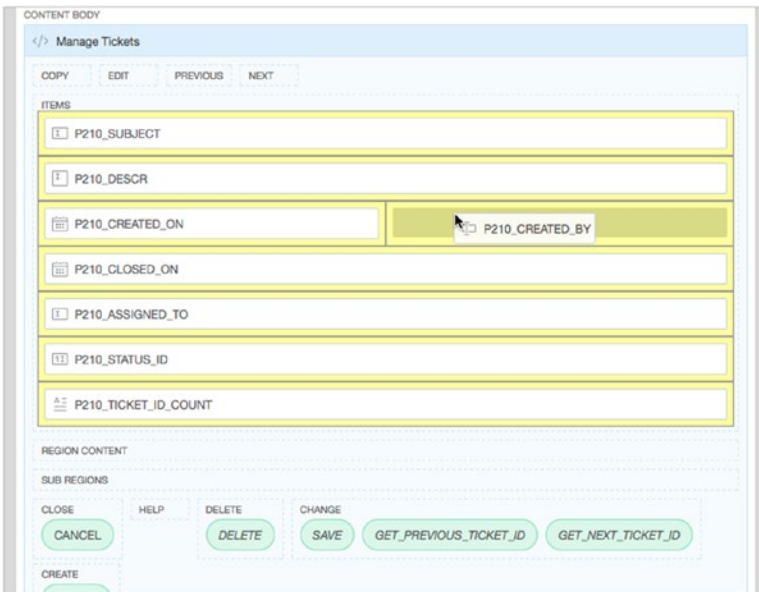
Figure 6-57. *Repositioning P210_CREATED_BY by clicking and dragging the component*

3. When you've positioned the fields correctly, the grid layout looks like Figure 6-58.
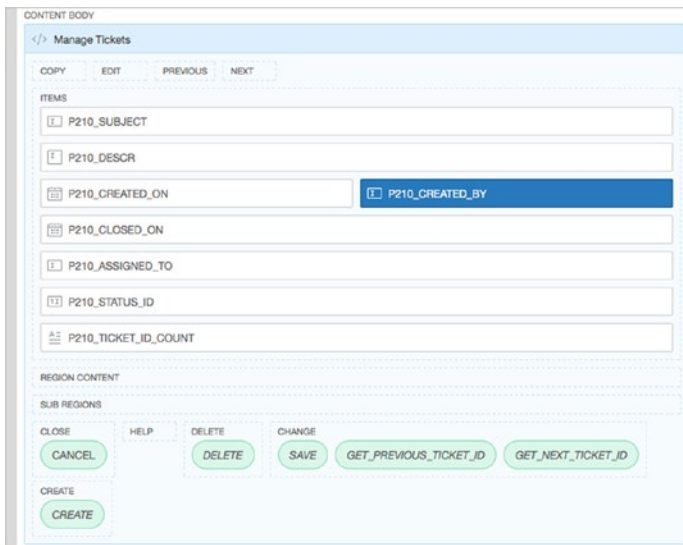
Figure 6-58. *The repositioned component in the grid layout*

Now you need to make sure the Assigned To, Created On, and Created By fields are displayed on the same line:

4. Using the same techniques you just learned, reposition P210_ASSIGNED_TO so that it is directly before P210_CLOSED_ON. When all of the components are positioned correctly, the grid layout will look as shown in Figure 6-59.
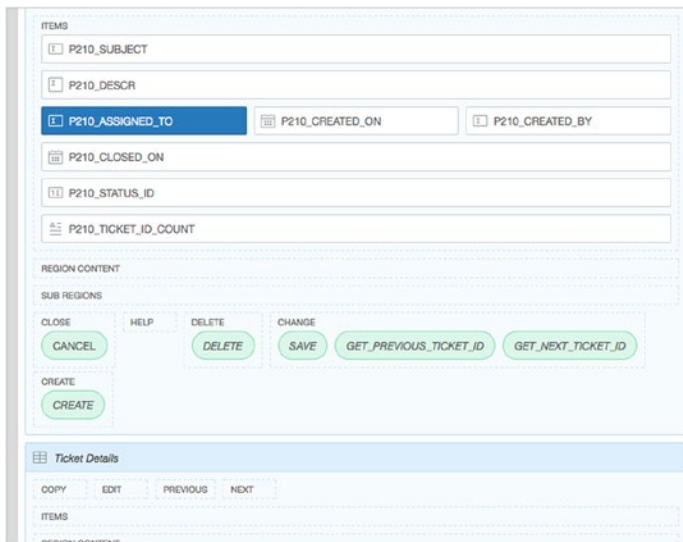
Figure 6-59. *The three repositioned components in the grid layout*

5. Click **Save**.

155

Implementing LOVs

Next, you'll tie the lists of values (LOVs) that you created in Chapter 4 to the P210_ASSIGNED_TO and P210_CREATED_BY items on the form:

1. Edit **Page 210** of the application.

2. Edit the item **P210_ASSIGNED_TO** by clicking its name.

3. In the **Identification** attribute group, set **Type** to **Select List**.

4. In the **List of Values** attribute group (see Figure 6-60), set **Type** to **Shared Component**, **List of Values** to **TECHS**, **Display Extra Values** to **No**, **Display Null Value** to **Yes**, and enter - Select a Tech - for **Null Display Value**.

Figure 6-60. Setting LOV attributes

5. Edit the item **P210_CREATED_BY** by double-clicking its name.

6. In the **Identification** attribute group, set **Type** to **Select List**.

7. In the **List of Values** attribute group (see Figure 6-60), set **Type** to **Shared Component**, **List of Values** to **USERS**, **Display Extra Values** to **No**, **Display Null Value** to **Yes**, and enter - Select a User - for **Null Display Value**.

8. **Save and Run** the application.

You should see results like those shown in Figure 6-61. Notice how the components are very small, even though there seems to be a lot of white space.

Figure 6-61. *The Manage Tickets form using the new field placement*

Clicking the Show Grid button in the Developer Toolbar at the bottom of the page, and then hovering your mouse over one of the field labels, will show that the label is taking up three columns of the grid, as shown in Figure 6-62.

Figure 6-62. *The Manage Tickets form with Show Grid turned on*

The page default is for each label to take up three columns of the grid; when you place more than one item on the same line, it forces the items to shrink to fit the space. We can fix this by adjusting the Label Column Span setting. However, if you only adjust for the three labels in question, you'll end up with form elements that don't line up with the rest of the form. In our case, we want to adjust the labels for all the enterable components on the screen:

9. Edit the following together by using CTRL-Click (COMMAND-Click for Mac):

```
P210_SUBJECT
P210_DESCR
P210_ASSIGNED_TO
P210_CREATED_ON
P210_CREATED_BY
P210_CLOSED_ON
P210_STATUS_ID
```

10. In the Property Editor, navigate to the **Grid** attribute group, as shown in Figure 6-63, set **Label Column Span** to **2**, and click **Save**.
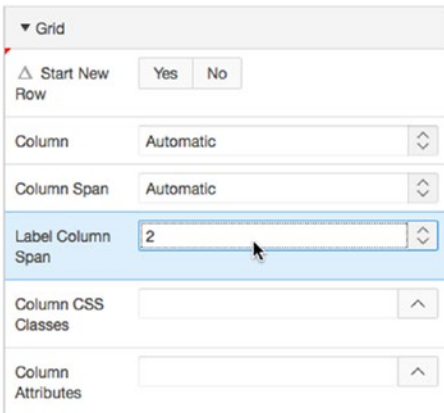
Figure 6-63. *Altering Label Column Span to allow for expanded item size*

11. Once again, run the application, and you will notice the difference in how the items are laid out on the page. You should see results like those shown in Figure 6-64.

Figure 6-64. *Corrected layout for the Manage Tickets form*

Master–Detail Cleanup

You need to make a few more minor tweaks to the master–detail report and form. Let's start by hiding the TICKET_ID column from the detail report and form. At the detail level, TICKET_ID is the foreign key and should not be an editable item:

1. Edit **Page 210** of the application.

2. Expand the **Columns** child node under the **Ticket Details** report node in the **Rendering** tree.

3. Click on the TICKET_ID column and hide it by editing its **Type** attribute and setting the value to **Hidden Column**.

4. Using multi-select, enable sorting for the DETAILS, CREATED_ON, and CREATED_BY columns by setting the **Sortable** attribute to **Yes**, as shown in Figure 6-65.

Figure 6-65. *Specifying whether sortable for columns*

5. Lastly, edit the `TICKET_DETAILS_ID` column and change its **<u>Heading</u>** attribute to `Edit`.

6. Click **Save**.

Finally, make a few small changes to the items on page 220:

1. Edit **Page 220** of the application.

2. Edit the item **P220_TICKET_ID**.

3. In the **<u>Identification</u>** attribute group, set **<u>Type</u>** to **Hidden**.

4. Edit the item **P220_DETAILS.**

5. In the **<u>Appearance</u>** attribute group, set **<u>Height</u>** to **5**.

6. Edit the item **P220_CREATED_ON**.

7. In the **<u>Default</u>** attribute group set **<u>Type</u>** to **PL/SQL Expression**, then enter `SYSDATE` as the **<u>PL/SQL Expression</u>**.

8. In the **<u>Read Only</u>** section, set **<u>Type</u>** to **Always**.

9. Edit the item **P220_CREATED_BY**.

10. Set **<u>Type</u>** to **Select List**. In the **<u>List of Values</u>** attribute group, set **<u>Type</u>** to **Shared Component**, **<u>List of Values</u>** to **TECHS**, **<u>Display Extra Values</u>** to **No**, **<u>Display Null Values</u>** to **Yes**, and enter `- Select a Tech -` for **<u>Null Display Value</u>**.

11. **Save** your changes.

Since Page 220 is set up as a modal dialog, you cannot run the page directly. Instead, you'll have to navigate to either Page 200 or 210 to run the application.

Your master–detail report and form are now complete. Using the Master Detail Form wizard, you generated a report and master–detail form on the `TICKETS` and `TICKET_DETAILS` tables. You modified the report to contain a user-friendly status value, sortable columns, and your preferred date formats. You modified the Manage Tickets and Ticket Details forms to order items on the page, use text areas, and select lists. Along the way, you reviewed the APEX components that make up your report and forms, as well as the form, report, and column attributes available for customizing forms and reports to suit your needs.

APEX Help

Providing help to end users is an often forgotten and typically tedious task. Developers typically take the easy route and skip it altogether, or the task is minimized or cut at the end of a project. Although APEX can't magically incorporate help into your applications, it does make it a lot easier for you, as a developer, to do so.

Adding a Help Text Region

The APEX Help Text region automatically displays any associated help text for a given page and its items. It can be placed on any page, including a global page. Although you can choose a region template for a Help Text region, there is no way to change the style of the actual text. As an example, let's add a Help Text region to page 210 as a sub-region to the master Edit region:

1. Edit **Page 210** of the application.

2. Create a new Help Text region by navigating to the **Regions** pallet of the **Component Gallery** and dragging the Help Text icon to the Sub Regions section inside the Manage Tickets region, as shown in Figure 6-66.
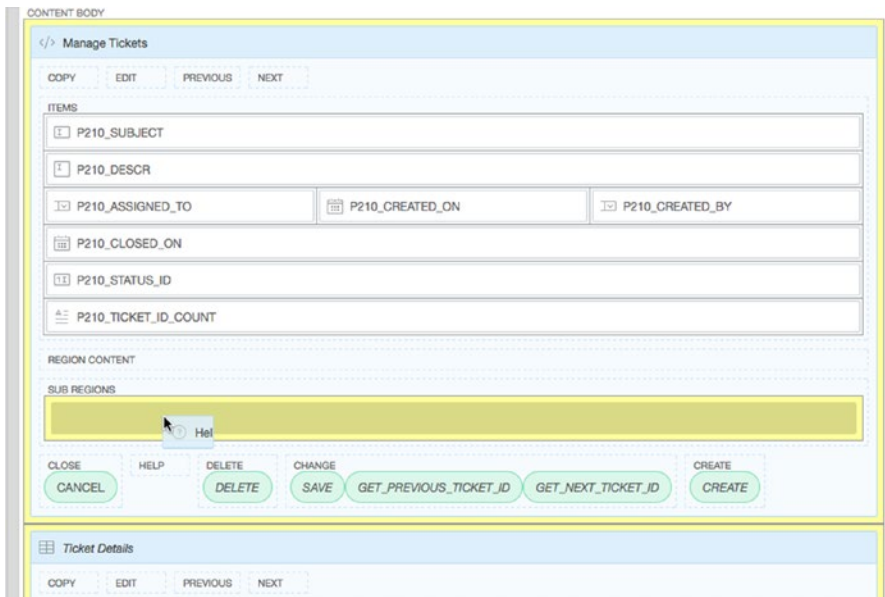
Figure 6-66. *Creating a Help Text region*

3. In the Properties Editor, set the **Name** to **Help**.

4. In the **Appearance** attributes group, set the **Template** to **Collapsible** and then click on the **Template Options** button to expand the Template Options pop-up.

5. Set the **Default State** to **Collapsed** and click **OK**.

6. **Save and Run** your application.

Notice that when you run page 210, you will see the region title Help rendered with a > next to it at the bottom of the Manage Tickets region. The newly created Help region was created as a sub-region, and therefore it appears within its parent region. Clicking the ➤ expands the region; thus, the help text is only displayed when the user explicitly requests it. Currently, the Help region doesn't have any help text. You seed the item-level help text in the next section. You can add page-level help by editing the page definition and entering text into the Help Text input of the Help section.

Seeding Help Text

Notice that not all the items are shown in the Help region. This is because some help text was added to this region when the UI Defaults were defined, but the other items' help text is still empty. Help text defined in the UI Defaults is automatically pulled into any form that is built using those defaults. You can manually add help text by editing each item. You can also seed any APEX items that don't have help text already assigned using yet another APEX wizard.

1. At the upper right in the Application Builder, click the **Utilities** icon, as shown in Figure 6-67, and select **Application Utilities** so as to go to the Application Utilities home page.
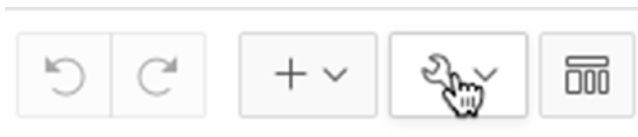
Figure 6-67. Locating the Application Utilities icon

2. In the **Page-Specific Utilities** region at right of the page, click **Item Utilities**.

3. Click **Grid Edit of all Item Help Text**.

The report here shows only those items that already have help text associated with them. However, you can use one of the buttons on this form to seed all empty help text in your application with a single default value. There is no perfect value with which to seed the help text, but something like "Need Help Text" indicates that the help for that item needs to be entered:

4. Click **Seed Item Help Text**.

5. Enter NEED HELP TEXT for **Default Help Text** in the **Seed Item Help** section, as shown in Figure 6-68, and click **Apply Changes**.
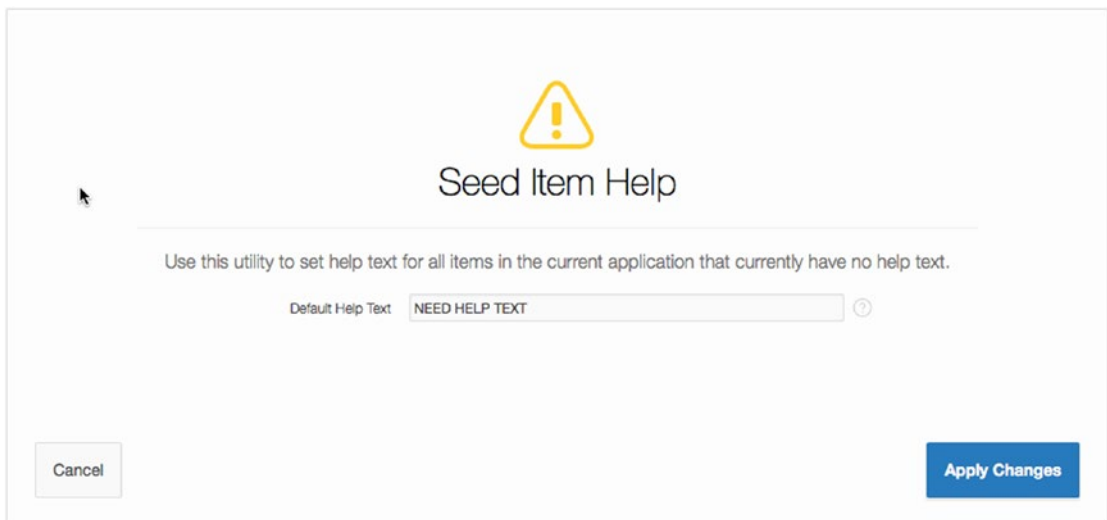
Figure 6-68. Seeding item help

162

The help text has been seeded, and you're taken back to the main report. From here you can narrow the items that are displayed and edit the help text directly:

6. In the Report Filter section at the top of the page, enter 210 for **Minimum Page Number**, and click Go.

At this point, you're viewing all the help text for any item on page 210 or greater in a single interface. Feel free to change the values for any of the items on page 210 in order to see them in the Help region.

Once you've altered and saved your help, run page 210. Note that if you click the question mark icon next to any individual item on page 210, a pop-up window appears, displaying the help specific to that item.

The APEX Help Text region automatically displays the help text for a given page and its associated items. Display of the help text is managed by APEX behind the scenes. Although it isn't very robust—there is no way to alter the look and feel of the region with templates or otherwise—there is now no excuse for not adding help to your application.

Declarative BLOBs

In Oracle, BLOB stands for Binary Large Object and is a data type designed to store binary files. APEX has streamlined how you can manage BLOB columns with a feature called Declarative BLOBs. The APEX wizards recognize a BLOB column and automatically alter the related APEX item and report so as to interact seamlessly with the column. Why do you care about BLOB columns? Using BLOB columns allows you to easily upload and download files, such as documents, spreadsheets, and images, into your applications.

Plan ahead when using the Declarative BLOBs feature. At design time, include these columns in tables that will use Declarative BLOBs:

- FILENAME: Stores the actual file name that is used when a user uploads the file

- MIME_TYPE: Stores the type of the file so browsers know which application to launch (Word for .doc, Excel for .xls, and so on)

- LAST_UPDATED: Stores the date the BLOB was last updated

- CHARACTER_SET: Stores the character set of the BLOB, which is essential for indexing and processing data that resides within the BLOB

The first two columns are essential for reading data out of the BLOB when needed. APEX uses the Number/Date format column attribute of the BLOB column to map these attributes to the BLOB column stored in the database.

If you add a BLOB column after creating a report or form using a wizard, you have to manually set the column or item properties in order to integrate BLOB processing.

Because you added a BLOB column to the TICKET_DETAILS table when you ran the SQL script, some things have been done for you. But you still need to do several things to use Declarative BLOBs properly. First, you have to map the FILENAME and MIME_TYPE columns to the form that is used to upload the document, so that these details are saved in the database. Let's address the form on page 220 first.

1. Edit **Page 220** of the application.

2. Edit the item **P220_ATTACHMENT**. In the Settings section, you will see the fields shown in Figure 6-69.

Figure 6-69. *Specifying BLOB settings*

3. In the **Settings** attribute group, enter MIME_TYPE for **MIME Type Column**, FILE_NAME for **Filename Column**, and Download for **Download Link Text**.

4. Click **Save**.

Next, alter the report on page 210:

1. Edit **Page 210** of the application.

2. Edit the **Ticket Details** region by clicking its name.

3. Locate and open the file ch6_blob_report.txt, which you can find where you extracted the class files from earlier, and copy the contents into the **SQL QUERY** attribute, replacing all text that is currently there. See Figure 6-70.

Figure 6-70. *Entering the report query with a BLOB column*

Notice the change in the last column in the select list. Using `dbms_lob.getlength` indicates to APEX whether the ATTACHMENT BLOB column contains any data. If it does, the query returns a number greater than 0.

Now you need to alter the report column to display a link that allows the end user to download any document that may have been uploaded:

4. Expand the **Columns** node under the **Ticket Details** node in the Rendering tree.

5. Edit the ATTACHMENT column, changing the **Type** attribute to **Download BLOB**.

6. In the newly visible **BLOB Attributes** attribute group, enter TICKET_DETAILS for **Table Name**, ATTACHMENT for **Blob Column**, TICKET_DETAILS_ID for **Primary Key Column 1**, MIME_TYPE for **Mime Type Column**, FILE_NAME for **Filename Column**, and under Appearance enter Download for **Download Text**, as shown in Figure 6-71.

165

Figure 6-71. *Modifying the BLOB column attributes*

7. **Save** your changes.

Run the application. Test the file upload and download capabilities by attaching a file to one of the Ticket Details records and then downloading it from the report.

This ability to easily upload and download files in APEX is extremely useful in building web applications where users need to upload and download data for whatever purpose. The Declarative BLOBs feature of APEX makes it simple for developers to add upload and download capabilities to an application.

Summary

You've reviewed most of the APEX form and report types and walked through building various forms and reports for your Help Desk system using the APEX form and report wizards. Along the way, you've learned about APEX items, session state, the APEX URL syntax, adding help to APEX pages, and incorporating upload and download functionality by using the Declarative BLOBs feature. That's a lot to digest, but the APEX wizards have done most of the work for you.

The common theme here is that the APEX form and reports wizards are huge time-savers for developers, creating all the objects—items, buttons, branches, processes, and so on—needed for a working form or report. You can then alter the created objects to quickly customize the form or report to suit your needs.

Still, you haven't strayed far from what APEX builds for you, and you've covered only the simplest types of forms and reports. The next chapter will look at more-complex types of APEX forms and reports, also generated by wizards.

■ ■ ■

Forms and Reports: Advanced

This chapter will focus on more complex types of forms and reports; it will also introduce charts and maps. Although these are more complex types of forms and reports, they're most often created by using the APEX form and report wizards.

In the sections that follow, you will learn how to use the APEX form and report wizards to add pages to your Help Desk application in order to manage multiple tickets on a single page, allow some interactive analysis of ticket data, and visualize tickets by date and status. To do so, you will create a tabular form, an interactive report, a calendar, and a pie chart, each demonstrating one of the more advanced types of APEX forms and reports.

Tabular Forms

Tabular forms allow users to edit both rows and columns of data at once, much like a spreadsheet. The developer can choose a different element type for each column—text box, text area, select list, check box, radio group, and so on. Users can make changes to multiple data elements and submit them as a single transaction. APEX tabular forms handle inserts, updates, and deletes—all with no code!

The APEX wizards create all of the required elements for a fully operational tabular form. Like all APEX forms, there is no logical relationship between items that make up a tabular form. Once the wizard creates the items, they're indistinguishable from other APEX page items and can be modified independently of one another. However, I recommend exercising caution when making modifications to items generated by an APEX wizard; doing so can cause the tabular forms to become inoperable.

You can opt to bypass the wizard and create your own tabular forms. As your application becomes more sophisticated, you may find it more efficient to create forms manually. However, this book focuses on the wizard approach.

Creating a Tabular Form

In this section you will create a new page that contains a tabular form based on the TICKETS table. The form allows multiple tickets to be edited on the same page. You will then alter the display properties of the tabular form's columns. Proceed as follows:

1. Navigate to the Application Builder Home Page for your application.

2. Click the **Create Page** button at upper right on the page.

3. Select **Form** and click **Next**.

4. Select **Tabular Form** and click **Next**.

5. Select your schema for **Table/View Owner**, and then select **TICKETS (table)** for
Table/View Name.

6. Make sure that **Allowed Operations** is set to **Update, Insert and Delete**.

7. By default, Use User Interface Defaults and all the columns are already selected,
as shown in Figure 7-1. Click **Next**.
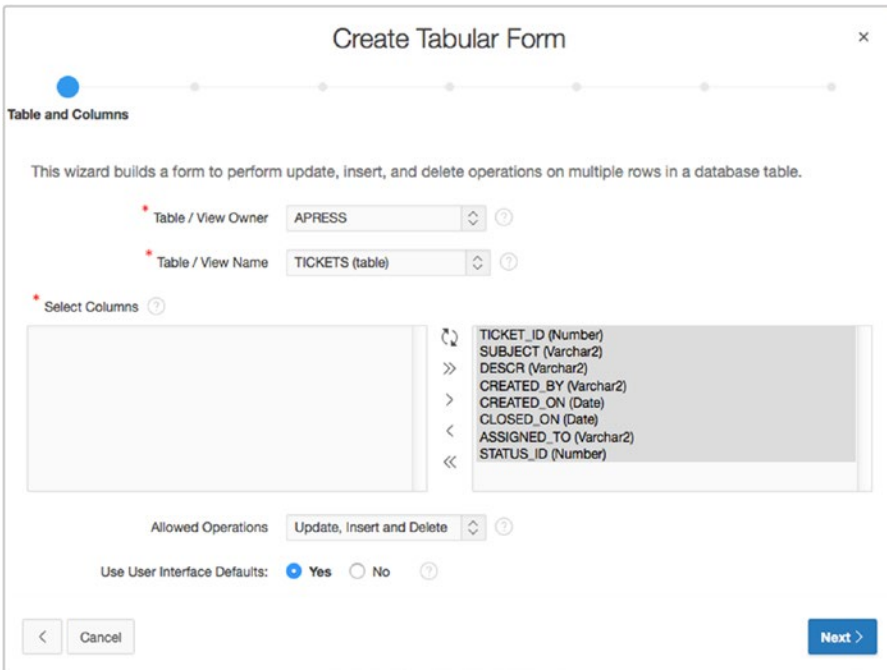
Figure 7-1. Selecting columns for a tabular form

8. Set **Primary Key Type** to **Select Primary Key Column(s)**.

9. Set **Primary Key Column 1** to **1. TICKET_ID (Number)**, and click **Next**.

10. Set **Source Type** to **Existing Trigger** and click **Next**.

11. Select all columns as **Updatable Columns**, as shown in Figure 7-2,
and click **Next**.

Figure 7-2. Selecting updatable columns for a tabular form

12. Enter 230 for **Page** and Manage Multiple Tickets for **Page Name** and **Region Title** as shown in Figure 7-3.

13. Set **Page Mode** to **Modal Dialog**

14. Set **Breadcrumb** to **Breadcrumb**.

15. When the page refreshes, set **Entry Name** to **Manage Multiple Tickets** and **Parent Entry** to **Tickets (Page 200)**, as shown in Figure 7-4, and click **Next**.
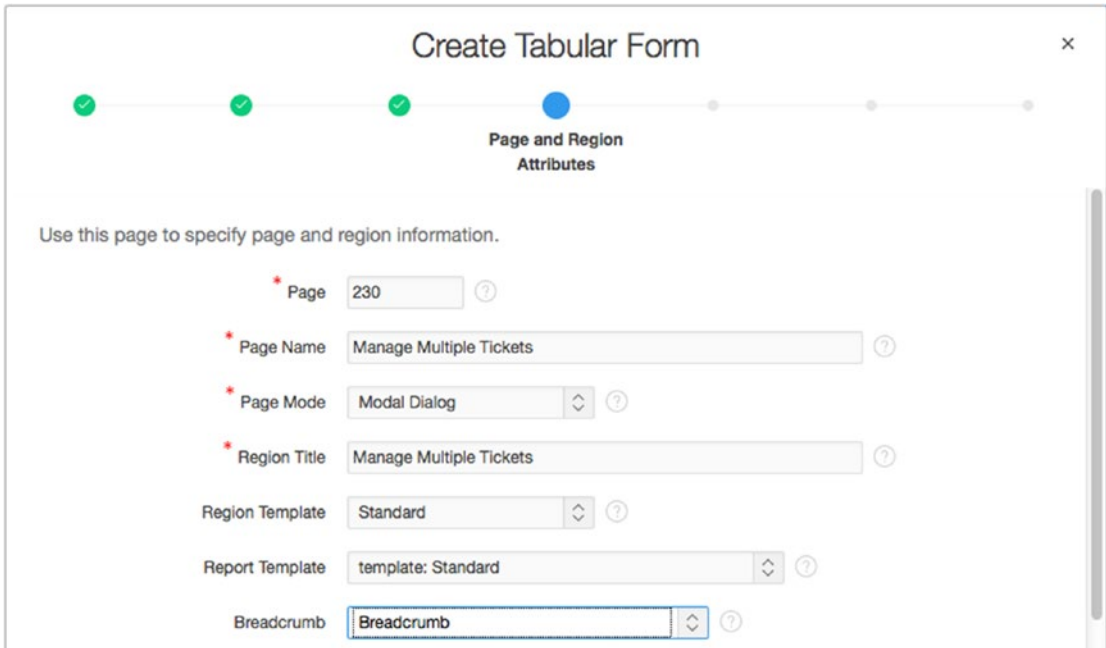
Figure 7-3. Identifying page and region attributes for a tabular form
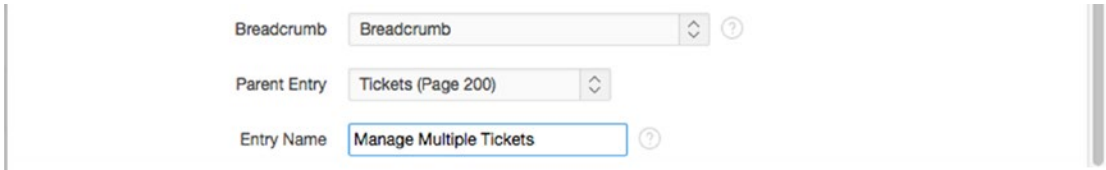
Figure 7-4. Creating a breadcrumb entry for a tabular form

16. For **Navigation Preference**, select **Identify an existing navigation menu entry for this page**. When the dialog refreshes, set **Existing Navigation Menu Entry** to **Tickets** and click **Next**.

17. Change the **Add Row Button Label** to Add Tickets.

18. Check your selections in the **Confirmation** scrollable region, as shown in Figure 7-5.

Figure 7-5. *Checking our choices in the Confirmation region*

19.   Click **Create**.

Modifying a Tabular Form

Your tabular form will work, but currently there is no way to navigate to it. First, you need to create a button on page 200 that links to your new tabular form:

1. Edit **Page 200** of the application.

2. Create a new button by dragging a **Text[Hot]** button from the **Component Gallery** to the **Create button** position of the **Tickets** region, as shown in Figure 7-6.
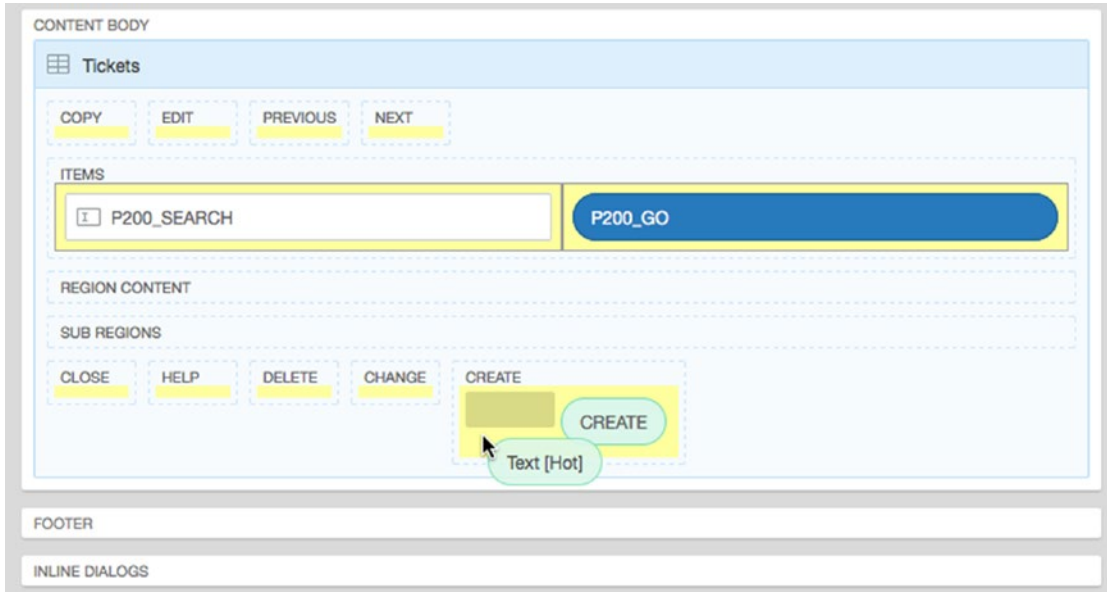
Figure 7-6. *Dragging a new button to the Tickets region*

3. Enter MANAGE_MULTIPLE_TICKETS for **Button Name** and Manage Multiple Tickets for **Label**, as shown in Figure 7-7.
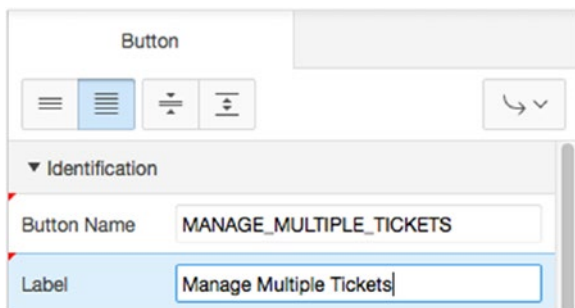
Figure 7-7. *Specifying button attributes*

4. In the Behavior attribute group, set **Action** to **Redirect to Page in This Application.** Click the **Options Dialog** button next to **Target**. Once the Options Dialog appears, set **Page** to 230, set **Reset Pagination** to **YES**, as shown in Figure 7-8, and click **OK**.

Figure 7-8. *Specifying button action attributes*

5. **Save and Run** your application.

At this point, you should be able to navigate to your tabular form from page 200 by clicking the Manage Multiple Tickets button.

However, now you need to make some cosmetic modifications so you can better control data entry and the look and feel of the dialog.

First, let's increase the size of the dialog window so we can see all the elements of the form:

1. Edit **Page 230** of the application.

2. Select **Page 230: Manage Multiple Tickets** in the **Rendering** tab of the **Tree Pane**.

3. In the Properties Editor set the dialog **Width** property to 1200 and the **Height** property to 720.

4. Click **Save**.

Next, you'll make some changes to the columns of the Tabular Form.

5. Expand the **Columns** node under the **<u>Manage Multiple Tickets</u>** node of the tree in the **<u>Rendering</u>** tab.

6. Edit the **TICKET_ID_DISPLAY** column and set **<u>Type</u>** attribute to **Hidden Column**.

7. Multi-select **SUBJECT**, **DESCR**, **CREATED_BY**, **CREATED_ON**, **CLOSED_ON**, **ASSIGNED_TO**, and **STATUS_ID** and make sure their **<u>Sortable</u>** properties are set to **YES**.

8. Multi-select the **SUBJECT** and **DESCR** columns.

9. In the Properties Editor, set **<u>Type</u>** to **Text Area**, **<u>Width</u>** to 16, and **<u>Height</u>** to 3.

10. Edit the **ASSIGNED_TO** column.

11. In the Properties Editor, set **<u>Type</u>** to **Select List**.

12. In the **<u>List of Values</u>** attribute group, set **<u>Type</u>** to **Shared Component**, **<u>List of Values</u>** to **TECHS**, **<u>Display Extra Values</u>** to **No**, **<u>Display Null</u>** to **Yes**, and enter - Select a Tech - for **<u>Null Display Value</u>**, as shown in Figure 7-9.

Figure 7-9. Specifying a LOV for the ASSIGNED_TO column

13. Edit the **CREATED_BY** column.

14. In the Properties Editor, set **<u>Type</u>** to **Select List**.

15. In the **<u>List of Values</u>** section, set **<u>Type</u>** to **Shared Component**, **<u>List of Values</u>** to **USERS**, **<u>Display Extra Values</u>** to **No**, and **<u>Display Null</u>** to **Yes**, and enter - Select a User - for **<u>Null Display Value</u>**, as shown in Figure 7-10.

Figure 7-10. *Specifying LOV attributes for the* CREATED_BY *column*

16. Edit the **STATUS_ID** column.

17. In the Properties Editor, navigate to the Default attribute group and set **Type** to **PL/SQL Expression** and enter get_status ('OPEN') for **Default**.

18. **Save** the edits you just made and then run your application. Note: Because page 230 is a modal dialog page, you will need to run it from the button you created on page 200.

Looking Behind the Scenes

Let's take a look at what the Tabular Form wizard has created for you—the contents of your tabular form. Edit page 230 to examine the various tabs of the Tree Pane. They should look similar to those shown in Figure 7-11.
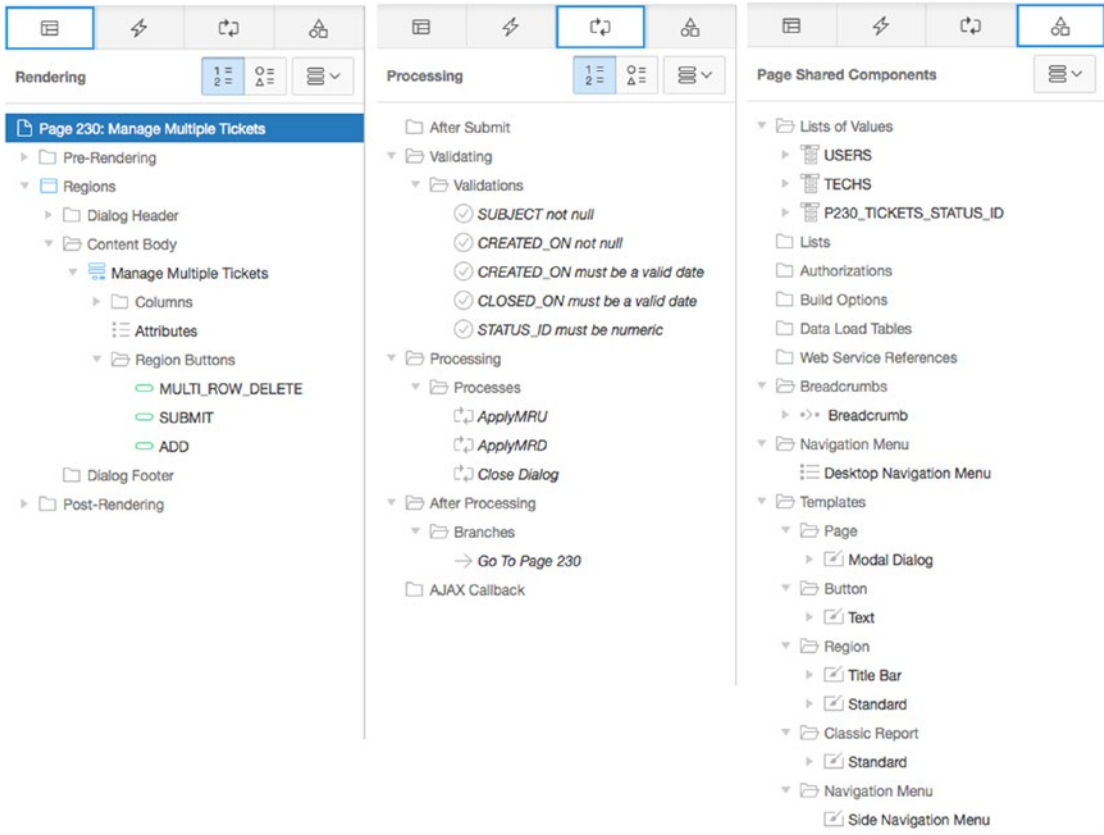
Figure 7-11. *The various tabs of the Tree Pane for page 230*

In the Rendering tab, APEX has created a Report region. But you created a form, didn't you? Despite its name, a tabular form is actually a SQL report with certain column-level options enabled and some processes added to handle data manipulation.

In the Processing tab, in the Processing section, you see two processes: ApplyMRU and ApplyMRD. These special types of processes handle the multiple-row inserts and updates (ApplyMRU) and deletes (ApplyMRD) on the TICKETS table. These processes handle all DML operations on the TICKETS table for you.

APEX has also created validations for several of the columns, which are created automatically based on the TICKETS table column definitions plus any UI Defaults defined on the TICKETS table.

In the Shared Components tab are the usual page and tab templates that are the defaults for your application.

As you can see, the ApplyMRU and ApplyMRD processes make the difference between the Report region being a static report region and being a fully functional tabular form. And it's so much easier to let the APEX wizard create all this for you!

Interactive Reports

Your ticket report is what's called a *classic report*. It's the original style of APEX report and still has practical applications in a variety of situations where the requirement is for a simple list of data with no interactivity. Most applications, including APEX itself, now employ the APEX interactive report, however.

Introduced in APEX 3.1, the interactive reports feature allows APEX to quickly and easily include user-driven ad hoc capabilities in your applications. Interactive reports are greatly enhanced in APEX 5.0. The beauty of APEX interactive reports is that they give the end user powerful ad hoc query capability with exactly zero lines of code written by the developer. End users can customize the following:

- Searching

- Sort order

- Columns

- Breaking

- Highlighting

- Computations

- Aggregations

- Charts

- Group by

- Flashback time

- Saved reports

- Subscription (email notification)

Interactive reports are technically nothing more than a report type. The Create Report wizard steps are similar to what we have already seen, and you will expend the same effort in building an interactive report as you would for a classic report.

Classic reports can be easily converted to interactive reports. There is no way to revert from an interactive report to a classic report, however. (But why would you want to?) The end-user features and overall value of interactive reports are best illustrated with an example, so let's add an interactive report to your application.

Creating an Interactive Report

Interactive reports require nothing more than a SQL query. APEX handles the rest. You start by creating a new page, menu item, and interactive report all at once on a view of your Help Desk data. Begin as follows:

1. Navigate to the Application Builder's Home Page for your application

2. Click **Create Page** button in the upper right of the screen.

3. Select **Report** and click **Next**.

4. Select **Interactive Report** and click **Next**.

5. Enter 300 for **Page Number** and Analysis for **Page Name** and **Region Name**, and set **Region Template** to **Interactive Report**.

6. Set **Breadcrumb** to **Breadcrumb** and, when the page refreshes, click **Next**.
    See Figure 7-12.

Figure 7-12. *Specifying the page number, name, and breadcrumbs for an interactive report*

7. Set **<u>Navigation Preference</u>** to **Create a new navigation menu entry**. When the
    page refreshes, it should look like Figure 7-13. Click **Next**.

Figure 7-13. *Specifying navigation options for an interactive report*

For this report you're going to use the TICKETS_V view instead of the TICKETS table directly. The view joins the TICKETS table to the STATUS_LOOKUPS table so you don't have to do it manually later at the column level:

8. Set the **Source Type** to **Table** and then select **TICKETS_V (view)** for **Table / View Name**. Set **Uniquely Identify Rows by** to **Unique Column**, enter TICKET_ID for **Unique Column**, and click **Next** (see Figure 7-14).
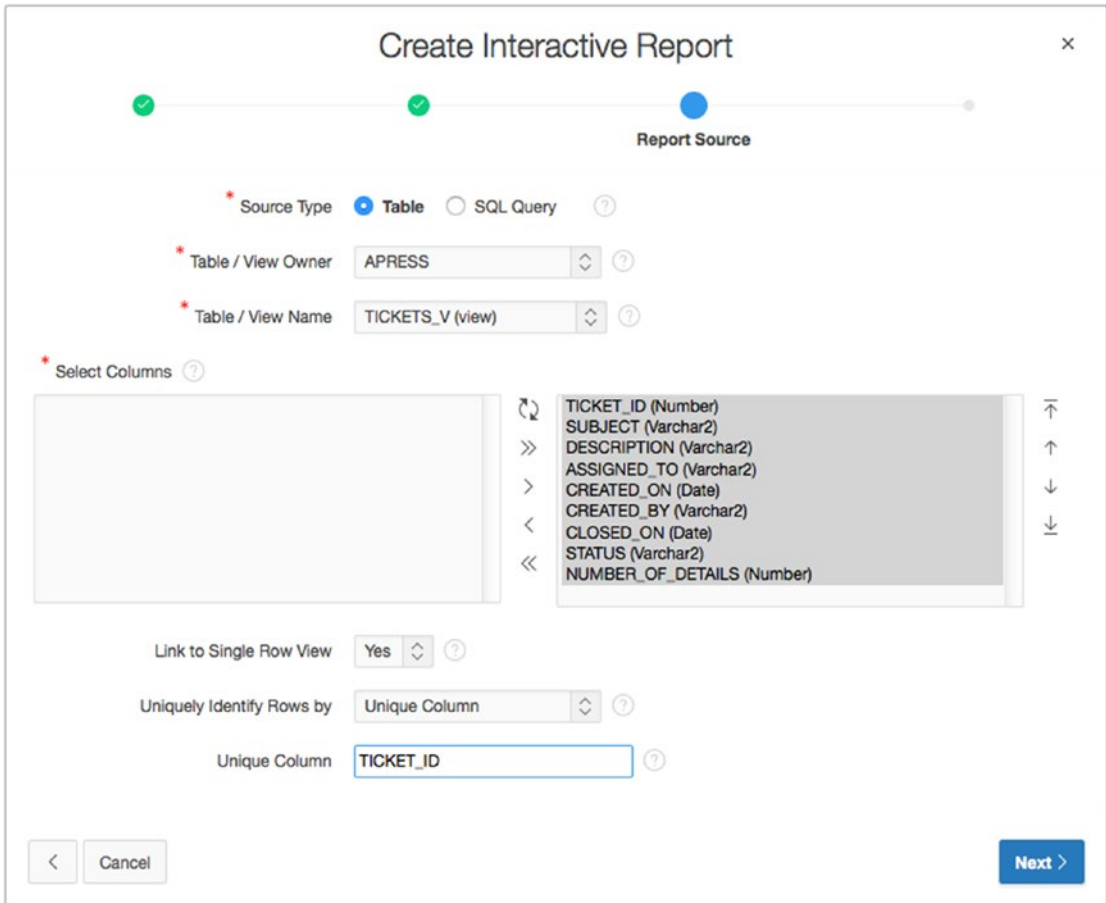
Figure 7-14. *Entering a SQL SELECT statement for an interactive report*

9. Click **Create**.

Running an Interactive Report

Run the application and navigate to the Analysis menu item. The page looks similar to that shown in Figure 7-15. At first glance, the interactive report looks no different than any other APEX report. However, the interactive report can perform a number of functions that a standard APEX report can't.

Figure 7-15. *Interactive report for tickets analysis*

The interactive report has a built-in Search Bar, which is command central for the interactive report. All of the end-user features are accessed through the Search Bar, which is located on the top of the interactive report, in the standard location for a report search field. But this is so much more than just a search field! The Search Bar includes the following:

- *Finder drop-down*: Represented by the magnifying glass, this feature allows the user to select which column to filter on.

- *Search field*: A search field where the user can enter and find text strings.

- *Report select list*: A select list of all saved reports. This select list is visible only when more than one saved report is available. We'll talk about saved reports in a moment.

- *Rows-per-page selector*: A select list of number-of-rows options. This function is turned off by default, because it's also available from within the Actions menu.

- *Actions menu*: A menu of actions enabled for this report—the "interactive" options of the interactive report.

To use the search field, type a string or phrase into it and click the Go button. The interactive report lists only results that match values you entered in the search field.

To use the Finder drop-down, click the arrow next to the magnifying glass icon to the left of the Search Bar. This action opens a menu of the report's column names. Selecting a column name causes the search to be performed on the selected column only.

To use the Report select list, select one of the Report list options to navigate to the selected report. To use the rows-per-page selector, select the desired number of rows per page to display from the select list.

To use the Actions menu, click it to expand the menu of interactive reports actions, and then select the desired action.

Restricting Functionality by Report

As the developer, you have control over which options on the Actions menu are available to the end user by setting options at the report level in the Page Builder. You can also control which of the preceding components are included on the Search Bar. The Search Bar options, shown in Figure 7-16, allow you to include the Search Bar or not and to elect which elements of the Search Bar are visible to the user. This controls end-user functionality at the report level.

Figure 7-16. *Specifying Search Bar options*

The Actions Menu toggles, shown in Figure 7-17, allow you to specify which Actions menu options are available to the user. Of these, the Save Report, Save Public Report, and Subscription options are only available to authenticated users. This is because APEX needs to know information about the authenticated user to be able to save reports and send subscriptions.

Figure 7-17. *Specifying the options for the Actions menu*

Restricting Functionality by Column

Specific interactive report actions can also be restricted on a column-by-column basis. For example, you can allow the report to be filtered, but not allow a specific column to be used in a filter. By editing report columns in the Page Builder, you can declaratively enable or disable the hide, sort, filter, highlight, control break, aggregate, compute, chart, group by, and pivot at the column level through the individual column's report attributes page, as part of the Enable Users To attribute group, as shown in Figure 7-18.

Figure 7-18. *Specifying individual column options*

You've examined the interactive report settings available to you as a developer at the report level and at the column level. Now, let's take a look at interactive report features from the end-user perspective. The following sections examine using the key features of an interactive report as an end user.

Using the Column Heading Menu

When running an interactive report, the column headings contain functionality all their own and are perhaps the fastest way to format a single column of a report. Figure 7-19 illustrates the interactive report column-heading features. Clicking a column heading opens a column-level menu with icon-driven options for quick sorting, removing the column from the report, adding a break on the column, searching, and filtering on the selected column. The Search Bar in this menu allows the end user to search for and filter directly on the values in that column. The Remove Column option lets the user quickly remove the column from the report. To restore the column, the user must choose the Select Columns option of the Actions menu. The Break option adds a break on the column.
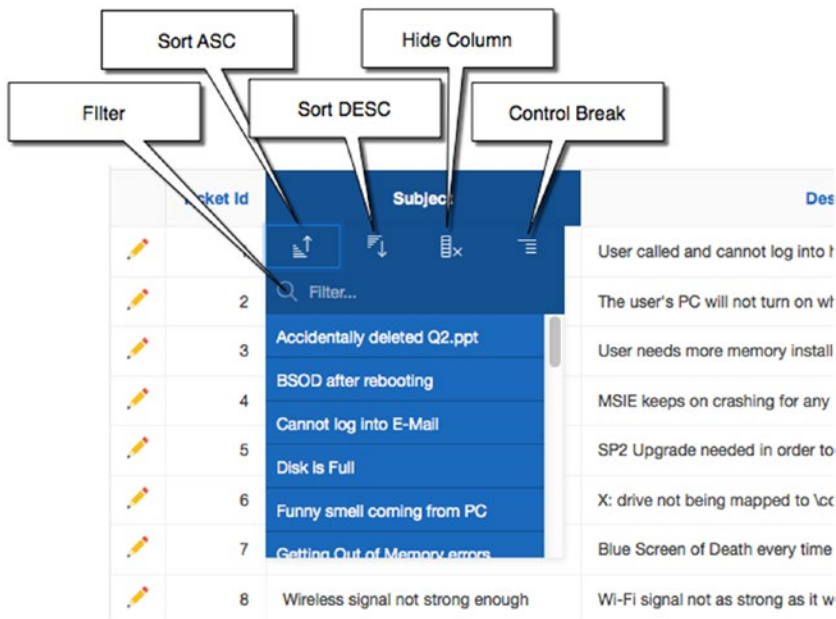
Figure 7-19. *Using the column-heading menu*

If you look below the Filter text field, you will see a full list of distinct values that occur in the column. Clicking any of these distinct values creates a filter on the column, showing only those rows that match the selected value.

Searching by Column

The magnifying glass icon at the left end of the Search Bar is actually a list of the visible columns in the report, which is helpful as a quick way to filter either on a specific column or on all columns. The selected column is the column to which the search text applies.

Entering a value in the search field applies a filter to either all columns (the default) or the selected column. Once a filter is applied, an option appears in the Control Summary region, as shown in Figure 7-20. The Control Summary region is the area between the Search Bar and your report. This region appears only when an action is applied to the interactive report and serves as a key to what actions are currently being applied. The Control Summary region contains one line for each action applied. Interactive report actions are additive: subsequent actions are applied in addition to the existing actions. The user can disable an action by unchecking its check box. The user can remove the action by clicking the × icon for that action. Clicking an action in the Control Summary region opens that action control for editing.
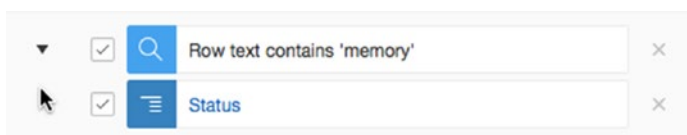

Figure 7-20. *Control Summary region when open*

The Control Summary panel can be toggled open or closed. You can minimize it by clicking the Close (downward-pointing triangle) icon.

The closed Control Summary region, shown in Figure 7-21, can be expanded by clicking the Open (rightward-pointing triangle) icon.
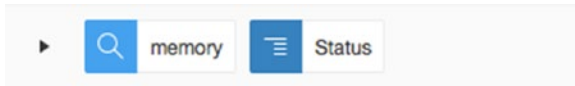

Figure 7-21. *Control Summary region when closed*

The Finder drop-down menu, accessible via the magnifying glass icon to the left of the Search Bar, displays a list of all columns in the interactive report, as shown in Figure 7-22. Selecting one of the columns limits the search function to that column.
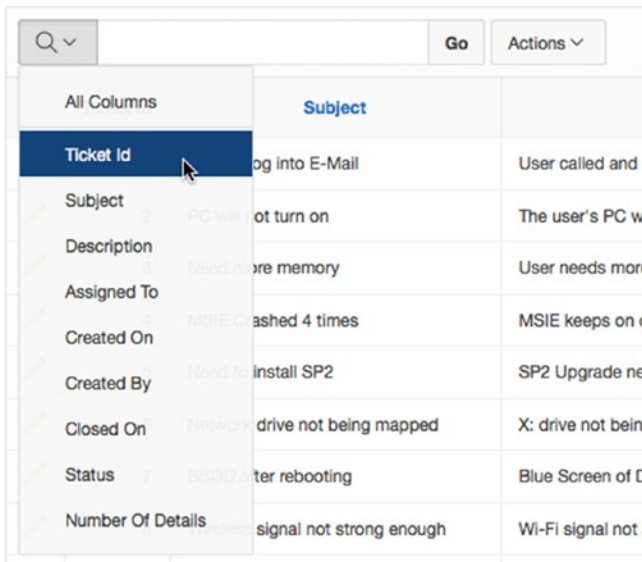

Figure 7-22. *Finder drop-down menu*

The Actions menu, shown in Figure 7-23, exposes an array of column-selection, filtering, and action options. Expanding the menu further under the Format option reveals additional actions for sorting, breaking, highlighting, computing new columns, aggregating, charting, and grouping. The expanded Format menu is shown in Figure 7-24.
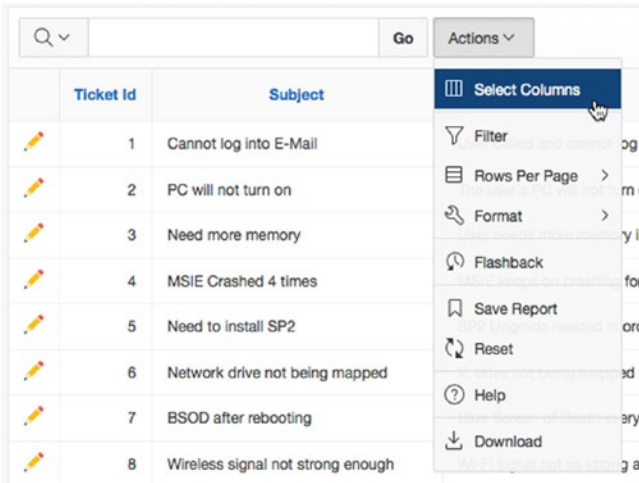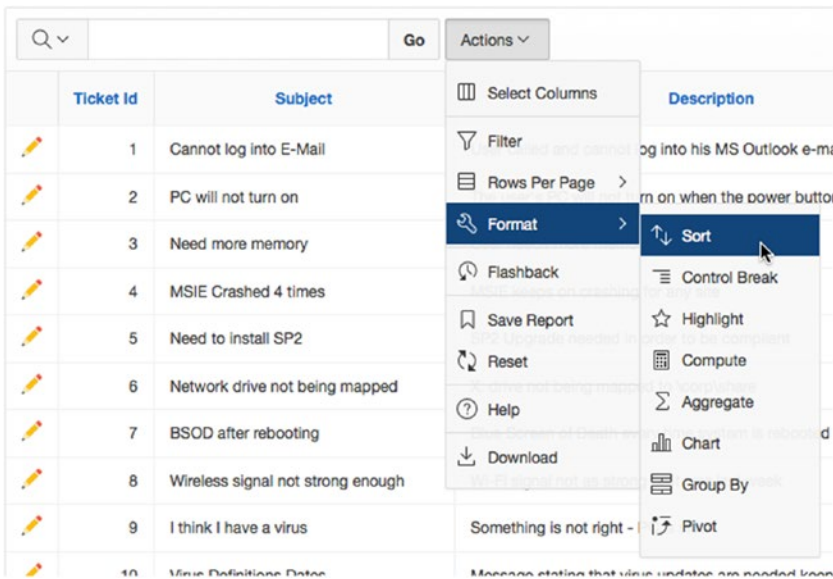

Figure 7-23. *Actions menu*

Figure 7-24. *Choosing a format option from the Actions menu*

Selecting Columns

The Select Columns action, shown in Figure 7-25, allows the user to select which columns to display and to reorder columns as desired. The shuttle control allows the user to easily add or remove columns using the center arrows and to order the columns that are displayed by using the up and down buttons to the right of the region.
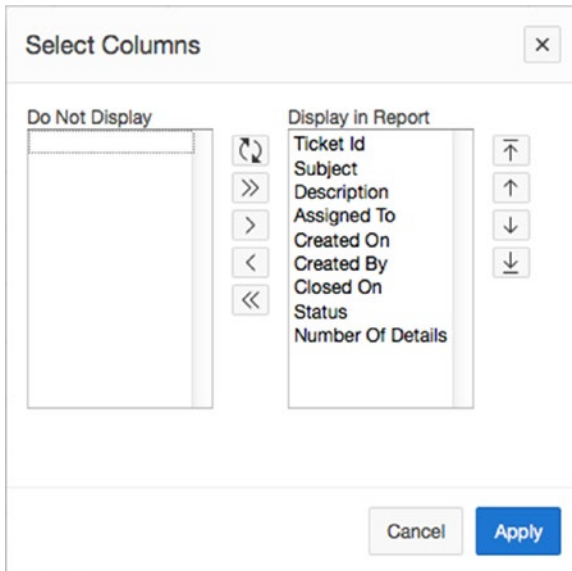

Figure 7-25. *Selecting columns*

■ **Note** The Select Columns action of an interactive report always controls which columns are displayed. If, as a developer, you modify the SQL query to add a column to an interactive report, that new column won't be visible until the new column is moved from the Do Not Display region to the Display in Report region of the shuttle.

Filtering

The Filter action allows the user to declaratively define filters based on the result of a number of operators. A user can define multiple filters per report. Multiple filters are combined with the logical AND operator. Filters defined through the Search Bar are combined with filters defined in the Filter action. Currently, there is no provision in interactive reports to implement a logical OR for filters.

The Filter action offers a full set of filter operations for selection, as shown in Figure 7-26.
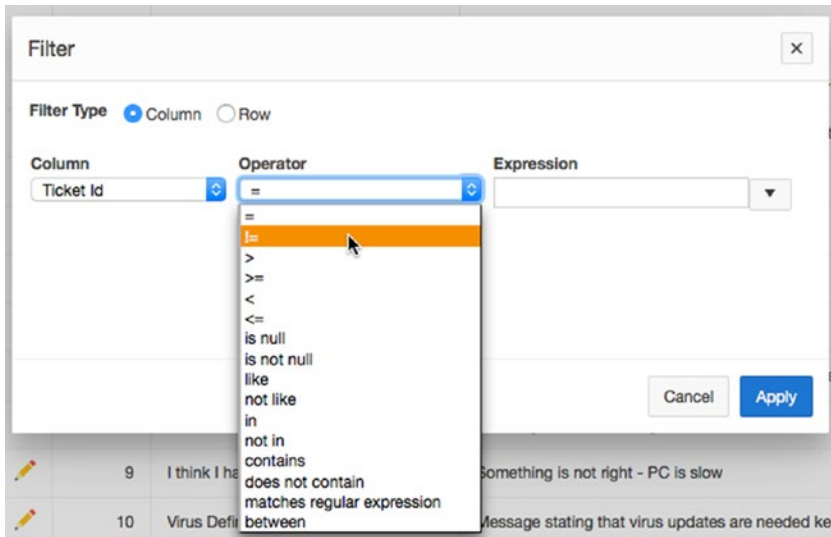
Figure 7-26. *Applying a filter to an interactive report*

The Filter action supports both column filters and row filters. Column filters are applied to a single column. The column filter options change interactively, depending on the type of the filtered column and the selected operator. For example, if you select a date column, such as Created On, and then select the Between operation, the Expression element now contains two fields, for the From and To of the between clause. In this case, the fields each have a date picker for ease in entering the Date From and To values. The end user can also construct a custom filter using the declarative Filter.

Row filters allow the user to build filter conditions that are based on multiple columns in the same row. A simple row filter for your Analysis report might be a filter for all tickets that were closed on the same day they were opened. The Filter expression may be built declaratively using selections in the Columns and Functions/Operators regions, shown in Figure 7-27, or may be entered manually. Within the Filter expression, selected columns are represented by their letter alias.
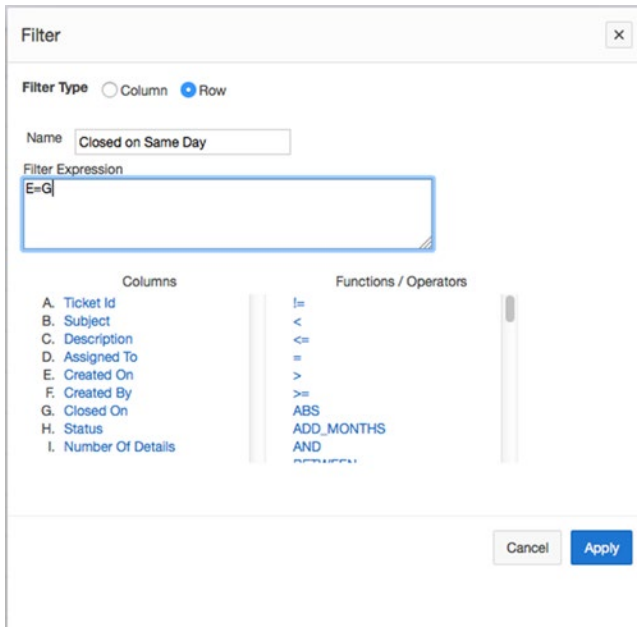

Figure 7-27. *Building a row filter*

Sorting

The Sort interface allows the user to specify sorts on up to six columns in either ascending or descending order and to specify whether NULLs are sorted first or last. The sort may be performed on both displayed and non-displayed columns (see Figure 7-28).

Figure 7-28. *Adding sorts to an interactive report*

Adding Breaks

The Control Break action allows the user to define break formatting on up to six columns. The user specifies the break column and whether the break is disabled or enabled. APEX automatically applies the declared break formats to the report. Note that break columns appear in the Control Summary as separate entries, letting the user enable, disable, or remove break columns individually. Figure 7-29 shows the Analysis report with breaks applied on the Assigned To and Status columns.

Figure 7-29. *Interactive report with control breaks applied*

Highlighting

The Highlight action allows the user to find matching data and highlight it by row or column, specifying the background and text colors for the highlight. The Highlight action interface is shown in Figure 7-30.

Figure 7-30. *Adding highlighting with the Highlight action*

The same operators that you saw in the Filter action apply here. The background and text colors may be specified using either hex notation or the color palettes. The Highlight action appears in the Control Summary region as a highlighted row.

Computing Columns

The user can define a new column as a computation based on existing columns and functions via the Compute action interface, shown in Figure 7-31.

Figure 7-31. *Computing a new interactive report column using the Compute action*

The user may either declaratively or manually define the computed value. The declarative interface is much the same as the row filter interface. Columns are specified in the computation as their letter aliases. This option is quite powerful, because it allows the end user to build essentially any column they desire.

Adding Aggregates

The Aggregate action performs one of the following aggregation functions on a column:

- Sum

- Average

- Count

- Count Distinct

- Minimum

- Maximum

- Median

The selected column must be of data type `NUMBER`. The results are displayed at the end of the report. Note that aggregate results are displayed only if the corresponding column is also displayed.

Adding Charts to Interactive Reports

The Chart action allows the user to display a dynamic Flash chart representation of the data in the report, as shown in Figure 7-32. The chart representation of the data is displayed instead of the tabular data representation. The display can be toggled by clicking the View Chart icon, as indicated in Figure 7-32. Use the Edit Chart link to reenter the Chart action interface.

Figure 7-32. *Interactive report pie chart*

The following chart types are supported in an interactive report:

- Horizontal bar

- Vertical bar

- Pie

- Line

The simple Chart action interface, shown in Figure 7-33, allows the user to select the chart type and assign a label column, a value column, a function, and a column to sort by.
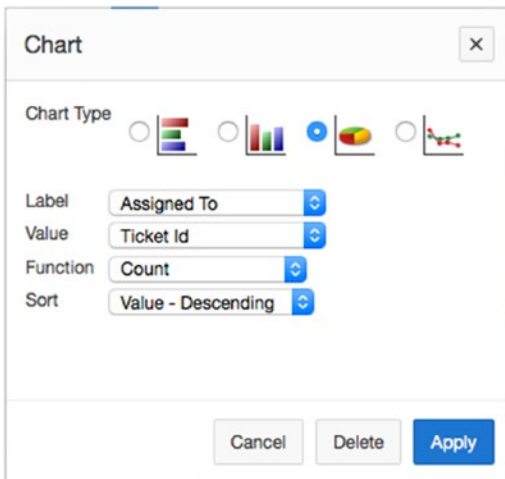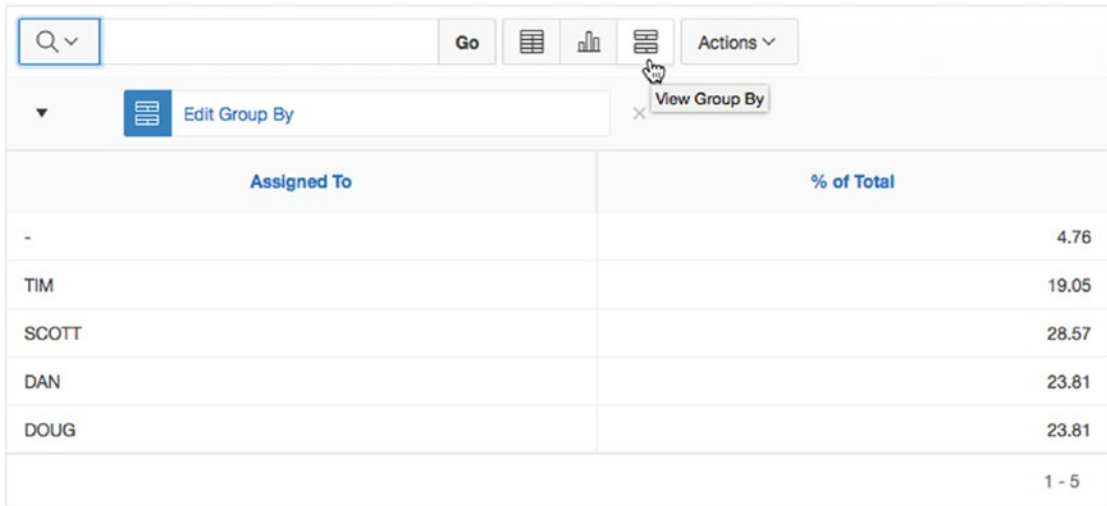
Figure 7-33. *Adding a chart using the Chart action*

The user doesn't have the full functionality of APEX charts within the Chart action, but the ease of displaying these most common chart types is quite valuable.

Grouping

The Group By action allows the user to define groups and then aggregate functions on those groups, thus letting the user declaratively define their own summary views of the report data. A sample result of using the Group By action is shown in Figure 7-34.

Assigned To	% of Total
-	4.76
TIM	19.05
SCOTT	28.57
DAN	23.81
DOUG	23.81
	1 - 5

Figure 7-34. *Grouping using the Group By action*

Like the Chart view, the Group By view of the data has a display icon in the center of the Search Bar, as indicated in Figure 7-34. The user may display the data view, the Group By view, or, if defined, the Chart view of the data by clicking the appropriate display icon.

Pivot

The Pivot action allows the user to define a pivot view of the data in the report, giving the user full control over the columns to pivot, the columns to display as rows, and the columns to aggregate with one of the available aggregate functions. A sample of the settings and the resulting Pivot report are shown in Figure 7-35.

Figure 7-35. *A Pivot report generated using the Pivot action*

197

Using Flashback

The Flashback action enables the user to flash back the database by the specified number of minutes to see what the data looked like at that point in time. The option is built on the Oracle database FLASHBACK feature. Database FLASHBACK must be enabled. The Flashback action asks for the number of minutes to flash back, as shown in Figure 7-36.
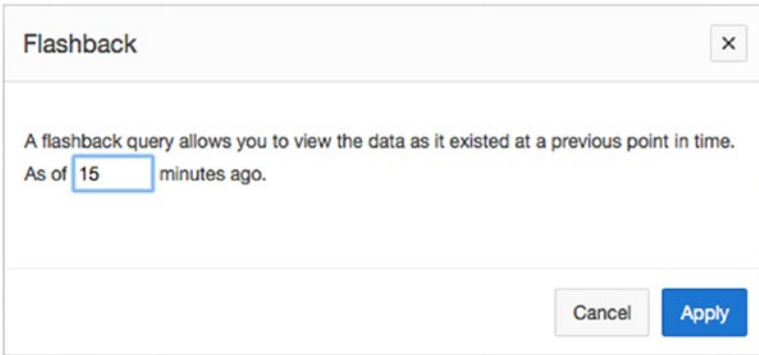
Figure 7-36. *Using the Flashback action*

The length of flashback time is configurable. The maximum flashback period is based on the UNDO_ RETENTION parameter in the database, which is set to three hours by default.

Saving an Interactive Report

The Save Report action allows the user to save the current configuration of the interactive report as a named report. If the end user is also an APEX developer, the user will see the Save As Default Report Settings option, shown in Figure 7-37.
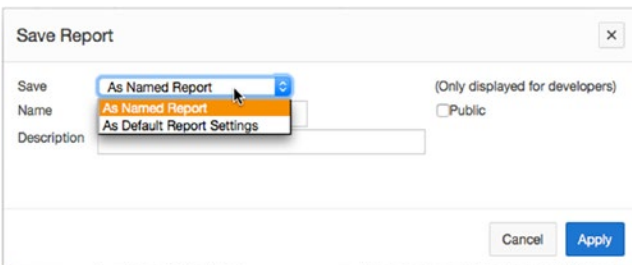
Figure 7-37. *Saving an interactive report using the Save Report action*

As a developer, you want to try to pre-create the versions of the report that you feel will be the most widely used by the largest subsection of users. You may save the current report configuration as being either the primary or the alternative default report settings, as shown in Figure 7-38. The primary report is the one that any brand-new user sees by default when logging on to the system. If alternative default reports exist, the user is able to choose them from the select list.

Figure 7-38. *Setting an alternative saved report*

Obviously you can't pre-create every possible iteration of a report. Therefore, the user may save reports as private reports. When a report is saved, it's added to the Reports menu in the Search Bar, as shown in Figure 7-39.

Figure 7-39. *Using the default Reports menu*

Resetting an Interactive Report

The Reset action, shown in Figure 7-40, restores the current report to the default settings. Any changes in formation or result set (by filtering) are lost, unless, of course, the report is a saved report. It may then be reinstated simply by selecting the report name from the select list.
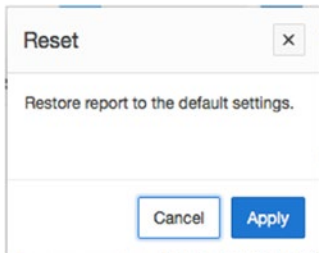
Figure 7-40. *Resetting an interactive report to its default settings*

Getting Help

The Help action opens a window that contains interactive report–specific help, as shown in Figure 7-41. All of the interactive report options are displayed in this Help window, regardless of whether they're enabled for the current report.
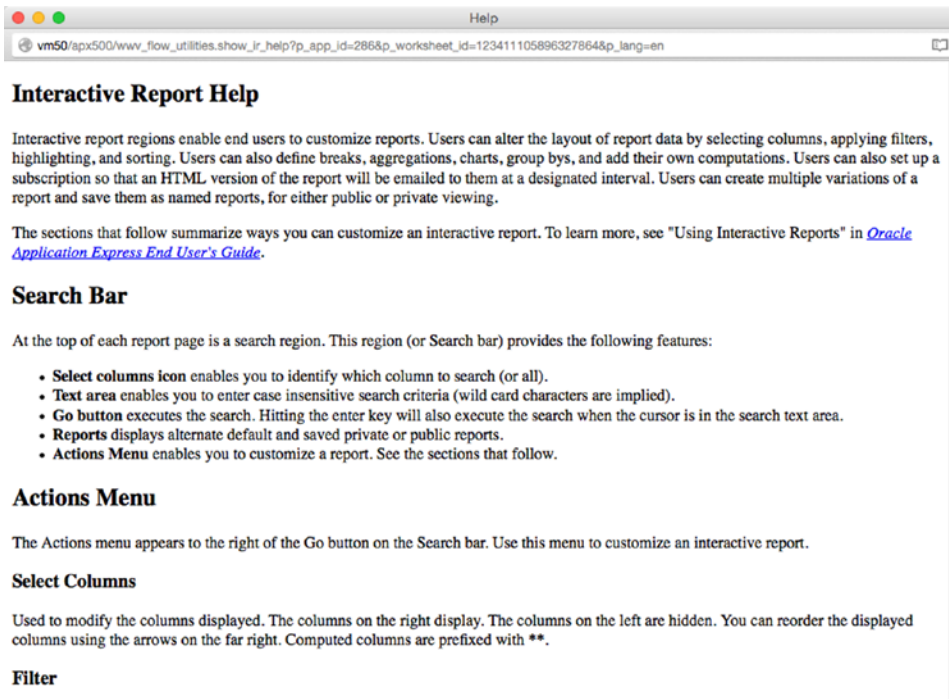
Figure 7-41. *The Interactive Report Help page*

Adding a Subscription

The Subscription action allows the user to email a report to designated email addresses on a scheduled basis. The user enters the email address, subject, frequency, and start and end dates, as shown in Figure 7-42. This action is available for authenticated users only. The email received is a searchable HTML version of your report. Break formatting and highlighting aren't preserved.

Figure 7-42. *Subscribing to an interactive report*

If a subscription for the current user is in effect, you can edit that subscription by using the Subscription action again. The form then presents the current subscription attributes and allows the user to either change or delete the subscription. The interface is exactly like that shown in Figure 7-42, the only addition being a Delete button.

Report subscriptions can also be managed by a Workspace Administrator through the Administration Home Page ➤ Tasks Menu ➤ Interactive Report Settings ➤ Subscriptions interface, as shown in Figure 7-43.

Figure 7-43. *Managing subscriptions through the manage-subscriptions interface*

Downloading

The Download action allows the user to download the current result set of their report in one of the following formats:

- CSV

- HTML

- Email

- PDF

- XLS (MS Excel)

- RTF (MS Word)

The latter two formats require Oracle BI Publisher, which may require a separate license from Oracle. The email option will only be available if your APEX administrator has configured APEX to integrate with an external email server. Figure 7-44 shows the download options without and with BI Publisher. You can specify which formats are available in the Download attributes region, as shown in Figure 7-45.
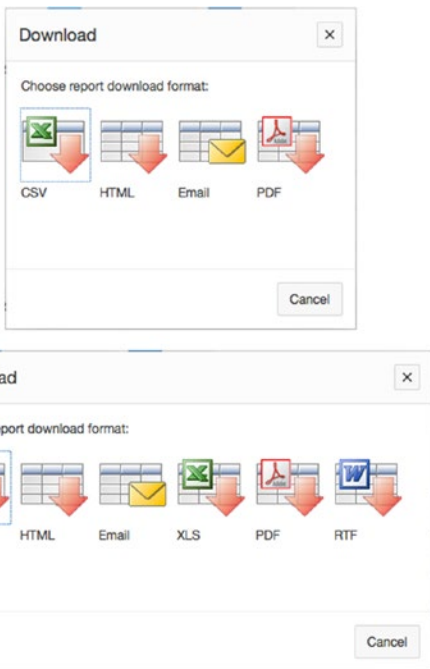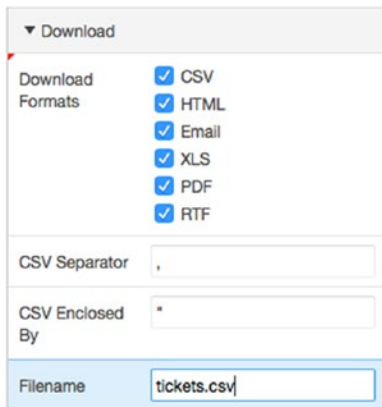
Figure 7-44. *Choosing download options, without and with BI Publisher*

Figure 7-45. *Specifying download attributes*

Reports downloaded in CSV format are plain, comma-delimited data. The content and order of data in the result set are retained in the CSV file, but break formatting and highlighting aren't.

Reports downloaded in HTML format are a searchable HTML version of the result set, as shown in Figure 7-46. Again, the result set content is preserved, but the break formatting and highlighting aren't.

Figure 7-46. *The searchable HTML download of an interactive report*

203

The email download is the same output as the HTML download, but delivered in an email. The XLS and RTF download formats require integration with Oracle BI Publisher, which may require a separate license from Oracle. The PDF output can be accomplished via either Oracle Rest Data Services, an external Formatting Objects Processor (FOP), or BI Publisher. A complete description of the use of Oracle BI Publisher to produce reports in these formats is beyond the scope of this book. See the Oracle APEX documentation section "Advanced Printing Options and Configuration" for more details. If these options aren't configured for your installation, they won't appear in the download options list.

Take some time to experiment with the features of the interactive report. If you get lost and need to start over, simply click the Actions button and select Reset. The interactive report will be reset to its original state, and all modifications that you made to it will be discarded.

Modifying an Interactive Report

Although an interactive report offers a tremendous amount of functionality, you may wish to limit which features are available to your end users. Each feature of the interactive report can be disabled on a report-by-report basis. In addition, you can set up default options for a specific report, making those available to all end users.

Adding Attributes and Removing Columns

Let's take another look at your interactive report. You can use a combination of interactive report end-user actions and developer settings to achieve modifications. First, remove a column from the report and add a sort attribute using the Actions menu:

1. Run the application and navigate to the **Analysis** menu item.

2. Click the **Actions** button to display the Actions menu.

3. Select the **Select Columns** option, as shown in Figure 7-47.
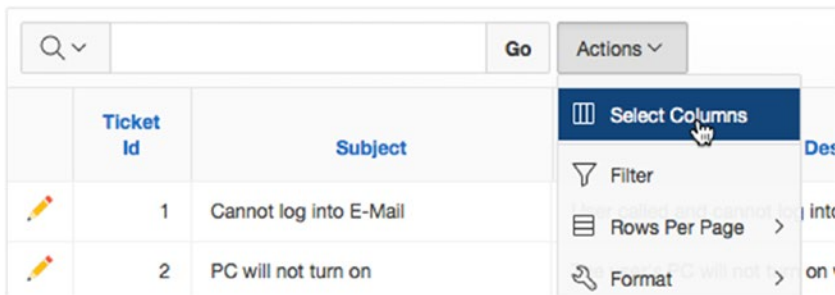
Figure 7-47. *Selecting the Select Columns option*

4. Move **Ticket Id** to the <u>**Do Not Display**</u> section of the shuttle, as shown in Figure 7-48, by double-clicking its name.

5. Using the up and down arrows, reorder the remaining columns so that **Status** appears after **Subject** and before **Description**, as shown in the Display in Report section in Figure 7-48, and click **Apply**.
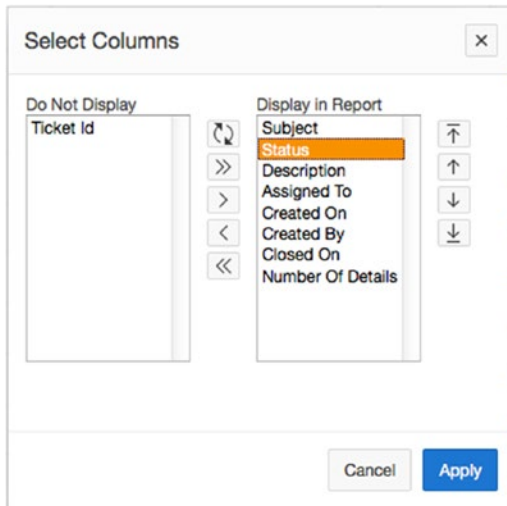
Figure 7-48. *Selecting columns*

Notice that the Ticket Id column is no longer displayed in your report and that the Status column appears immediately after the Subject column.

Next, you can set your changes as default options for the interactive report. These options will be applied for all end users who use the interactive report. The Save As Default Report Settings option is only available to end users who are APEX developers:

6. Click the **Actions** button and select the **Save Report** item.

7.   Set **<u>Save</u>** to **As Default Report Settings**, as shown in Figure 7-49.
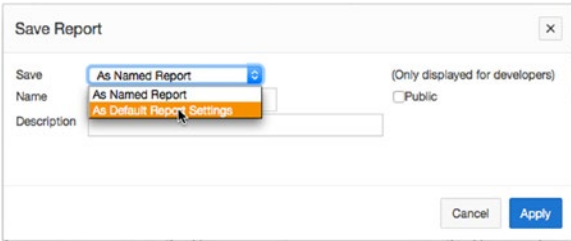
Figure 7-49.   *The Save As Default Report setting*

8.   The region immediately changes, allowing you to save the report either as the primary default or as a named alternative. Make this one the **Primary** default, as shown in Figure 7-50. Click **Apply**.
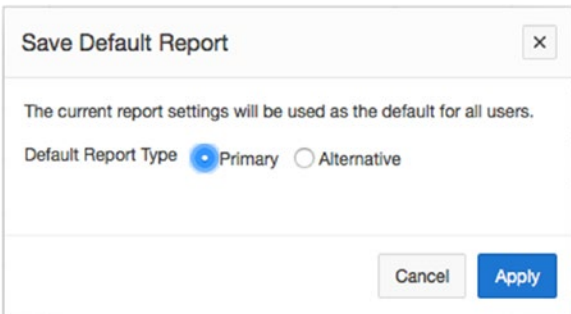
Figure 7-50.   *Saving a primary interactive report*

Now, create a named alternative default report that does a control break on the Status column:

9.   Click the **Actions** button and navigate to **Format ➤ Control Break**, as shown in Figure 7-51.
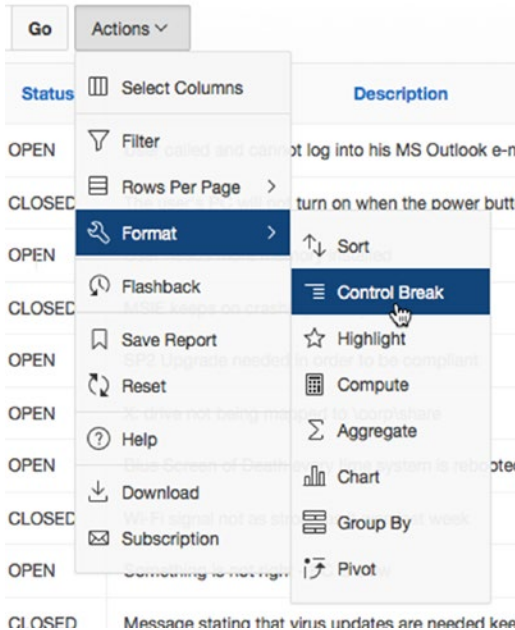
Figure 7-51. *Selecting the Control Break action*

10.   Select **Status** in the first **Column** select list and make sure it's set to **Enabled**, as shown in Figure 7-52. Click **Apply**.
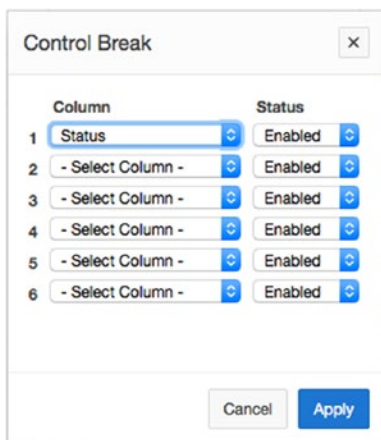
Figure 7-52. *Applying a control break to an interactive report*

11. Click the **Actions** button and select the **Save Report** option.

12. Set <u>Save</u> to **As Default Report Settings**, as shown in Figure 7-53.
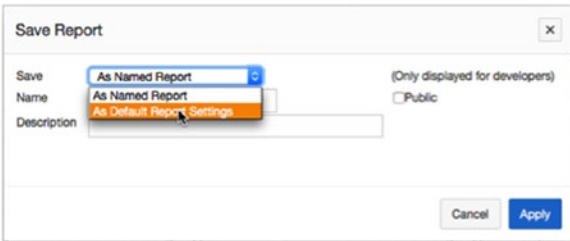
Figure 7-53. *Saving an interactive report as a default setting*

13. The region immediately changes. This time, save the report as a named alternative: select **Alternative** for <u>Default Report Type</u>, enter Tickets by Status for <u>Name</u>, as shown in Figure 7-54, and click **Apply**.
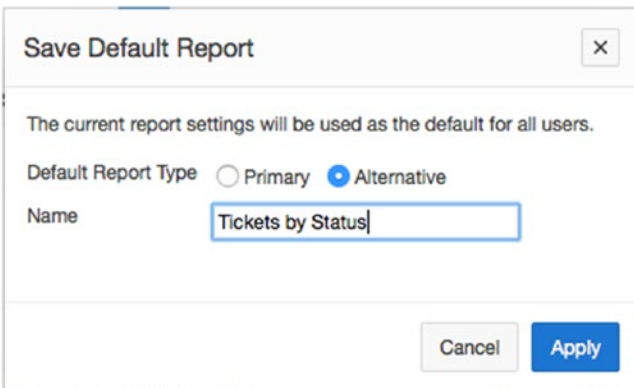
Figure 7-54. *Saving on interactive report as an alternate report*

The toolbar at the top of the report now has a new Reports select list that contains both your default and alternative reports, as shown in Figure 7-55.
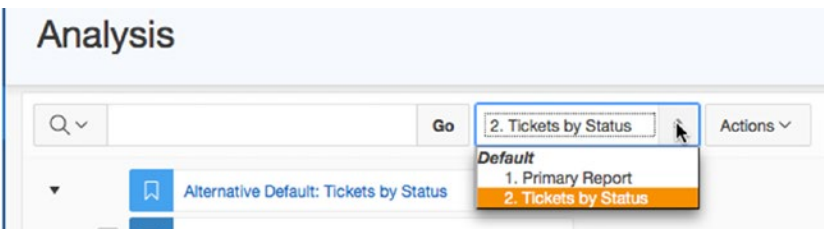
Figure 7-55. *Reports select list showing both the primary and named alternative reports*

Selectively Enabling and Disabling Items

As a developer, you can selectively enable or disable items from the Actions menu. Doing so restricts which options are available to the end user for a specific interactive report. Here's an example to work through:

1. Edit **Page 300** of the application.

2. Edit the **Analysis** report's interactive report properties by clicking the **Attributes** node in the **Rendering** tree.

3. Scroll down to the **Actions Menu** attributes group. Set **Flashback** and **Save Report** to **NO** and set **Subscription** to **YES**, as shown in Figure 7-56.

Figure 7-56. *Selecting Actions menu options*

4. **Save** your changes.

Run your report again, and then click the Actions button to expand the actions menu. Notice that the Flashback item is no longer present. And while we turned Save Report off, you can still save reports because you're logged in as a developer to the underlying workspace. Standard end users won't see this option. You should also see a new option for subscriptions.

Limiting an Action to Specific Columns

In addition to controlling which actions appear for an interactive report, you can get even more granular and determine on which columns a specific action can be performed. Figure 7-57 shows the column-level Actions settings for your interactive report. Proceed as follows:

1. Edit **Page 300** of your application.

2. Expand the **Columns** node under the **Analysis** Interactive Report node in the **Rendering** tree and select the **Description** column.

3. In the **Enable Users To** attribute group, set the **Sort** and **Filter** attributes to **NO**.

4. **Save** your changes.

5. Run **Page 300** of your application.

Figure 7-57. *Selecting column-level actions for an interactive report*

6. Click the **Actions** button and select **Format ➤ Sort**. The Sort action interface should look similar to Figure 7-58.
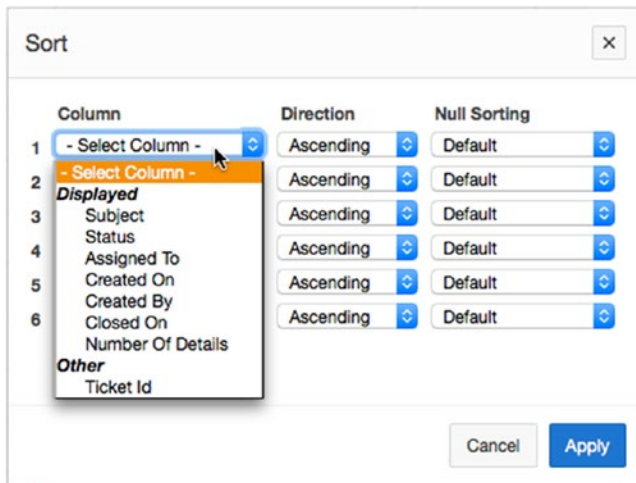
Figure 7-58. *Modified Select Column list*

7. Notice that **Description** no longer appears as a column name in the list of columns.

By default, the interactive report links to something called Single Row View. This view shows a read-only region that contains all the details about a specific row. In this case, you may want to link back to the form you created on page 210. In this case, you can alter the interactive report to use a more traditional page link instead of the Single Row View. You do this by editing the Link Column attributes, as shown in Figure 7-59:

8. Edit **Page 300** of the application.

9. Click on the **Attributes** node for the **Analysis** Interactive Report.

10. In the **Link** attribute group, set **Link Column** to **Link to Custom Target**.

11. Click on the **Options Dialog** button for the **Target** attribute.

12. In the resulting dialog, make sure **Type** is set to **Page in This Application**, set **Page** to 210, enter P210_TICKET_ID for the **Name** and #TICKET_ID# for the **Value** on the first line, and then click **OK** (see Figure 7-59).

Figure 7-59. *Setting the Link Column attributes*

13. **Save** your changes.

Name and Value tell the link to pass the current ticket's ID (identified by #TICKET_ID#) and assign it to P210_TICKET_ID in session state.

Run page 300 of your application. You should now be able to drill into the details of any row by clicking in the column with the Edit link.

Looking Behind the Scenes

Let's look behind the scenes of the interactive report. You may be surprised to see that there is only a single interactive report region in the Rendering tab, as shown in Figure 7-60.
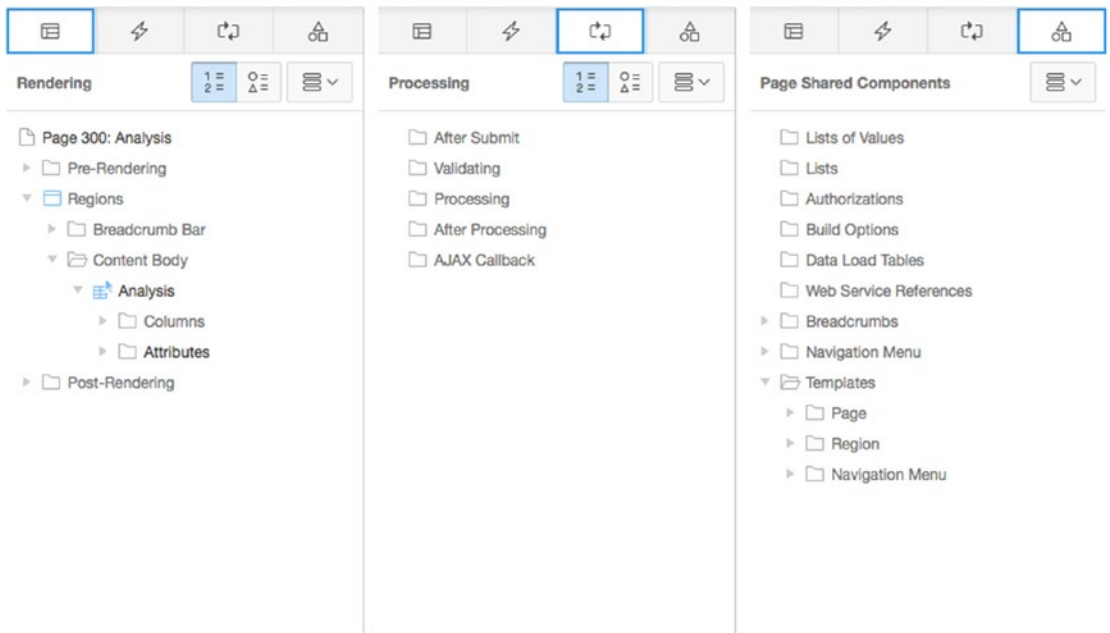
Figure 7-60. *The Application Builder view of the interactive report*

The Processing tab contains no elements, and the Shared Components tab contains only the expected elements.

This is the first case where you can't easily re-create the interactive report using standard declarative APEX elements. The additional functionality is from a collection of JavaScript functions, CSS, and HTML that are all contained within the interactive report region type. Although you could build this from scratch, the APEX interactive report is a huge timesaver.

Calendars

Sometimes there are trends in data that aren't obvious when viewed in the traditional row/column format. By simply displaying data in a different way, such as in a calendar report, trends can become obvious. The APEX calendar report can display data in a daily, weekly, or monthly view and doesn't require that you enter any SQL.

Understanding Calendar Types

An APEX calendar is a type of APEX report. Data is rendered on a calendar instead of in a traditional row/column format. The single requirement for an APEX calendar is that the underlying table or view must have at least one DATE column.

There are two types of APEX calendars:

- *Calendar*: This is based on an open-source jQuery component and gives quite a lot of functionality out of the box.

- *Legacy Calendar*: This uses the calendar region type that was available up through APEX 4.2 and has less functionality than the new jQuery-based calendar.

Data in a calendar can act as a column link, the same as in any other report column. This makes it simple to build a calendar that lets the user click a date and drill to another page or URL.

Creating a Calendar

To implement an APEX calendar, you can create a new page and a Calendar region using the Create Page wizard. Here are the steps to follow:

1. Navigate to the Application Builder Home Page for your application.

2. Click the **Create Page** button in the upper right of the screen.

3. Select **Calendar** and click **Next**.

4. Again, select **Calendar** and click **Next**.

5. As shown in Figure 7-61, enter 400 for **Page Number** and Ticket Activity Calendar for both **Page Name** and **Region Name**, and set **Breadcrumb** to **Breadcrumb**.

Figure 7-61. *Creating a ticket activity calendar*

6. When the page reloads, enter Ticket Activity Calendar for **Entry Name** and click **Next** (see Figure 7-62).

Figure 7-62. *Specifying the breadcrumb entry for the calendar*

7. Set **Navigation Preference** to **Create a new navigation menu entry**. When the page refreshes, enter Calendar for **New Navigation Menu Entry** and click **Next** (Figure 7-63).

Figure 7-63. *Specifying tabs for a calendar*

8. Make sure the **<u>Source Type</u>** is set to **Table**, select your schema as the **<u>Table/View Owner</u>**, select **TICKETS_V (view)** for **<u>Table/View Name</u>**, as shown in Figure 7-64, and click **Next**.

Figure 7-64. *Specifying the table owner and table name for a calendar*

The next step in the wizard allows you to choose what is displayed on the calendar, as well as the begin and end dates. Also, on this page you can choose how you want to see the date displayed (Date only or Date and Time).

There are also options that will automatically generate a create page or an edit page, and choose whether or not you want to be able to use drag and drop on your calendar to alter the begin and end dates of an event.

We're going to use Page 210 for our create and edit pages, but we do want to turn Drag & Drop on. Continue as follows:

9. For **Display Column**, select SUBJECT.

10. For **Start Date Column**, select CREATED_ON. For **End Date Column**, select CLOSED_ON.

11. Set **Show Time** to **Yes**.

12. Set **Add Create Page** and **Add Edit Page** to **No**.

13. Set **Generate Drag & Drop Code** to **Yes**.

14. Click **Next**.

15. Set **Primary Key Type** to **Select Primary Key Column(s).**

16. Select TICKET_ID(Number) as the **Primary Key Column**.

17. Click **Next**.

18. Set **Source Type** to **Existing Trigger** and click **Next**.

19. Click **Create**.

Run the Calendar report, and you'll see something similar to Figure 7-65. Hovering over any of the event bars will show a hover-hint with more of the subject and date information displayed. Since we turned on Drag & Drop, you can click and drag any event to change its Created On date.
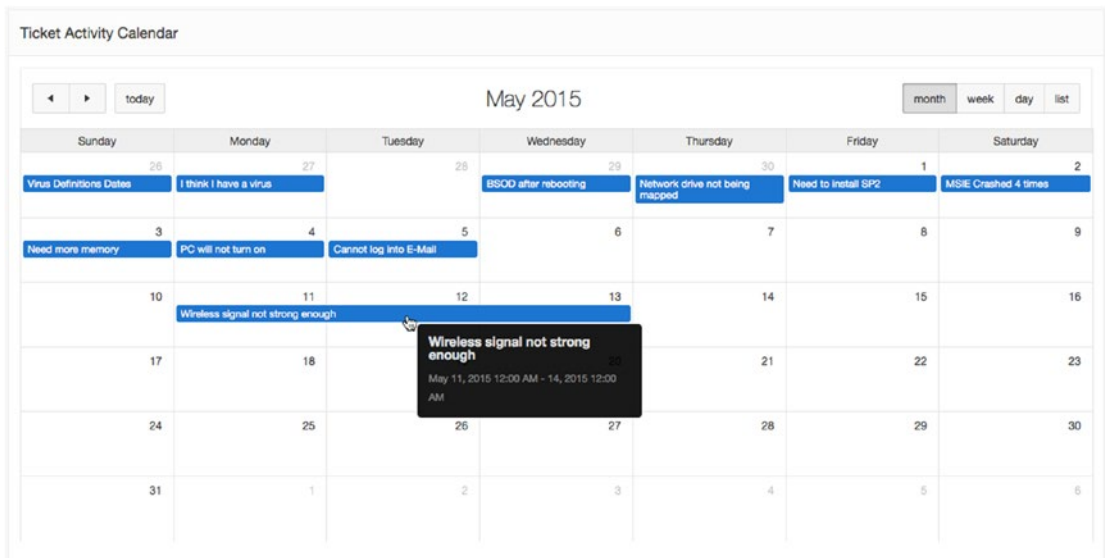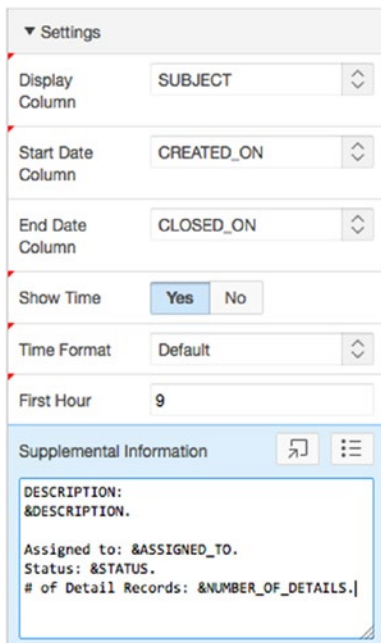
Figure 7-65. *The Calendar report as generated by the wizard*

There are a few tweaks that we can make to the calendar to provide some more information to the viewer and to link it with the Ticket edit screen on Page 210:

1. Edit **Page 400** of the application.

2. Edit the **Attributes** of the **Ticket Activity Calendar**.

3. Locate and open the file ch7_calendar_details.txt, which you can find where you extracted the book files. Copy and paste the contents of that file into the **Supplemental Information** text area under the **Settings** attributes group, as shown in Figure 7-66.

Figure 7-66. *Setting calendar Supplemental Information attribute*

4. Click the Options button next to **<u>View/Edit Link</u>**. The button should say **No Link Defined**.

5. In the resulting pop-up dialog, set the **<u>Page</u>** to 210, set **<u>Name</u>** to P210_TICKET_ID, set **<u>Value</u>** to &TICKET_ID., set **<u>Clear Cache</u>** to 210, as shown in Figure 7-67, and then click **OK. Save** your changes.

Figure 7-67. *Setting the View/Edit Link attributes*

Run the page, and your calendar should look similar to that in Figure 7-68. When you hover over an event bar now, you can see more of the ticket detail. You can now also click the event bars to edit the Ticket details.
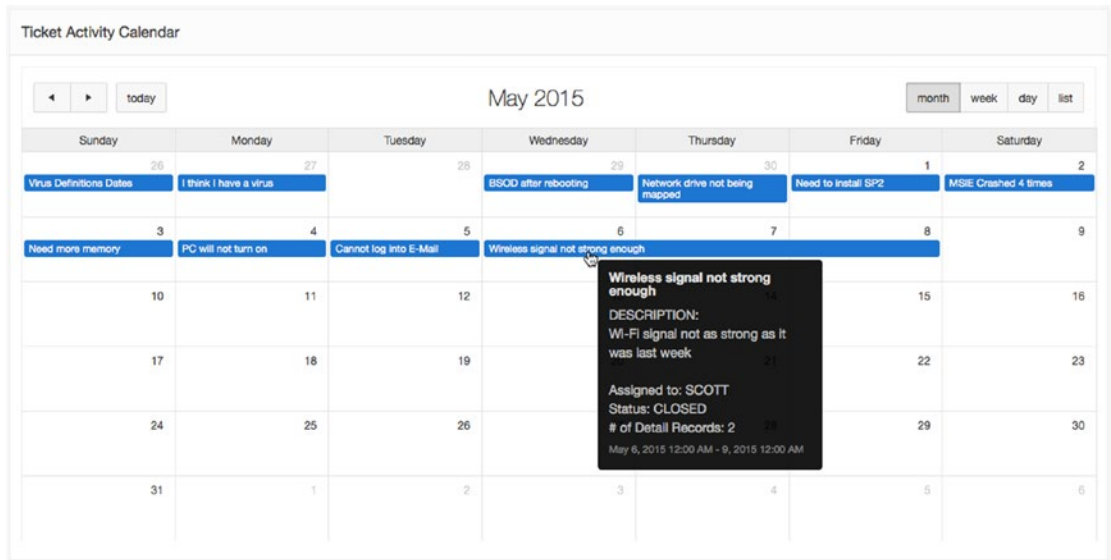
Figure 7-68. *The altered ticket activity calendar*

■ **Note**   If the data being shown in the Supplemental Information may contain special characters such as & ' " or / APEX will attempt to encode them to reduce the risk of Cross-Site scripting. If you trust the source of the data, you can edit the Calendar region, and in the Security attributes section, set "Escape Special Characters" to No.

Looking Behind the Scenes

Now that your calendar works, let's look at what the Calendar wizard built for you. Edit page 400. In the Rendering tab, shown in Figure 7-69, you'll basically see only the Calendar region.
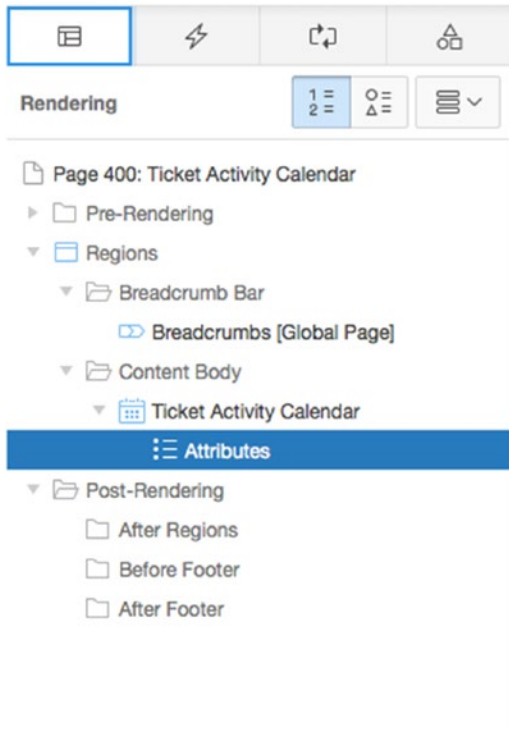
Figure 7-69. *Page Rendering region for your calendar*

Notice that there are no processes in the Processing tab of the page. That is because the Calendar region uses JavaScript calls to get the data from the database as well as to change any values that shift based on the drag and drop action. This is just a brief example of what JavaScript and jQuery can do.

Charts

In APEX 4.2, charts got a major facelift with the incorporation of AnyChart 6, and that facelift has extended into APEX 5.0. Not only does this release of AnyChart produce charts that look much more professional than in previous releases, but the charting engine also provides the option to use either Flash-based or HTML5-based charts. This is a huge leap forward for applications aimed at the mobile market, because HTML5 charts render on most modern browsers with no need for extra plug-ins.

The beauty of the new charting engine is that you can flip between rendering Flash and HTML5 charts at any time during the development of the page, and the declarative data remains the same, regardless of the choice of rendering.

HTML5 charts also maintain the same level of interactivity that Flash charts have, including hover and click-and-drill functionality.

Figure 7-70 shows both the Flash version and the HTML5 version of a bar chart, showing that, although the look changes a certain amount, the same code generates very similar charts.
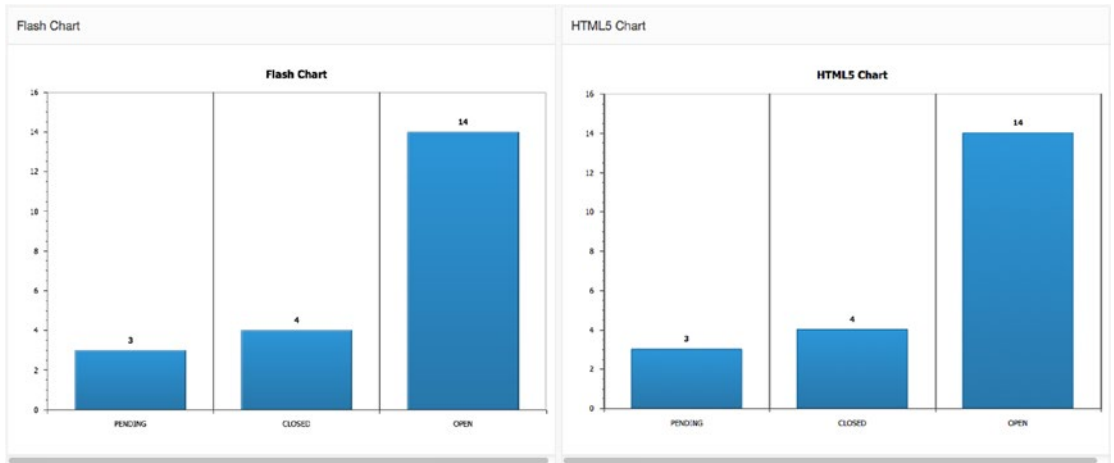
Figure 7-70. *The same chart rendered with Flash and HTML5*

Flash and HTML5 charts have almost identical functionality, but HTML5 charts are able to render natively on any platform, whereas Flash charts will not work on any of the Apple mobile platforms.

Writing Queries for Charts

APEX charts generally need a query of this type:

```
SELECT
  link,
  label,
  value
FROM
  table
WHERE
  where conditions
GROUP BY
  group by column list
ORDER BY
  Order by column list
```

where

- `link` is a link to an APEX page or other URL;

- `label` is the label for the chart element; and

- `value` is the value to be charted.

The exact syntax changes slightly to suit the needs of the various chart types, but the general link-label-value format remains the same. For the correct syntax for each chart type, see the APEX online documentation.

Creating a Chart

Let's create a pie chart that shows the count of tickets in each status. Later, you'll link the action of clicking a pie piece to filtering the tickets report to show only tickets of that status. Follow these steps:

1. Edit any page of the application.

2. Click the **Create (+)** button in the Page Designer toolbar and select **Page**.

3. Select **Chart** and click **Next**.

4. Select **HTML5 Chart** from the select list.

5. When the page refreshes, select **Pie & Doughnut** and click **Next**.

6. Select **2D Pie** and click **Next**.

7. Enter 500 for **Page Number** and `Tickets by Status` for both **Page Name** and **Region Name**, and set **Breadcrumb** to **Breadcrumb** (see Figure 7-71). When the page reloads, enter `Tickets by Status` for **Entry Name** and click **Next**.

Figure 7-71. *Setting the Page Number, Page Name, and Region Name attributes for a chart*

8. Set **Navigation Preference** to **Create a new navigation menu entry**. When the page refreshes, enter Chart for **Tab Label** and click **Next** (see Figure 7-72).

Figure 7-72. *Setting the navigation attributes for a chart*

9. Set **Chart Title** to Ticket Statuses and click **Next**.

10. Locate and open the file ch7_chart_query.txt, which you can find where you extracted the book files. The contents of the file should be similar to this query:

```
SELECT
  'f?p=&APP_ID.:200:' || :APP_SESSION || ':::::P200_STATUS_ID:' || sl.status_id link,
  sl.status label,
  count(*) value
FROM
  tickets t,
  status_lookup sl
WHERE
  t.status_id = sl.status_id
GROUP BY
  sl.status_id, sl.status
ORDER BY
  3 DESC
```

11. Paste the contents of the file ch7_chart_query.txt into the **Enter SQL Query or PL/SQL Function Returning a SQL Query** region, or type the previous query into the region, and click **Next**.

12. Click **Create**.

Run the page. Your chart should look similar to the one in Figure 7-73.
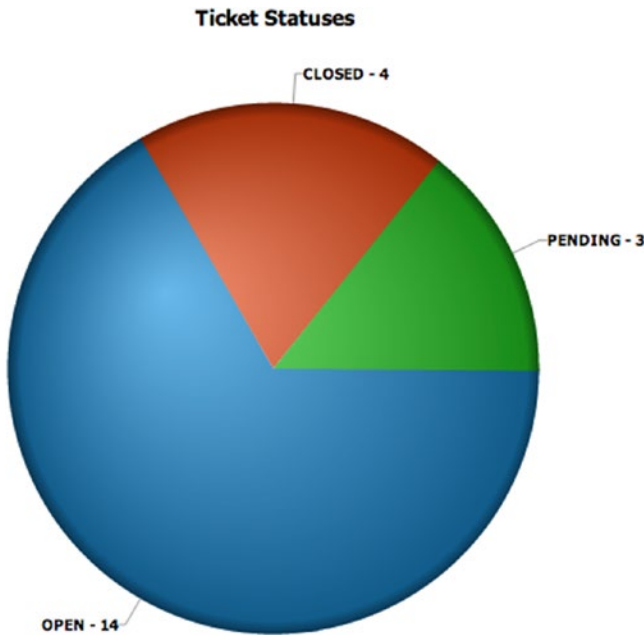
Figure 7-73. *The Ticket Statuses chart*

Filtering Data for a Chart

The link that you included in your SQL statement passes a status value to the P200_STATUS_ID field on page 200. However, you haven't created that item yet. The next steps create the item P200_STATUS_ID on page 200 so that, when a slice of the chart is clicked, the report can filter based on the status:

1. Edit **Page 200** of the application.

2. Create a new item by dragging a Select List item from the **Items** section of the Component Gallery into the **Tickets** region. Place it just in between the P200_SEARCH item and the P200_GO button, as shown in Figure 7-74.

226
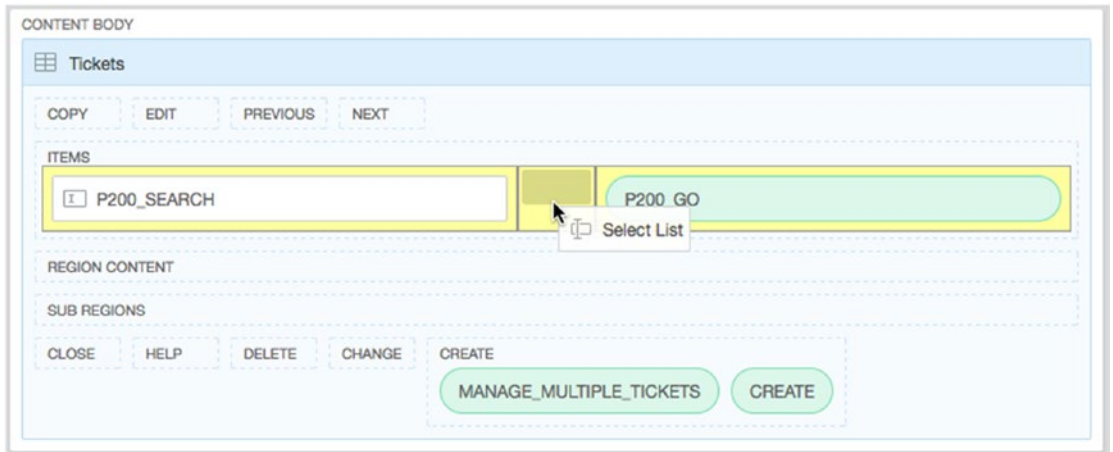
Figure 7-74. *Adding a select list item*

3. In the Edit Attributes pane, set the **Name** to P200_STATUS_ID and the **Label** to Status. (See Figure 7-75).

Figure 7-75. *Setting the name and label*

4. In the **List of Values** attribute group, set **Type to Shared Component** and **List of Values** to `P210_TICKETS_STATUS_ID`, ensure that **Display Null Values** is set to **Yes**, enter `- All Statuses -` for **Null Display Value**, and enter `%` for **Null Return Value**.

5. In the **Default** attribute group, set **Type** to **Static Value** and enter `%` for the **Static Value**, as shown in Figure 7-76.
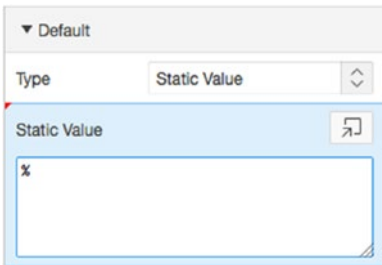
Figure 7-76. *Setting the default*

6. **Save** your changes.

Again, by default, the labels of both the Search and Description fields are set to occupy three columns of the layout grid. Let's change this so that they only take up one column each:

1. Multi-select **P200_SEARCH** and **P200_STATUS_ID**.

2. In the **Edit Attributes** pane, navigate to the **Grid** attribute group and change the **Label Column Span** to 1.

3. **Save** your changes**.**

Finally, you have to change the query for the Tickets report on page 200 to account for the value of the item P200_STATUS_ID is set to:

4. Edit the **Tickets** report on page 200 by clicking its name.

5. Append the following line to the end of the query and **Save** your changes:

```
AND tickets.status_id LIKE :P200_STATUS_ID
```

Now, run the application and navigate to the Chart page. Click any value in the chart, and that value should be passed to the Tickets page and in to the Status filter. The resulting report should only display those records that correspond to the status that was clicked in the chart.

Looking Behind the Scenes

Viewing the Chart page in the Application Builder, you can see that the only element generated is the Chart region in the Page Rendering region, as shown in Figure 7-77. The Chart region is interesting in that it has a Series element, which contains your SQL query. The Chart region embodies the logic that passes your query to the AnyChart engine to produce the chart.
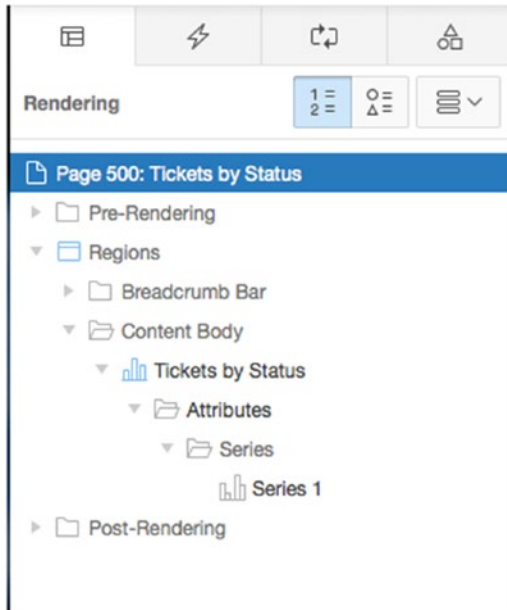
Figure 7-77. *Rendering tab of the Ticket Statuses chart*

Summary

You've reviewed most of the APEX forms and report types, and you've walked through building various forms and reports for the Help Desk system using the APEX form and report wizards. You have created an interactive report and made adjustments as both a developer and an end user. You've been introduced to charts, and you added a chart to the application to visualize your ticket status.

The common theme here is that the APEX form and report wizards are huge time-savers for developers, creating all the objects—items, buttons, branches, processes, and so on—needed for a working form, report, calendar, or chart. You were able to alter the created objects to quickly customize the generated form or report to suit your needs. Still, you haven't strayed far from what APEX builds for you.

As your application becomes more complex, there will be places where you wish to add code to enforce business rules or to perform more complex processing logic than a simple insert, update, or delete. To do so, you can use the various programmatic elements of APEX. The next chapter will address the topics of validations, computations, and processes.

■ ■ ■

Programmatic Elements

This chapter will cover the programmatic elements that can provide both simple and complex features to the APEX framework. APEX provides simple declarative features with wizards to guide you. Because of its integration with the database, APEX can also use the full power of the PL/SQL engine inside the Oracle database. As of the implementation of APEX 4, even JavaScript interactivity has been made declarative and extendable in the framework.

Conditions

Throughout the building of the Help Desk application, there will be times when you want to take advantage of the conditional logic available with APEX components. Rather than try to understand every type of condition (there are around 60 in the list of condition types), you should focus mainly on grasping the concept of a condition in general.

The condition feature provides a place where logic can turn on or off a particular piece of APEX technology. Before action is taken to display or execute a particular APEX component, the condition applied to that component is evaluated for a TRUE, or positive, result.

The logic options available to develop a condition are very broad. The condition type defines the particular mechanics used to evaluate the condition, using parameters as appropriate. Simple page-item comparisons are the easiest to explain. For example, a process may only need to be run if a particular page item has a value. In the case of sending an email, an attempt to send a message should be made only if an email address is given. From that simple start, conditions can become as complex as you need them to be. In advanced cases, conditions can also include browser and web server options.

Take time to review the condition types that are available and become familiar with their usage. It isn't as important to understand the technical implementation or syntax of each item as much as what options make up a single condition. This familiarity will be helpful when you start defining APEX components and understanding considerations for a flexible and modular application design.

Required Values

Requiring a value is a common need, and APEX 5.0 supports required values through what is essentially a NOT NULL flag at the page-item level. You don't need to create a full-blown validation (discussed next) to make an item required. You must simply make a choice from a toggle item.

Continuing with the Help Desk application, let's implement a Value Required validation on the Description field:

1. Edit **Page 210** of the application.

2. Edit the **P210_DESCR** page item.

3. In the **Validation** attribute group, change **Value Required** to **Yes**, as shown in Figure 8-1. (Depending on how you set up your UI Defaults, Value Required may already be set to Yes.)
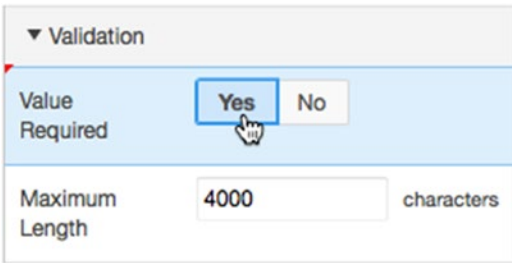
Figure 8-1. *Requiring a value to be present*

4. **Save** your changes.

To test the new validation, start by creating a ticket. Before you enter any values, click the Create button. Figure 8-2 shows the expected results with both a consolidated page-validation message box and item-validation messages.

Figure 8-2. *Validation showing required values for two elements both inline and consolidated at the page level*

In the application, the Subject element was already set up with a value requiring validation. This was done because, when you created the form using the wizard, APEX took into account the NOT NULL property of the column at the table level. You also see that the APEX wizard chose an item label template that includes an asterisk (*) at the beginning of the label text. This gives the end user the visual clue that the column is required. Be careful, however, that you don't mistake choosing a label that indicates that the field is required for actually making the field required using either the VALUE_REQUIRED attribute or a validation.

The error messages for multiple validations are cumulative. You see all validation messages when a page is processed. See Figure 8-2.

■ **Note** The message text shown is a default and can be replaced by application-specific text as a feature of globalization in the Shared Components area. There is only a single default for the entire application per language. When you need custom messages in a single-language application, I recommend using standard validation types that allow a different message for each validation you create.

Validations

The purpose of validations is to assist in providing data quality and to ensure the integrity of data entered by the user. Mechanically, validations are tests that evaluate to TRUE or FALSE. Validations are evaluated when a page is processed or submitted. All of the validations are evaluated; a FALSE return from any one of them prevents additional page processes from executing and, ideally, results in feedback to the user. Validations can also be executed on the client side using JavaScript. Although the interactive nature of JavaScript can be very attractive in the user interface, it can also be circumvented easily. Any validations that are executed in JavaScript should also be supported with appropriate validations during page processing or at the database level.

■ **Note**   It's a good practice to assume that every transaction is malicious. It's possible to implement validations strictly for security purposes, but sometimes it's difficult to step away from a process enough to identify where weak points may exist. For example, in a shopping-cart application, what would happen to the total if someone ordered -1 of a product? Would they automatically get a credit? Take extra time in the development process to look at your application so as to identify where security weaknesses may exist and to implement features that make it generally more robust and secure.

There are four types of validations: item level, page level, and, for tabular forms, column level and row level. Item-level validations operate against a single APEX item. Page-level validations are used when multiple items are involved in validating the condition. Tabular form validations behave similarly but are done against the columns and rows of the tabular form. You use an example of each in the Help Desk application.

Item-Level Validation

Validations on a single element can have attributes specific to that element, and behavior can be customized as required by that element. The example you will implement here is a validation that checks its condition only when a specific criterion is true. The requirement is to have an end date entered whenever the status is closed. Follow these steps:

1. Edit **Page 210** of the application.

2. Navigate to the **Processing** tab of the Tree Pane.

3. Right click on the **Validating** node of the **Processing** tree and select **Create Validation**, as shown in Figure 8-3.
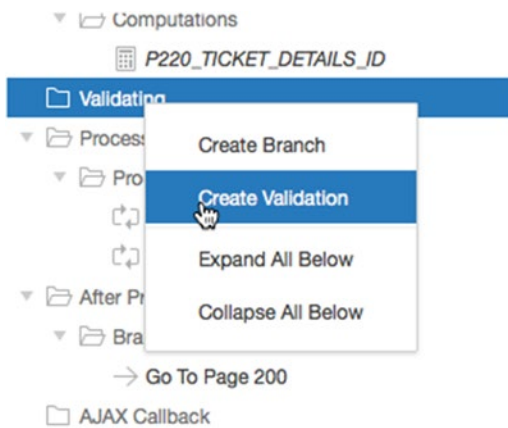
Figure 8-3. *Preparing to create a validation for the page*

4. In the **Attributes** pane, set the **Name** to Check CLOSED_ON date.

5. In the Validation attribute group, select **Item is NOT NULL** for the **Type** and then select **P210_CLOSED_ON** for **Item**.

6. Enter Please enter a value for #LABEL#. into the **Error Message** text area, and set the **Associated Item** as **P210_CLOSED_ON**, as shown in Figure 8-4. The error message shown uses a substitution variable #LABEL# to include the label of the item in the message. This way, when the label on the form item changes in the future, the validation error message will automatically reference the new label.

Figure 8-4. *Setting the validation properties*

In this step, we'll make the validation apply only when the current status of the ticket is CLOSED:

7. In the **Condition** attribute group, set **Type** to **PL/SQL Function Body**, as shown in Figure 8-5.
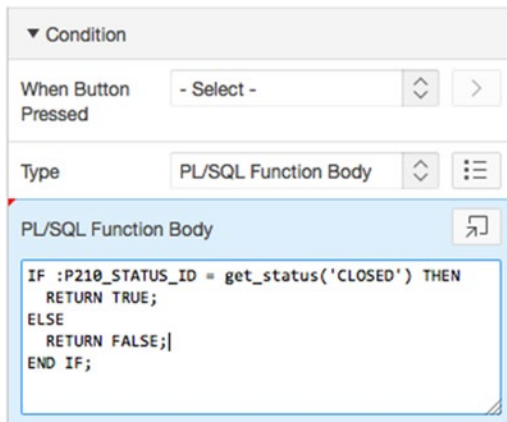
Figure 8-5. *Setting the Condition Type and Function body for the validation*

8. Enter the following code into **PL/SQL Function body**:

```
IF :P210_STATUS_ID = get_status('CLOSED') THEN
  RETURN TRUE;
ELSE
  RETURN FALSE;
END IF;
```

9. **Save** your changes.

Once the validation has been created, it appears in both the Rendering and Processing tabs on the APEX Page Designer, as shown in Figure 8-6. Both references point to the same implementation and are shown for easy navigation.
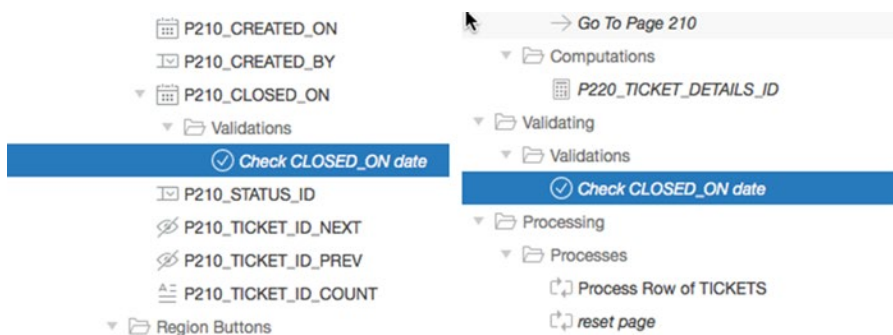
Figure 8-6. *Validations created appear in two places on the Application Builder page*

This validation now requires that a value be entered for the Closed On item when the ticket status is set to CLOSED. The condition applied to the validation is evaluated every time the page is submitted.

Page-Level Validation

Page-level validations apply to one or more items simultaneously and often can be an entire PL/SQL block of code that must evaluate to TRUE in order for the validation to be successful. The requirement for the Help Desk application is to compare the Created On date with the Closed On date to ensure that they occur in chronological order. A ticket that is closed before it's created doesn't make any sense. This is a good example of using a validation to ensure data quality. Here's how to create the validation you need:

1. Edit **Page 210** of the application.

2. Navigate to the **<u>Processing</u>** tab of the Tree Pane.

3. Right click on the **<u>Validating</u>** node of the **<u>Processing</u>** tree and select **Create Validation**.

4. In the **<u>Attributes</u>** pane, set the **<u>Name</u>** to `Closed Date must be after Creation Date`.

5. Enter `Closed On Date must be Later than the Created Date` for **<u>Error Message</u>**, set the **<u>Type</u>** to **PL/SQL Function Body (returning Boolean)**, and then enter the following code into the **<u>PL/SQL Expression</u>** text area. Figure 8-7 shows the completed values. Click **Next** to continue:

```
IF TO_DATE(:P210_CREATED_ON,'DD-MON-YYYY') >
  TO_DATE(:P210_CLOSED_ON,'DD-MON-YYYY')
THEN
  RETURN FALSE;
ELSE
  RETURN TRUE;
END IF;
```
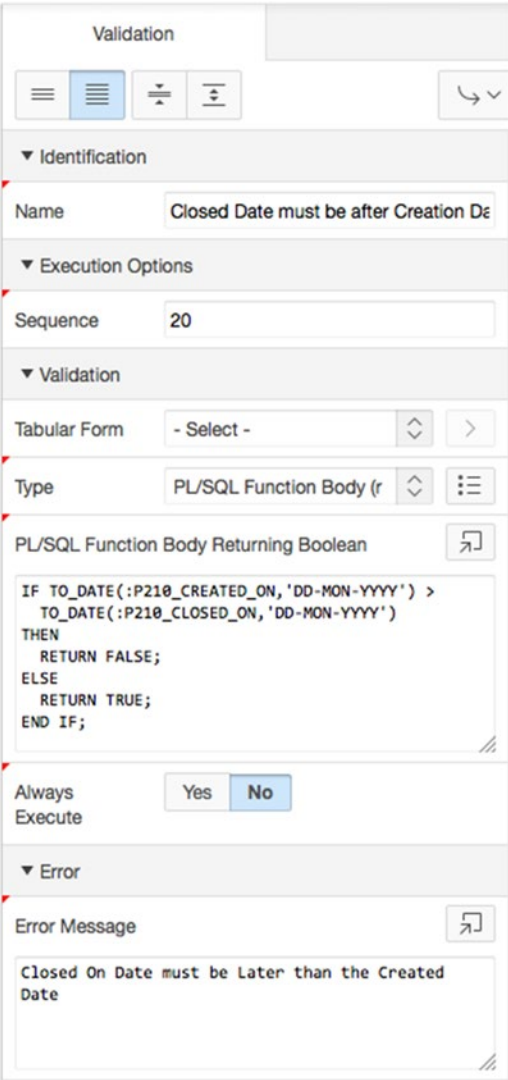
Figure 8-7. *Validation attributes*

    6.   **Save** your changes.

    In your application, you now have a feature that helps ensure the quality of the data being entered. This type of data check makes sure any metric that calculates time from start to end doesn't produce a negative answer due to dates. This improves the quality of the data and the reliability of the metrics that are produced in reports.

Tabular Form Validation

Tabular forms in APEX 5.0 are able to perform validations better than they did in previous versions. The wizard that creates a tabular form also adds validations for you. The wizard creates validations automatically based on the data model. However, a wizard can only know so much about your business process, and the data model may have more flexibility than you want in your application.

Looking at the definition of page 230, the wizard has created a number of Not Null validations for you, based on the NOT NULL attributes in the underlying TICKETS table. However, the wizard can't know that you require a Closed On date when a ticket is closed. You can apply that validation using a column-level tabular form validation:

1. Edit **Page 230** of the application.

2. Navigate to the **<u>Rendering</u>** tab in the Tree Pane.

3. Expand the **Columns** node of the **<u>Manage Multiple Tickets</u>** tabular from.

4. Right click on **CLOSED_ON** and select **Create Validation** from the context menu.

5. In the Attributes pane, set **<u>Name</u>** to CLOSED_ON is Not Null if Ticket is CLOSED.

6. Set the **<u>Tabular Form</u>** to **Manage Multiple Tickets**, the **<u>Type</u>** to **Column is NOT NULL**, and the **<u>Column</u>** to **CLOSED_ON**, as shown in Figure 8-8.
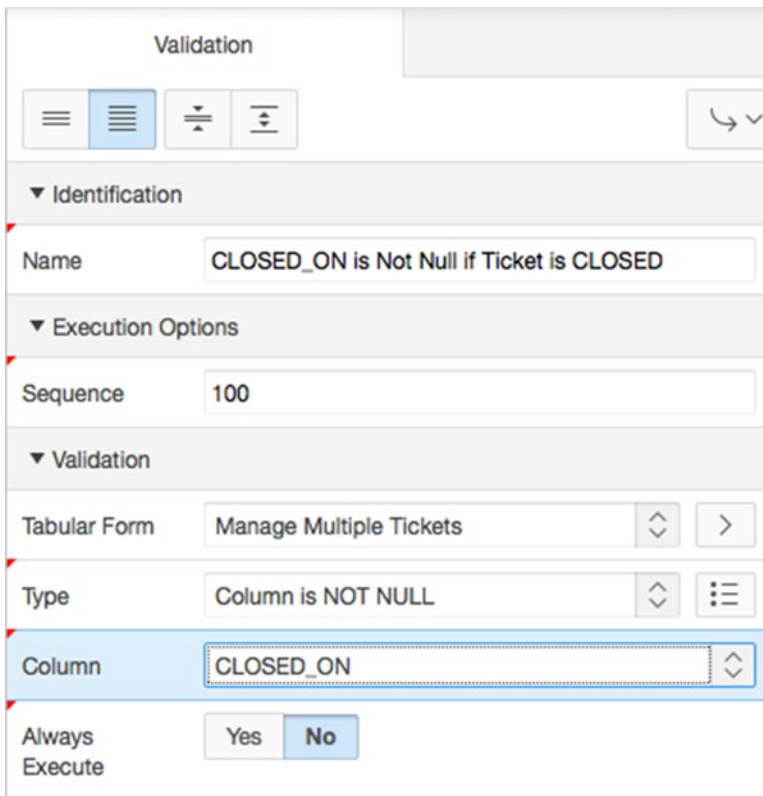
Figure 8-8. *Setting Validation Name and Validation attributes*

7. For **Error Message**, enter #COLUMN_HEADER# must be entered if Status is CLOSED, as shown in Figure 8-9. Click **Next**.

Figure 8-9. *An error message using substitution variables*

8. In the Condition attribute group, set **Type** to **PL/SQL Function Body** and the **Execution Scope** to **For Created and Modified Rows**, and then type the following code into the **PL/SQL Function Body** text area:

```
IF :STATUS_ID = get_status('CLOSED') THEN
  RETURN TRUE;
ELSE
  RETURN FALSE;
END IF;
```

9. **Save** your changes.

When you run the Manage Multiple Tickets page, you can test the new validation either by adding a new ticket with a status of CLOSED and no Closed On date set, or by removing the Closed On date of an existing closed ticket and attempting to save the changes. In Figure 8-10, each row that doesn't meet the validation requirement is highlighted and appears in a list of errors at the top of the page. In this example, the rows that didn't have a Closed On date failed the validation and are flagged as needing attention.
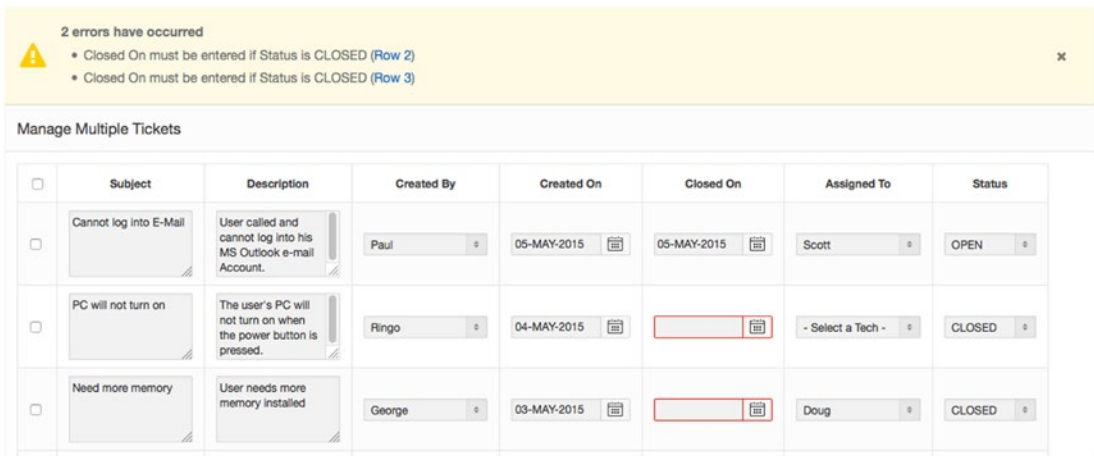
Figure 8-10. *Results that fail validation are highlighted and presented in the message area*

■ **Note** By default, these validations are only executed for new or changed rows. You can change this behavior by setting the Execution Scope of the validation, located in the Conditions section.

The Create Validation wizard also allows the creation of row-level validations on tabular forms. These validations are run once for each row being processed by the tabular form. At this level, you could easily create a validation, similar to the one created for page 210, that checked to see if the Closed On date occurred after the Created On date.

As an exercise to see how much you've learned, try to implement that validation at the row level of the tabular form on page 230.

Computations

An APEX computation is analogous to a PL/SQL function. The intent is to act on an item in the application by setting the value using a variety of methods. This allows information to be derived rather than just stored in the data tables. Computations can be implemented when a page is rendered or after a page is submitted back to the server, depending on the needs of the application. Computations can act on any item available within an application. Items that can be set include items on the current page, items on another page, and even application-level items.

There is also a type of computation that can be used at the application level. It's available in an application's shared components. This type of computation has additional options for execution points, including a computation point called On New Instance that executes when a new session (or instance) is given to a user when they log in.

Execution

It's important to understand when a computation is executed relative to when a value is shown on a page and to when other values are available to the computation. When using the value of an item in a computation, the current session state for that item is the value that is used. A computation sets an item value in session state,

and any processing (computations, validations, or processes) that uses that item after it has been set sees the results of that computation. When a page is rendered, it shows what is in the session state for that item at the time it's shown on the page. The computation point is the setting that determines when the computation is executed.

On the page definition screen, several computation points are shown in the page tree. You can adjust the computation point by clicking and dragging the computation in the tree to a different computation point, or by editing the computation and changing the values for the sequence and computation point directly. The sequence only orders the computations within a given computation point. In general, the page renders and processes as shown on the page definition screen, starting at the top and going down the list to the bottom. There are only minor exceptions, such as dynamic actions and AJAX callbacks, which have variable points of execution.

Types

Computations have much of the same flexibility as other APEX components do. They can be complex or simple, with the full capabilities of the Oracle database to support them. The types of computations are as follows:

- *Static Value*: Simple static text value

- *Item*: Name of another item in the application

- *SQL Query (Return Single Value)*: Any SQL statement as long as it returns a single row and a single column

- *SQL Query (Return Colon-Separated Values)*: SQL used for multi-select items

- *SQL Expression*: Expression used in the SELECT portion of an SQL statement

- *PL/SQL Expression*: Same as SQL Expression

- *PL/SQL Function Body*: PL/SQL function syntax with a RETURN statement

- *Preference*: The value of an APEX user preference as stored in the metadata

Computations can be conditional in the same manner as many of the other APEX components are. The conditions can be as complex as the business rules require, with the ability to use the database features and APEX session items to evaluate the condition. Conditions evaluating to TRUE result in the computation being executed.

Creating a Computation

The Help Desk application has a requirement to display the number of days a ticket has been open. The result should be a derived value that changes depending on the day and status of the record being reviewed. You can accomplish this by putting a new item on the page that displays the result of the computation:

1. Edit **Page 210**.

2. From the Items section of the Components Gallery, click and drag the Display Only item so that it appears just to the right of P210_SUBJECT. See Figure 8-11.
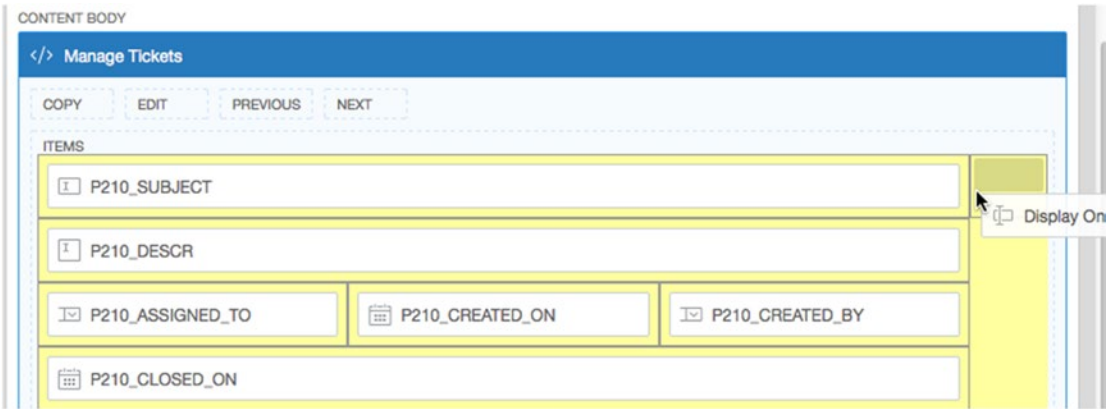
Figure 8-11. *Placing the Display Only item*

3. In the Attributes pane enter P210_DAYS_OPEN for **Name** and Days Open for the **Label**. Set **Save Session State** to **No**.

Now there's a new item in the region that you use as a container for the calculation. Next, we create the calculation so that the value is set to the number of days the ticket has been open. However, we only want this to appear for tickets that have already been created and not for new tickets being entered.

4. In the **Conditions** attribute group, set **Type** to **Item Is NOT NULL**.

5. When the region refreshes, set the value of **Item** to **P210_TICKET_ID**, as shown in Figure 8-12.
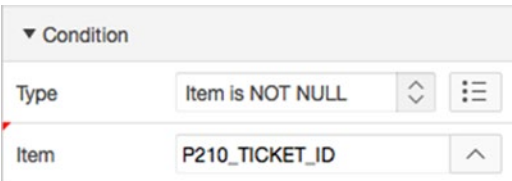
Figure 8-12. *Showing an item only when another item contains a value*

6. In the **Source** attribute group, set the **Type** to **Null**.

7. In the Rendering tab of the Tree Pane, right-click **P210_DAYS_OPEN**, and from the context menu, select **Create Computation**, as shown in Figure 8-13.
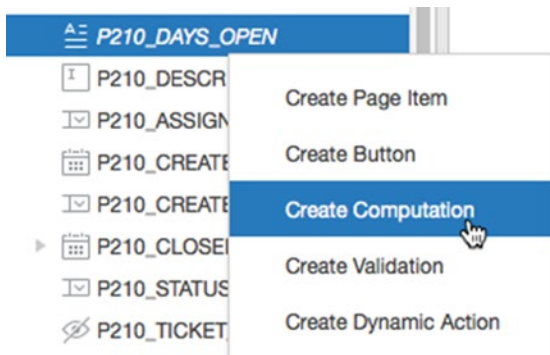
Figure 8-13. *Using the right-click shortcut to create a computation*

8. In the Attributes pane, set <u>**Type**</u> to **SQL Query (Return Single Value)**.

9. In the <u>**SQL Query**</u> text area, enter the following SQL statement (also shown in Figure 8-14), and then **Save** your changes:

```
SELECT
  DECODE(status, 'CLOSED', closed, open_or_pending) days_open
FROM
  (
  SELECT
    ROUND(sysdate - t.created_on) open_or_pending,
    NVL(ROUND(t.closed_on - t.created_on),0) closed,
    sl.status status
  FROM
    tickets t,
    status_lookup sl
  WHERE
    t.status_id = sl.status_id
    and t.ticket_id = :P210_TICKET_ID)
```

```
▼ Computation

Type        SQL Query (return single value)  ⇕  ☰

SQL Query                                        ⟱

SELECT
  DECODE(status, 'CLOSED', closed, open_or_pending)
days_open
FROM
  (
  SELECT
    ROUND(sysdate - t.created_on) open_or_pending,
    NVL(ROUND(t.closed_on - t.created_on),0) closed,
    sl.status status
  FROM
    tickets t,
    status_lookup sl
  WHERE
    t.status_id = sl.status_id
    and t.ticket_id = :P210_TICKET_ID)
```
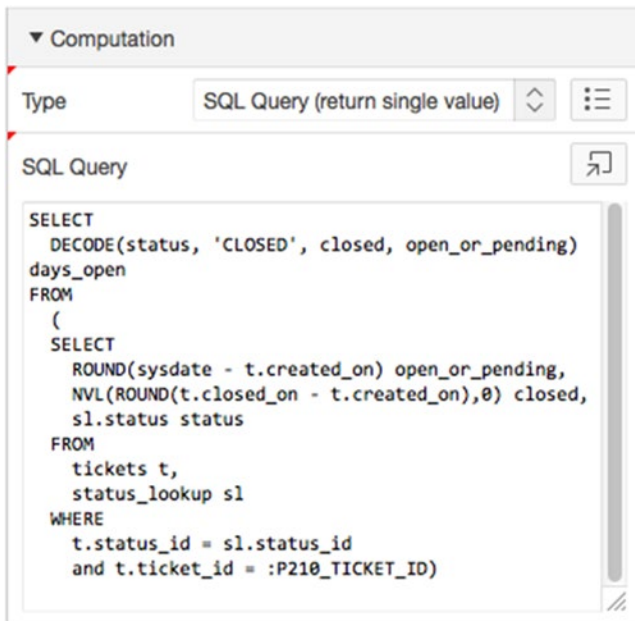
Figure 8-14. *Entering the SQL statement for a computation*

To see the results of adding the new item, run the application and navigate to the Tickets report (page 200). Click one of the Edit icons to bring up the single-record view (page 210). You should now see the result of the computation as a number of days. When starting the process of creating a new ticket, the field isn't displayed, as the condition prevents the field from showing.

Processes

If computations are analogous to database functions, then processes are analogous to database procedures. A process is a container for a unit of logic.

Processes are arguably the most complex part of APEX, because they're the construct used to deal with data processing in the database as well as with references to APIs, such as those used to send email and perform any other business logic required in the application. When dealing with data forms, the APEX wizard creates built-in processes that manage the reading and writing of data from the form. Those types of built-in processes are called *data-manipulation processes*.

Processes, similar to computations, can occur during both page rendering and page processing. Processes support the APEX conditions feature, which allows processes to be written as individual logic units, with conditions determining whether the logic is needed.

Execution Points

Process execution points are the same as execution points for computations. The most commonly used execution points for processes are *On Submit - After Computations and Validations* and *On Demand - Run This Process When Requested by AJAX*, because these points support button-press activities and dynamic actions. The full list is as follows:

- New Session
- Before Header
- After Header
- Before Regions
- After Regions
- Before Footer
- After Footer
- After Submit
- Processing
- AJAX Callback

Processes can be defined at the individual page level or at the application level as part of the shared components. Functionally, page processes and application processes behave the same way. The difference is found in where business logic is contained. For processes that need to run on all pages, you can define an application process. Also, just as with regions, you can use Global Pages to define processes that run on every page, but only for page rendering.

Process Types

Each different process type has a different use depending on the requirements. The types and their uses are as follows:

- *Automatic Row Fetch*: Retrieves records from a single database table or view
- *Automatic Row Processing (DML)*: Process to insert, update, or delete a record from a single database table or view
- *Clear Session State*: Clears session state values; also referred to as cache
- *Close Dialog*: Process to close the current modal or non-modal dialog
- *Form Pagination*: Process to retrieve the previous or next record from a database table or view. Most often used in master-detail forms
- *Load Uploaded Data*: Process to load the parsed spreadsheet data into an existing table or view
- *Parse Uploaded Data*: Process to parse the prepared spreadsheet data in preparation for loading into an existing table
- *PL/SQL Code*: Generally use for utilizing database PL/SQL logic
- *Prepare Uploaded Data*: Process to prepare spreadsheet data for uploading into an existing table

- *Reset Pagination*: Resets pagination for a report

- *Send Email*: Declarative interface to easily send email

- *Tabular Form – Add Rows*: Process to add a row into a tabular form region.

- *Tabular Form – Multi-Row Delete*: Process to delete multiple rows from a tabular form region

- *Tabular Form – Multi-Row Update*: Process to update multiple rows from a tabular form region

- *User Preference*: Process to set user preferences for the end user.

- *Web Services*: Submits a request to a web-service provider

- *Plug-ins*: Processes functionality provided by plug-ins

Processes in the Help Desk Application

The details behind processes can be very complex. In order to provide an adequate example, let's include a simple process in the Help Desk application: a requirement that the application keep track of the last time a record was modified. You can do this by updating a Last Updated date on the record every time it's saved. There's more than one way to accomplish this task. Here, you will do it with a process.

First, you need to add the LAST_UPDATED field to the TICKETS table. To do this you use the SQL Workshop again:

1. From the **SQL Workshop** drop-down menu, choose **Object Browser**, as shown in Figure 8-15.
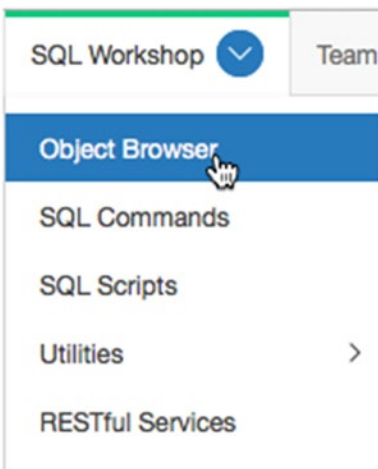
Figure 8-15. *Navigating to the SQL Workshop Object Browser*

2. Select the TICKETS table from the list of objects at left.

3. Click the **Add Column** button above the table definition, as shown in Figure 8-16.

Column Name	Data Type	Nullable	Default	Primary Key
TICKET_ID	NUMBER	No	-	1
SUBJECT	VARCHAR2(255)	No	-	-
DESCR	VARCHAR2(4000)	Yes	-	-
CREATED_BY	VARCHAR2(50)	Yes	-	-
CREATED_ON	DATE	No	-	-
CLOSED_ON	DATE	Yes	-	-
ASSIGNED_TO	VARCHAR2(50)	Yes	-	-
STATUS_ID	NUMBER	Yes	-	-

Download | Print

Figure 8-16. *Adding a column to the table*

4. Enter LAST_UPDATED for **Add Column** and DATE for **Type**, and click **Next**.

5. Click **Finish**.

Now you can add the process to the page:

6. Edit **Page 210** of the application.

7. In the Processing tab of the Tree Pane, right-click the **Processes** node and choose **Create** from the context menu, as shown in Figure 8-17.
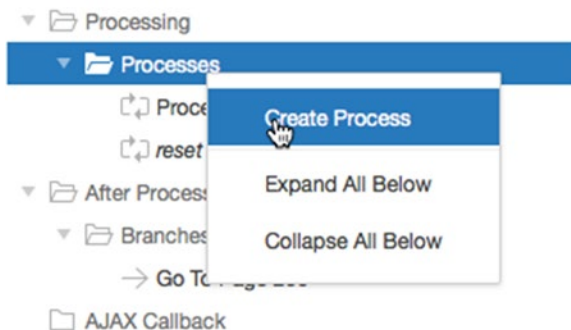
Figure 8-17. *Using the context menu to create a process*

8. Set the **Name** of the process to Set Last Processed and set **Point** to **Processing**. Set the **Type** to **PL/SQL Code**.

249

In the next step, you will set the contents of your anonymous PL/SQL block. If you're unfamiliar with a PL/SQL anonymous block, it's PL/SQL code that has a BEGIN and an END that wrap the contents. You need to follow PL/SQL syntax conventions, including ending statements with semicolons. It's possible to nest anonymous blocks of code, but that isn't necessary for this example:

9. Enter the following SQL into the **PL/SQL Code** text area (see Figure 8-18):

```
BEGIN
  UPDATE tickets SET last_updated = sysdate
    WHERE ticket_id = :P210_TICKET_ID;
END;
```
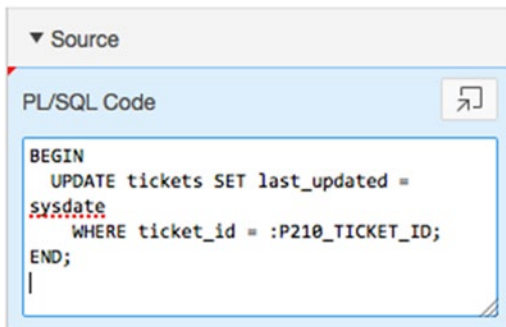

Figure 8-18. *Entering the anonymous PL/SQL block*

10. Leave both the **Success** and **Error** messages empty. These messages will appear at the top of a page as feedback to the user after the process completes. Your requirements don't call for you to notify the user that the Last Updated date was changed.

11. In the **Condition** attribute group, change **When Button Pressed** to **SAVE**.

12. **Save** your changes.

At this point, the process has been created. Currently, you don't show the Last Updated date in the summary report. In order to see the value on the report, you will need to add the LAST_UPDATED column to the query from which the report draws data. That report resides on page 200 of your application:

1. Edit **Page 200**.

2. Edit the **Tickets** region by clicking the region's name in the tree.

3. Add the LAST_UPDATED date to the **Region Source** of the report, as in the following SQL. Click **Save** when you're finished:

```
select TICKET_ID,
       SUBJECT,
       DESCR,
       CREATED_BY,
       CREATED_ON,
       CLOSED_ON,
       ASSIGNED_TO,
       STATUS,
```

```
    LAST_UPDATED
  from TICKETS,
       STATUS_LOOKUP
 where TICKETS.STATUS_ID = STATUS_LOOKUP.STATUS_ID
   and UPPER(SUBJECT) LIKE '%'||UPPER(:P200_SEARCH)||'%'
   and tickets.status_id LIKE :P200_STATUS_ID
```

To test and review the change, run the application and navigate to the Tickets report. Edit any ticket, and click the Apply Changes button. You should now see a value for Last Updated indicating the current day.

This is a quick example of how you can use a process to apply form-based logic. When the form is used to make changes, a brief piece of PL/SQL makes a record change automatically. Packages, procedures, and APIs all can be reached using processes similar to this one.

PL/SQL Regions

The PL/SQL region type is effectively an open container for PL/SQL with the additional option to generate output. You can use Oracle Web Application (OWA) Toolkit procedures such as htp.p to generate the output. References to APEX items can be made using bind variable syntax (for example, :P1_ITEM_NAME), the v function (for example, v('P1_ITEM_NAME')), or substitution string syntax (for example, &P1_ITEM_NAME.) to support the logic contained in the region.

PL/SQL regions differ from process regions in that PL/SQL regions are executed only during page rendering, whereas processes can run during both page processing and page rendering. PL/SQL regions have the advantage of being able to generate content directly on the page. A use case for this type of output is the need for a complex report format that is beyond the ability of a standard report template. In that case, a PL/SQL package that generates the needed HTML output can be written and called by a PL/SQL region.

In the Help Desk application, you want to make the home page a bit more useful by adding a quick summary of the number of tickets an individual has open. This is applicable only if someone is logged in. So if they aren't logged in, a simple greeting message will suffice. You can accomplish the task of adding the summary by adding a PL/SQL region with some logic to output the appropriate message:

1. Edit **Page 1**.

2. Edit the **APEX Issue Tracker** region by clicking its name in the tree.

Currently, this region is a standard Static Content region, emitting exactly the HTML code you enter into it. You want to make it dynamic, so switch it so it uses PL/SQL:

3. In the **Identification** attribute group, change **Type** to **PL/SQL Dynamic Content.**

4. Enter the following code for the **PL/SQL Code** text area, replacing the static HTML that was there, and then **Save** your changes:

```
DECLARE
  l_count NUMBER;
  l_status_id NUMBER := get_status('OPEN');
BEGIN
IF :APP_USER != 'nobody' THEN
  SELECT count(*)
    INTO l_count
    FROM tickets
    WHERE assigned_to = :APP_USER
    AND status_id = l_status_id;
```

251

```
        htp.p('<h1>Welcome to the APEX Issue Tracking System, '
          || :APP_USER || '</h1>'
          || 'You have ' || l_count || ' Open tickets.<br />'
          || 'Select an option from the list');

    ELSE
        htp.p('<h1>Welcome to the APEX Issue Tracking System</h1>'
          || 'Select an option from the list');
    END IF;

END;
```

This code implements logic that makes a decision based on the user-substitution variable `:APP_USER` and tailors the `htp.p` output according to that distinguishing factor. APEX provides "nobody" as a user name when a user isn't yet logged in, so the logic keys off of that value.

When the PL/SQL region is generated for a user who isn't yet logged in, a simple welcome message is produced (see Figure 8-19). When a user who has credentials is logged in to the application, a message similar to that in Figure 8-20 is produced that shows a user-specific greeting and a quick count of the number of open tickets assigned to that user.
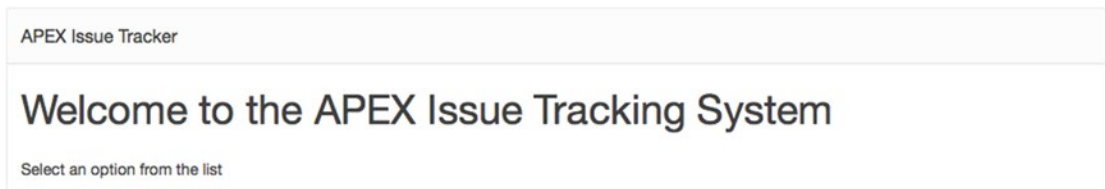
Figure 8-19. *Issue Tracker PL/SQL region when the user isn't yet logged in*
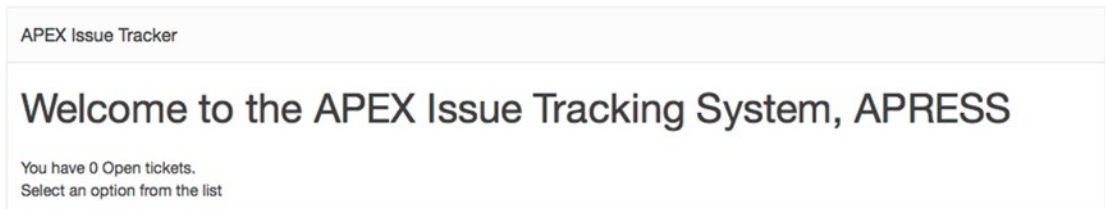

Figure 8-20. *With an authenticated user, the PL/SQL region generates a greeting and a ticket count*

In this section, you've created a dynamic PL/SQL region that alters the output based on the application user. This section's example, although simple, shows how the content of a region can be as dynamic as necessary with the use of the PL/SQL in the database.

Dynamic SQL

Dynamic SQL is a term for SQL that isn't finalized at design time, but rather is assembled at runtime by any number of dynamic criteria. Dynamic SQL is used when the exact requirements of an SQL statement aren't known until runtime, or when the SQL needs to change while the application is running. Dynamic SQL lets you modify column lists, where clauses, joins, and any other portion of an SQL statement while an application is running.

APEX supports dynamic SQL in reports and can support PL/SQL functions returning SQL statements as a result. There are some constraints, however. Functions must return a valid SQL statement. Depending on the implementation, a statement may need to return a set of generic columns if the number of columns isn't known or will vary.

The Help Desk application has the requirement to differentiate public tickets from private ones. To accomplish that goal, you can implement a public flag feature. Implementing the flag requires a quick update to your data model and then an implementation of dynamic SQL on the Home Page report. Start by making the data modification:

1. Navigate to the **SQL Workshop**.

2. Click the **SQL Commands** icon.

3. Enter the following SQL statement in the text area, and click the **Run** button. This adds the new column called PUBLIC_FLAG to the TICKETS table:

   ```
   ALTER TABLE tickets ADD (public_flag VARCHAR2(1))
   ```

4. Enter the following SQL statement in the text area, replacing the current statement, and click **Run**. Ensure that the Autocommit checkbox is checked. This updates all the current tickets to a default value of N:

   ```
   UPDATE tickets SET public_flag = 'N'
   ```

Now that the data-model modifications are complete, you can move on to the application. Add the option to see and edit the new value in the ticket edit screen:

5. Edit **Page 210** of the **Help Desk** application.

6. Add a new Radio Group item to the **Manage Tickets** region using Drag & Drop from the Component Gallery. Position the new item to the right of the Status ID item.

7. Enter P210_PUBLIC_FLAG for **Name** and Public Flag for the **Label**.

8. In the **Settings** attribute group set **Number of Columns** to 2.

9. In the **Appearance** attribute group set **Template** to **Required**.

10. Set **Value Required** field to **Yes.**

11. In the **List of Values** attribute group, set **Type** to **Static Values**, while in the **Static Values** text area enter STATIC:Y,N

12. Set **Display Null Value** to **No**, as shown in Figure 8-21.

Figure 8-21. *The LOV for the public flag*

When you add a column to a form that relates to a database column in the table on which the form operates, a few settings have to be changed. Source Used and Source Type work together to identify how each item gets its value:

13. In the **Source** attribute group, set **Type** to **Database Column**, which in turn sets **Source Used** to **Always, replacing any existing value in session state**. Ensure that **Database Column Name** is PUBLIC_FLAG.

14. In the **Default** attribute group, set the **Type** to **Static Value** and enter N for **Static Value**.

15. **Save** your changes.

Now that you have a PUBLIC_FLAG column in your data model and the ability to control it through the Tickets form, you can create the dynamic SQL report on page 1 to display tickets with a Public option for unauthenticated users:

1. Edit **Page 1** in your application.

2. Create a new region by clicking the **Create** button in the Page Designer toolbar and selecting **Report Region**, as indicated in Figure 8-22.
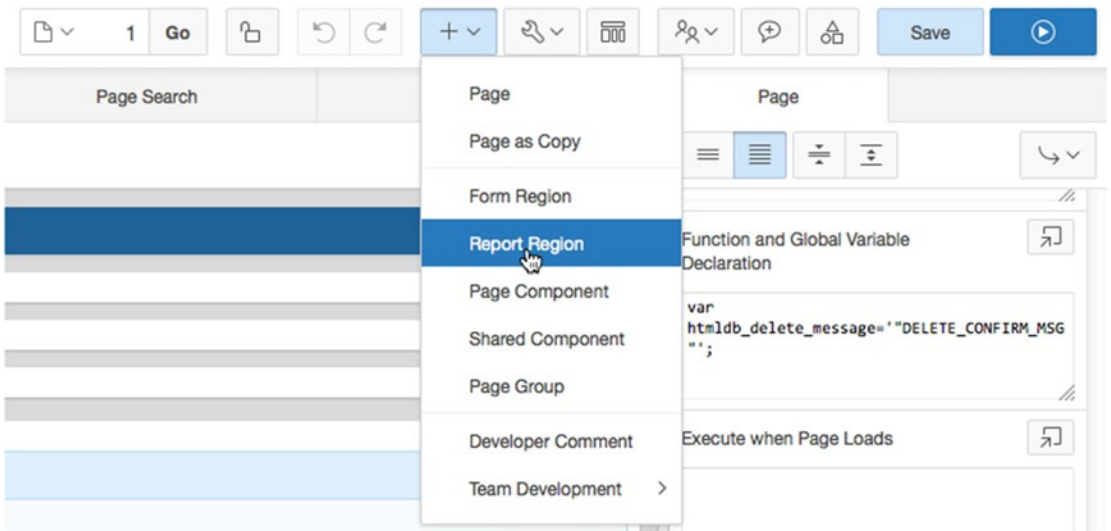
Figure 8-22. *Creating a region for the SQL to generate your report*

    3.   Select **Classic Report**, and click **Next**.

    4.   Enter `Current Open Issues` for **Title**, as shown in Figure 8-23. Click **Next**.
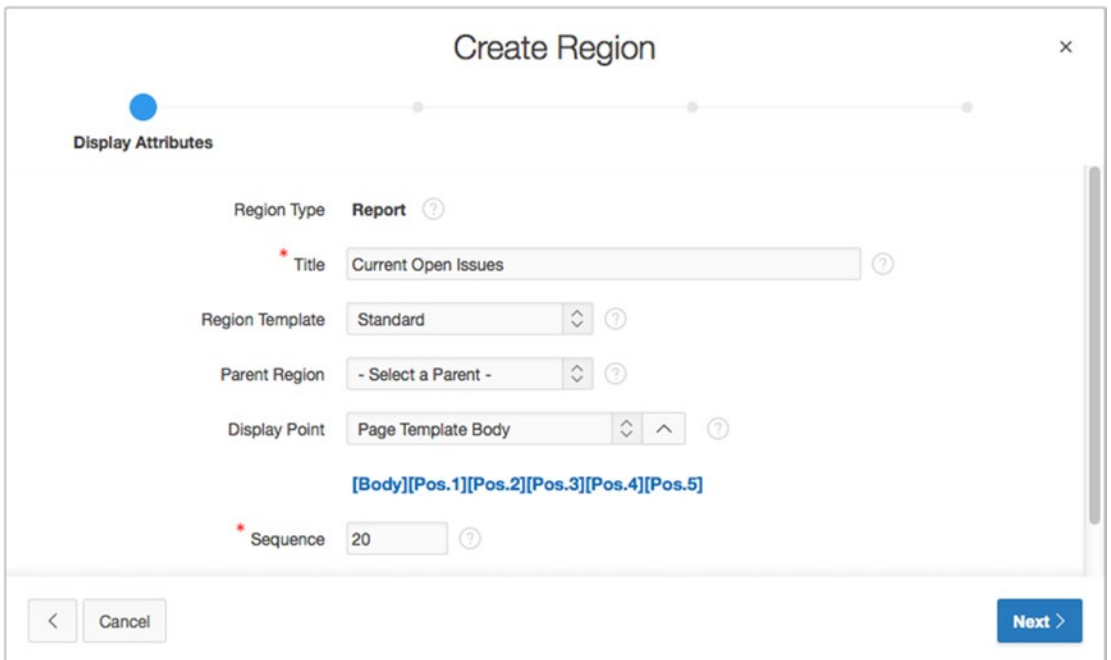
Figure 8-23. *Region title and display point*

5. Set the **Source Type** to **SQL Query** and enter the following SQL into
    the **Region Source**. Click the Create Region button when you're finished to
    accept the defaults for all of the remaining settings:

```
DECLARE
  l_sql VARCHAR2(500);
BEGIN

l_sql := l_sql ||  q'!
                SELECT
                  subject,
                  created_on,
                  assigned_to
                FROM
                  tickets t,
                  status_lookup sl
                WHERE
                  t.status_id = sl.status_id
                  AND sl.status = 'OPEN'
         !';

IF :APP_USER = 'nobody' THEN
  l_sql := l_sql || q'! AND public_flag = 'Y' !';
END IF;

RETURN l_sql;
END;
```

To see the results of this report fully, you need to set a few tickets with the new PUBLIC setting. Navigate to the ticket summary screen as a logged-in user and change a few OPEN tickets to have the PUBLIC option set to Yes. When you navigate to the home screen as a logged-in user, a full list of open tickets should appear, as shown in Figure 8-24. After logging out, you will see only the tickets that have been identified as PUBLIC.

APEX Issue Tracker

Welcome to the APEX Issue Tracking System, APRESS

You have 0 Open tickets.
Select an option from the list

Current Open Issues

Subject	Created On	Assigned To
Accidentally deleted Q2.ppt	24-APR-2015	DAN
Smartphone will not sync with Outlook	22-APR-2015	DOUG
VPN Client Install Issues	20-APR-2015	DOUG
Mouse is not working	19-APR-2015	TIM
Speakers are too soft	18-APR-2015	SCOTT
Disk is Full	16-APR-2015	DOUG
New Ticket	17-JUL-2015	-
I think I have a virus	27-APR-2015	DAN
BSOD after rebooting	29-APR-2015	DOUG
Network drive not being mapped	30-APR-2015	TIM
Need to install SP2	01-MAY-2015	TIM
Need more memory	03-MAY-2015	DOUG
Cannot log into E-Mail	05-MAY-2015	SCOTT
Funny smell coming from PC	25-APR-2015	TIM

1 - 14

Figure 8-24. *Resulting report generated from dynamic SQL*

■ **Note**   The SQL statement uses a quoting syntax that you may not be familiar with. Oracle Database 10g introduced a quoting mechanism for string literals that allows you to define your own string delimiters, removing the need to double up single quotes in strings. Any character can be used as a delimiter, including bracket combinations () {} [] <>. The basic syntax is q'X string X' where X is any single character. The q'X opens the literal string, and the X' closes the literal string. You can find more details on the literal syntax in the *Oracle Database SQL Language Reference*.

Summary

As with any programming language or framework, learning the basics is the first step. This chapter touched on a lot of points that could be considered to be the tips of icebergs. Each section has the capability to reach into a vast set of technologies, with the Oracle database being primary among them. The intention here is to demonstrate how the APEX framework works through the example application and to provide a starting point for additional detail discovery.

CHAPTER 9

■ ■ ■

Security

Security has varying degrees of implementation; there's never a black-and-white answer. The question of how much security is needed is followed up by additional questions regarding the value of what is being protected and the risks, repercussions, and likelihood of it being sought after. For every security measure, there will always be someone trying to circumvent it. This chapter will review basic security features and an approach to securing the Help Desk application. The concepts reviewed here apply to all APEX applications and are specific to the APEX framework.

User-Maintenance Navigation

In the Help Desk application, you have the requirement to allow users to be maintained in the application through the web interface. Let's add a section to the application that allows for the maintenance of user accounts, and then let's modify the tab structure so as to navigate to the newly created form. This time, you won't use the Create Page wizard to create the menu items, but instead will create them from scratch in the Shared Components section so that you may gain a better understanding of how the menu hierarchy works.

First, create a blank page that will be the landing page for your new tab (tabs require a page to reference):

1. From the Application Builder home page, while editing the Help Desk application, click the **Create Page** button.

2. Select the option for **Blank Page** and click **Next**.

3. Set **Page Number** to 600, enter Users for the **Name** field, and set the **Breadcrumb** selection to **Breadcrumb**. When the page refreshes, ensure that **Entry Name** is **Users** and click **Next**.

4. Select **Do not associate this page with a navigation menu entry** for the **Navigation Preference** radio group. Click **Next**.

5. Click **Finish** to complete the creation of the page. The completed page should be empty, and you should not see any new menu item related to it, as shown in Figure 9-1.
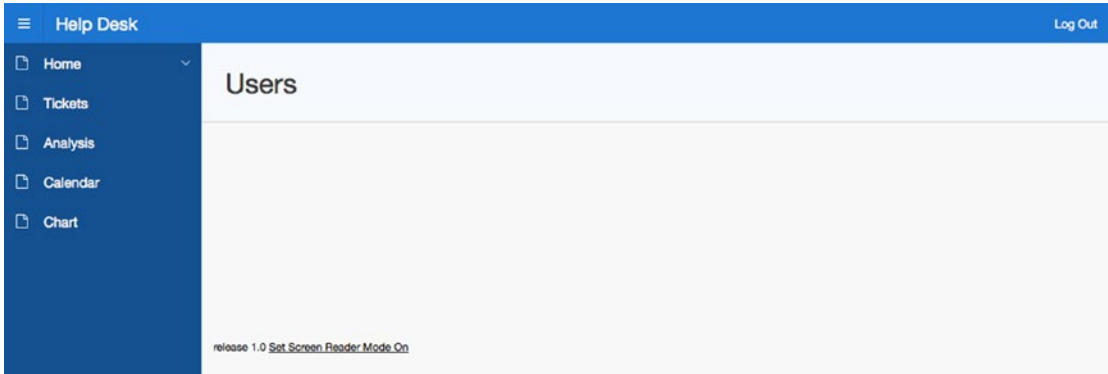
Figure 9-1. *Viewing the newly created empty page with its single breadcrumb entry*

Now that you have a Users page, you need to make a modification to the navigation. We'll add an Admin entry to the menu and create sub-entries for user maintenance. Here's the process to follow to add the new menu items with the correct hierarchy:

1. Navigate to the **Shared Components** page.

2. In the **Navigation** section, click **Navigation Menu**.

3. The interactive report shows the available menus. Click on **Desktop Navigation Menu**, as shown in Figure 9-2.
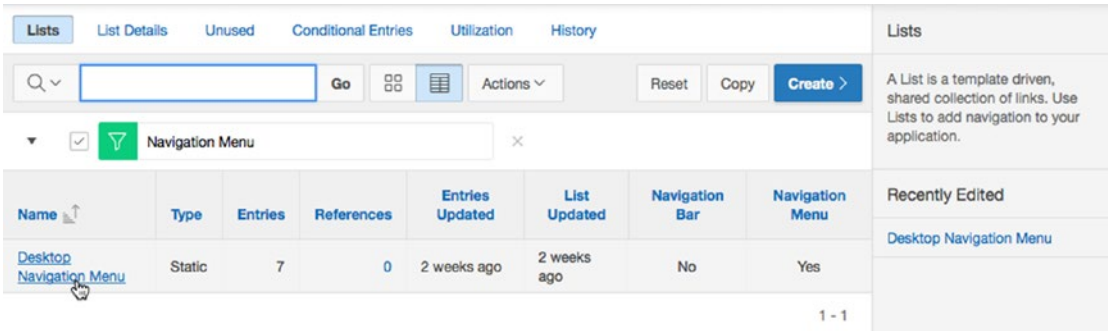

Figure 9-2. *Clicking the Desktop Navigation Menu*

Looking at the current menu structure, as shown in the report in Figure 9-3, you can see we already have seven entries, and that two of the entries (Submit a Ticket and Contact Us) are sub-entries of the Home menu entry. Next, we will add the Admin menu entry and then add the sub-entry for user maintenance.

Sequence ↑	Name	Parent Entry	Target	Conditional	Updated	Level	Authorization Scheme	Copy
10	Home	-	f?p=&APP_ID.:1:&APP_SESSION.::&DEBUG.:	-	4 weeks ago	1	-	🗐
20	Submit a Ticket	Home	f?p=&APP_ID.:2:&SESSION.::&DEBUG.:2:::	-	-	2	-	🗐
30	Contact Us	Home	f?p=&APP_ID.:3:&SESSION.::&DEBUG.:3:::	-	-	2	-	🗐
40	Tickets	-	f?p=&APP_ID.:200:&SESSION.::&DEBUG.	-	3 weeks ago	1	-	🗐
50	Analysis	-	f?p=&APP_ID.:300:&SESSION.::&DEBUG.	-	3 weeks ago	1	-	🗐
60	Calendar	-	f?p=&APP_ID.:400:&SESSION.::&DEBUG.	-	3 weeks ago	1	-	🗐
70	Chart	-	f?p=&APP_ID.:500:&SESSION.::&DEBUG.	-	2 weeks ago	1	-	🗐

Figure 9-3. *Viewing the currently existing Navigation Menu entries*

4. To add the Admin menu entry, click the **Create List Entry** button in the upper-right corner, as indicated in Figure 9-4.

Figure 9-4. *Creating a new menu entry using the Create List Entry button*

5. In the Entry section, enter Admin for **List Entry Label**.

6. In the Target section, enter 600 for **Page** and click **Create List Entry at the top of the page**. See Figure 9-5.

Figure 9-5. *Entering the attributes for the new Admin menu entry*

1. To add the User Maintenance menu entry, click the **Create List Entry** button in the upper-right corner.

2. In the Entry section, set the **Parent List Entry** to **Admin** and then enter `User Maintenance` for **List Entry Label**.

3. In the Target section, enter 600 for **Page.**

4. In the Current List entry section, set **List Entry Current for Pages Type** to **Comma Delimited Page List**.

5. In the **List Entry Current for Condition**, enter `600,610` and click the **Create List Entry** button at the top of the page.

You now have a User Maintenance menu item as a sub-entry to your Admin menu item. When the user clicks the parent menu item, they are taken to the page indicated when you created the menu entry, but a tab may be active for other pages in the application, too. In this case, the User Maintenance menu item will be current for both page 600 and 610. It's OK that you haven't created page 610 yet—you will shortly.

Running the application now will show the results seen in Figure 9-6. The page that is currently active changes the highlight that is applied to the different tab elements.
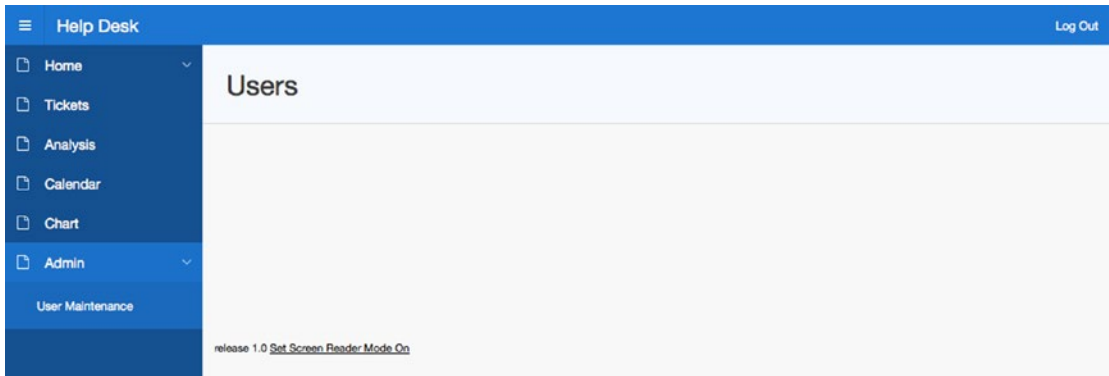
Figure 9-6. *The new navigation menu showing the Admin entry and User Maintenance sub-entry*

You now have a navigational framework that clearly distinguishes the items needed to administer the application. This design is extensible. As the application grows with time, additional features requiring administration could be added to this navigational structure.

User-Maintenance Data Entry

As part of the Help Desk design, you should be able to maintain the users from the application. To do this, you need to implement some new database objects.

    1.   Upload and run the script ch9_security_objects.sql. Refer to Chapter 4 if you need step-by-step instructions. You should see 13 rows, all of which complete successfully.

        Let's walk through briefly what this script does for you:

- *Lines 1–16*: Create a function called hash_password that encodes any string passed to it.

- *Lines 18–24*: Create the USERS table that will hold the user records.

- *Lines 26–27*: Create the USER_SEQ sequence that will be used as the primary key for the USERS table.

- *Lines 29–37*: Create a Before Insert trigger on the USERS table that automatically assigns the next sequence as the primary key, converts the user name to uppercase, and calls the hash_password function to encrypt the user's password.

- *Lines 39–50*: Create a Before Update trigger that converts the user name to uppercase and hashes the user's password if it has changed.

- *Lines 52–87*: Create the authenticate_user function that validates whether the passed user name and password are valid compared to what exists in the USERS table.

- *Lines 90–103*: Create six entries in the USERS table, all with the password *apress*.

Now that you have your new database objects, you can continue to implement the security model:

2. Edit **Page 600** of the application.

3. Create a new region by clicking the **Create** button and selecting **Form Region**.

4. Select **Form on Table with Report** and click **Next**.

Because the report is actually quite small and contains very few columns, it's probably overkill to create it as an interactive report, so stick to the Classic report in this instance:

5. Set **Implementation** to **Classic**.

6. Enter Users for **Region Title** and set **Region Template** to **Standard**. The settings look like those in Figure 9-7.

Figure 9-7. *Report page setup*

7. Click **Next**.

8. Set **Table/View Owner** to your schema name and set the **Table/View Name** to **Users (table)**, as shown in Figure 9-8.

Figure 9-8. Setting the owner and table names

9. Click **Next**.

10. Select USER_ID and USER_NAME as the columns to be displayed in the report. Remove the PASSWORD column by shuttling it to the left, and then click **Next**.

11. Select any Edit link image and click **Next**.

12. Enter 610 for **Page Number** and Manage Users for **Page Name** and **Region Title**, as shown in Figure 9-9. Click **Next**.

Figure 9-9. *Defining the name of the Manage Users form*

13. Set **Primary Key Type** to **Select Primary Key Column(s)** and, when the page refreshes, select USER_ID for **Primary Key Column 1**. Click **Next**.

14. Select **Existing Trigger** for **Primary Key Source** and click **Next**.

15. Select USER_NAME and PASSWORD as the columns to be editable on the form, as shown in Figure 9-10, and click **Next**.
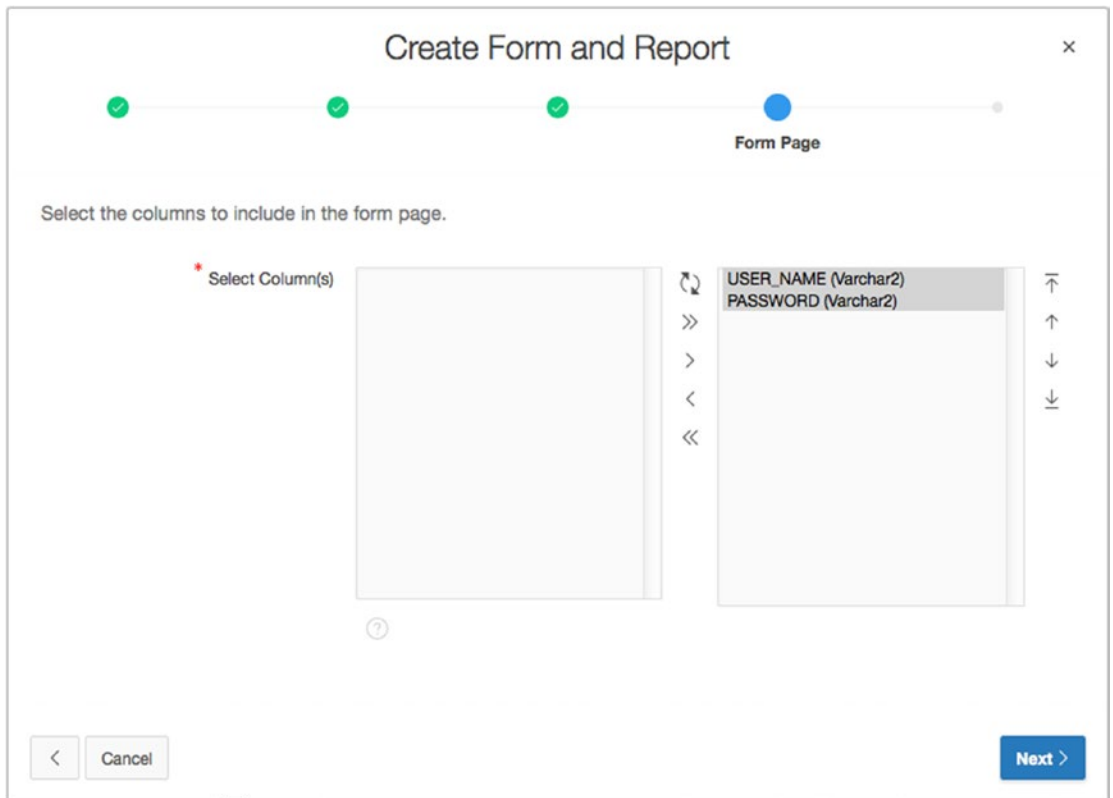
Figure 9-10. *Select* USER_NAME *and* PASSWORD *as fields to be seen in the form*

16. Set **Insert**, **Update**, and **Delete** all to **Yes** and click **Next**.

17. Click **Create**.

At the completion of these steps, the Help Desk application has some additional objects. The region on page 600 is the report of the current users. Also notice the new page that allows editing of the data values, including all the processes to do the corresponding database transactions. However, you still need to do a few things to page 610 in order for it to display the breadcrumbs properly:

18. Navigate to the Shared Components region for your application.

19. In the Navigation Section, click the **Breadcrumbs** link.

20. Click the **Breadcrumb** icon to view all of the breadcrumb entries.

Viewing the breadcrumb entries, we can see that, while there is an entry for page 600, there is no entry for page 610. We'll need to add that manually:

21. Click the **Create Breadcrumb Entry** button in the upper-right part of the page.

22. In the Breadcrumb section, enter 610 for **Page.**

23. In the Entry section, select **Users(Page 600)** as the **Parent Entry** and then enter
    Manage Users for the **Short Name**.

24. In the Target section, enter 610 for **Page**. These entries are shown in Figure 9-11.

Figure 9-11. *Entering the breadcrumb details for page 610*

25. At the top of the page, click **Create Breadcrumb Entry**.

When you're finished, page 610 has a Shared Components breadcrumb entry just like page 600. Running the application displays shows a breadcrumb entry for both the Users report page and the Manage Users page, as shown in Figure 9-12.
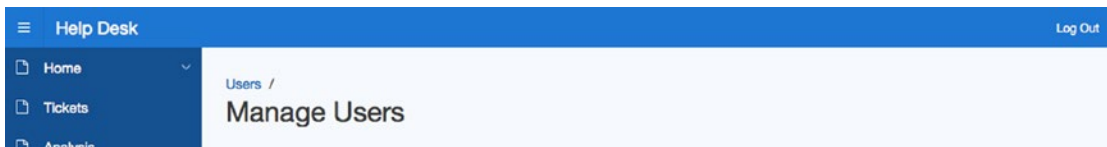

Figure 9-12. *Showing the breadcrumb entry for the Manage Users page*

Finally, you need to change the item type of P610_PASSWORD to Password, so it accepts a user's input but displays asterisk (*) characters as the password is typed. This item type is designed not to retrieve data when a record is edited, despite being bound to a database column. Also, the item type doesn't save any value in session state, meaning it doesn't remember the value entered after the page processing is complete. This is a security feature to prevent data identified as a password from being retrieved inappropriately. Here are the steps:

26. Edit **Page 610**.

27. Edit the item **P610_PASSWORD**.

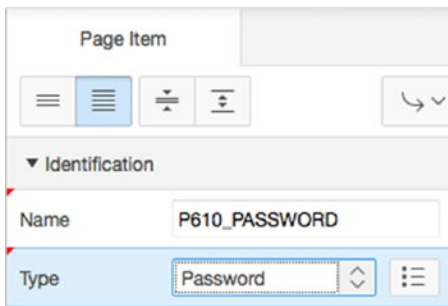28. In the Properties Editor, set **Type** to **Password**, as shown in Figure 9-13.

Figure 9-13. *Setting the P610_PASSWORD element to a password field*

Although you want a password to be required when creating a new account, if the admin user doesn't enter a password while editing an existing user, you want the system to keep the current password. Because of this, you need to set the Value Required attribute of the password field to NO and instead implement a validation that only fires when you're creating a new user:

29.   In the **<u>Validating</u>** section of **P610_PASSWORD**, set the **<u>Value Required</u>** attribute to **NO**.

30.   While editing **Page 610**, right-click **P610_PASSWORD**, and select **Create Validation**.

31.   In the Properties Editor, set **<u>Name</u>** to **P610_PASSWORD Is Not Null**.

32.   In the Validation Section, select **Item is Not Null** as **<u>Type</u>** and set **<u>Item</u>** to **P610_PASSWORD**.

33.   Enter `A password must be specified.` for **<u>Error Message</u>**.

34.   In the Condition section, set **<u>When Button Pressed</u>** to **CREATE**.

35.   **Save** your changes.

This completes the navigation and UI part of the security scheme you're implementing. With the navigation and maintenance in place, you can now implement the authentication scheme that will use the information.

Authentication

The key to making a secure application is to understand whom the accessing user is. APEX refers to this as *authentication*. Authentication answers the question, "Who are you?" The APEX tool provides a series of predefined authentication mechanisms, including a built-in authentication framework and an extensible custom framework. At design time, it's easy to switch between authentication methods by setting the active scheme. There can be only one active authentication scheme at a time for an application. The following are the major types of authentication schemes:

- *Application Express Accounts*: Users are managed in the APEX workspace and are maintained just like workspace developer accounts.

- *LDAP Directory*: The user is an existing LDAP-compliant server such as Active Directory or Oracle Internet Directory.

- *Oracle Application Server Single Sign On*: Authentication can pass between APEX and an existing Oracle SSO server. Logging into the SSO server once passes the same credentials to all APEX applications.

- *Database Accounts*: Database user names and passwords determine authentication. Don't confuse this with data access in an APEX application.

- *HTTP Header Variable*: This approach supports the use of HTTP header variables to identify a user and to create an Application Express user session.

- *Custom*: Logic is determined by the developer. An example of usage is for Internet-facing applications where self-registration may be desired. Another example is when more than one authentication source is used simultaneously, such as using two LDAP servers.

- *Open Door*: Developer testing simulates logging in as different individuals. This isn't intended to be used as a public authentication scheme.

- *No Authentication*: This option is intended to allow all parts of the application to be reachable without needing a user to log in.

Each application has its own set of authentication schemes managed as part of its Shared Components. Authentication schemes can be copied between applications when needed. This ability to copy is especially useful when a custom authentication scheme has been developed and is desired in more than one application. The authentication schemes also utilize the APEX subscription framework to allow a master copy to be applied to subscribers inside of a single workspace.

Custom Authentication Schemes

In the previous section, the script that was imported included definitions for tables, triggers, and functions. You will use those elements as part of your custom authentication scheme. The key component of the authentication scheme is a function that compares the given user name and password to the stored values in the USERS table. If there is a match, then the user is authenticated. You should review the database objects and PL/SQL function code from the SQL Workshop for more details on how this is implemented.

■ **Note**   Although the USERS table contains a field named PASSWORD, it's not the actual password value; it's an encrypted hash of the password. Passwords should never be stored as plain text.

Here's the process to follow to create a custom authentication scheme based on the database objects just mentioned:

1. Navigate to the **Shared Components** of the application.

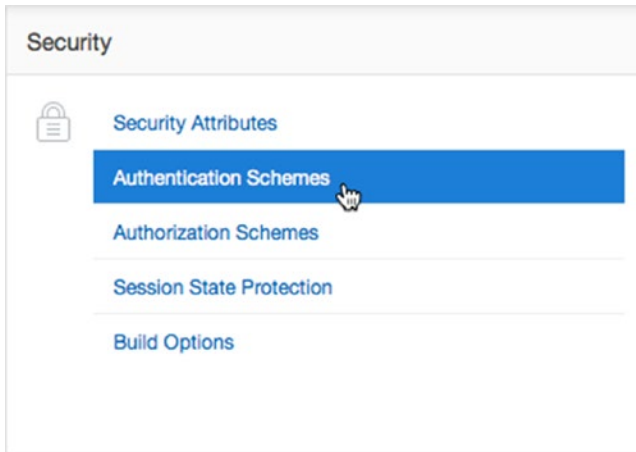2. In the **Security** region, click **Authentication Schemes** as shown in Figure .

Figure 9-14. *Navigating to the Authentication Schemes shared component*

3. Click the **Create** button at the upper right on the **Authentication Schemes** screen.

4. Select **Based on a pre-configured scheme from the gallery** and click **Next**.

5. Enter Custom Authentication Scheme for **Name**, and then select **Custom** for **Scheme Type**. The page refreshes and displays different entry options based on the scheme type selected.

6. In the **Settings** section, enter authenticate_user for **Authentication Function Name**, as shown in Figure 9-15. You don't need to fill out any of the other items in this section.

Figure 9-15. *Setting the Authentication Function Name*

7. Click **Create Authentication Scheme.**

■ **Note** No parameters are used here, nor is a PL/SQL semicolon. This is part of the definition of how APEX handles custom authentication functions. The `authenticate_user` function that was created earlier conforms to the expected signature: a function returning a `BOOLEAN` value with two parameters: `p_username` `varchar2(255)` and `p_password varchar2(255)`.

By default, when you create a new authentication scheme, it's automatically set to be the active scheme. Now you must use the user names and passwords that exist in the `USERS` table to log in to your application.

Run the application, and if it shows that you're logged in, log out. You can sign on as any of the following users: Scott, Doug, Martin, Karen, Patrick, or Tim; all passwords are *apress* in lowercase.

Conditional Security

Many aspects of APEX are conditional. One pair of conditions is particularly applicable to the authentication status: User Is the Public User and User Is Authenticated. These conditions can help you limit objects in APEX to be available either to public users (those who haven't logged in) or to authenticated users (those who have logged in).

By applying security rules to the Help Desk application, you can improve usability by restricting the display of menu options that aren't available to the public. This avoids confusion and improves the overall user experience when accessing the application. Let's walk through the creation of this condition:

1. Navigate to the **Shared Components** area of the application.

2. In the Navigation section click the **Navigation Menu** link.

3. Click the **Desktop Navigation Menu** link to view the Navigation Menu entries.

4. Edit the **Tickets** menu item by clicking on its name.

5. In the Condition section, set **Condition Type** to **User is Authenticated (not public)** and click **Apply Changes**. Figure 9-16 shows the expected value of the condition.

Conditions	
Condition Type	User is Authenticated (not public)
	PL/SQL item / column=value item / column not null item / column null request=e1 exists never none

Figure 9-16. Setting the menu item condition

6. Repeat steps 4 and 5 for the **Analysis**, **Calendar**, and **Chart**, and **Admin** menu items.

Run the application now and click the Logout link. The Admin, Tickets, Analysis, Calendar, and Chart menu items should disappear, leaving only the Home menu item and its children. Logging in again should restore the display of the tabs as they were previously seen.

Access Control

APEX includes a built-in feature for creating an access-control framework with three roles: Administrator, Edit, and View. The wizard is designed to create data structures to store the roles, pages to edit the assignments, and authorization schemes to be used throughout an application. This wizard makes the job of creating basic security capability very easy in an application. The summary of the objects created can be seen in Figure 9-18 as the last step in the wizard.

There are, however, downsides to using the built-in access-control mechanism. If you require more granular access control than the Administrator, Edit, and View roles provide, then you're likely going to want to create your own access-control mechanisms from scratch. For the Help Desk application, these roles will suffice. Here's how to implement access control in the Help Desk application:

1. Navigate to the Application Builder home page for your application and click **Create Page**.

2. Select **Access Control** and click **Next**.

3. Enter 620 for **Administration Page Number** and click **Next**.

4. Select **Create a new navigation menu entry**, allow the page to refresh, and then enter `Access Control` for **New Navigation Menu Entry**. Then, set the **Parent Navigation Menu Entry** to **Admin**, as shown in Figure 9-17. Click **Next**.

Figure 9-17. *Assign page 620 to a new menu entry under the Admin entry*
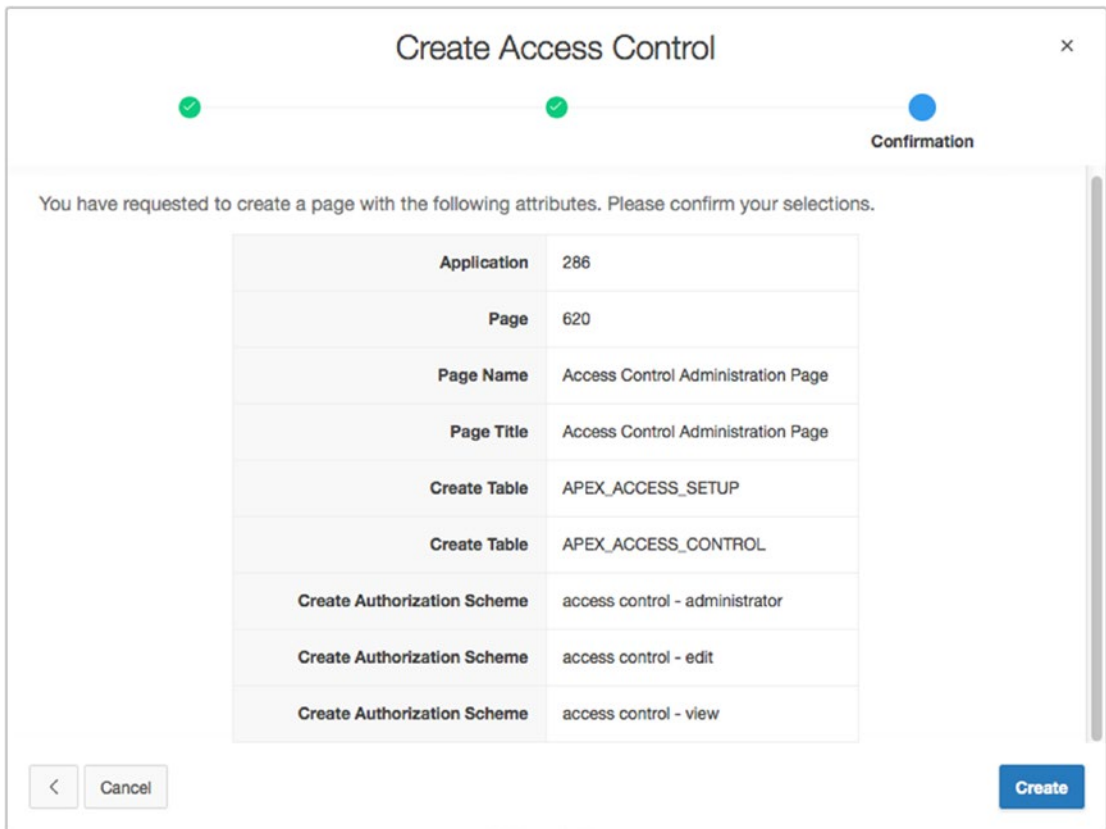
5. Click **Create**, as shown in Figure 9-18.

Figure 9-18. *Viewing the object summary as part of the Access Control wizard*

With the completion of the wizard, all the objects have been created and are available for use. Before you enable the security utility, you need to add some users to allow you to use the admin functions. Running the application now, you may notice that the user name is simply an open text field. You should create a list of values (LOV) as a shared component that contains all the users for whom you want to control access. Because the access-control page is now part of the application, you can alter it as needed. To increase the quality of the data entered, update the user field to be a select list:

6. Edit **Page 620**.

7. Expand the **Columns** node for the **Access Control List** report.

8. Edit the ADMIN_USERNAME column.

9. In the **Identification** section, set **Type** to **Select List.**

10. In the **List of Values** section, set the **Type** to **SQL Query** and then enter the following SQL Statement in the **SQL Query** text area and save your changes:

```
SELECT user_name d, user_name r
FROM users
```

When you run page 620, notice that no breadcrumb has been created for the page. You can do this as follows:

1. Navigate to the shared components for your application.

2. In the Navigation section, click the **Breadcrumbs** link.

3. Click the **Breadcrumb icon** to edit the breadcrumb entries.

4. Click the **Create Breadcrumb Entry** button in the upper right of the page.

5. In the Breadcrumb section, enter 620 for **Page.**

6. In the Entry section, enter Access Control for the **Short Name**.

7. In the Target section, enter 620 for **Page**.

8. At the top of the page, click **Create Breadcrumb Entry**.

Next, you need to associate a privilege with each of the existing users via the access-control pages:

9. Run the application and log in with the user **SCOTT**.

10. Navigate to the **Access Control** screen by clicking the **Admin** menu item and then the **Access Control** sub-entry.

11. In the **Access Control List** section, click **Add User**.

12. Select **Scott** for **Username**, set **Privilege** to **Administrator**, and click **Add User**.

13. Select **Doug** for **Username**, set **Privilege** to **Edit**, and click **Add User**.

14. Select **Patrick** for **Username**, set **Privilege** to **Edit**, and click **Add User**.

15. Enter **Martin** for **Username**, set **Privilege** to **View**, and click **Apply Changes**.

Your results should look similar to those in Figure 9-19. Every time a new user is added, the listing in the report updates. You can now use these users to test the application.

Figure 9-19. The Access Control List with user names and privileges

One of the features of the access-control utility is the ability to enable or disable the enforcement of the utility itself. Running page 620 displays the header shown in Figure 9-20. By default, the access-control utility is set to Full Access. To enable the access-control features, set the mode using the following steps:

16. Run **Page 620**.

17. Set **Application Mode** to **Public read only. Edit and administrative privileges controlled by access control list.**

18. Click the **Set Application Mode** button shown in Figure 9-20.
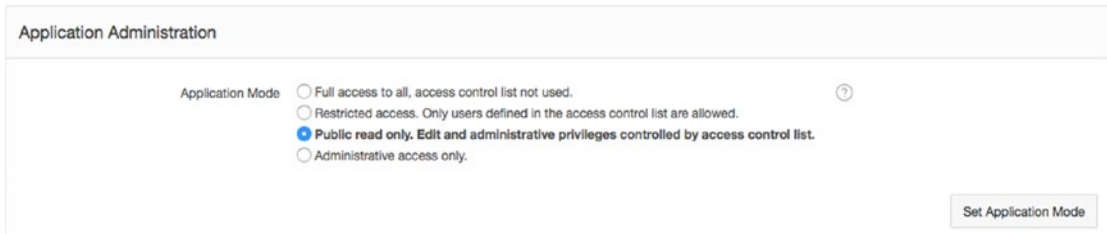
Figure 9-20. *The access-control list enabled as public read only*

You now have the editing forms in place and all the data set up properly, although the application isn't yet using any of the restrictions you've created. You will do that in the next section.

Authorization

Whereas authentication answers the question "Who are you?" authorization works to answer the question "What are you allowed to do once logged in?" APEX provides shared components of an application called *authorization schemes*. These authorization schemes can be applied to components within the application to tell the APEX engine when the components should be executed or rendered.

When you created the access-control pages, APEX created three authorization schemes for you, one for each role available in the edit screens: Admin, Edit, and View. Figure 9-21 shows the Authorization Schemes shared component report.

Figure 9-21. *The authorization schemes created as part of the access-control mechanisms*

The last step in this process is to start locking down pages using these authorization schemes. First, let's lock down the administrator section of the application so that only a user with Admin privileges can use it:

19. Edit **Page 620**.

20. Edit **Page Attributes** by clicking the page name.

21. In the **Security** section of the Properties Editor, set **Authorization Scheme** to **access control - administrator**, as shown in Figure 9-22. **Save** your changes.
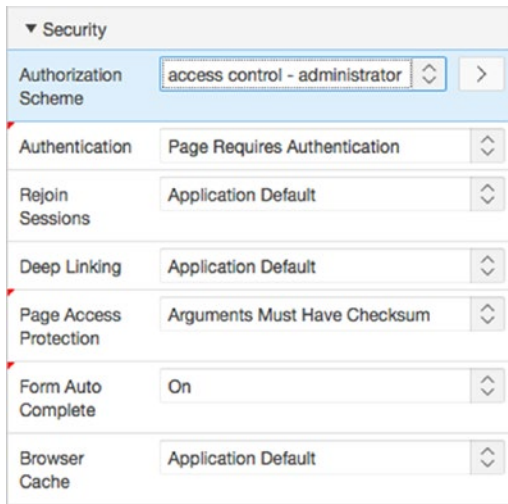

Figure 9-22. *Setting the authorization scheme at a page level*

22. Repeat steps **22** and **23** for pages **600** and **610**.

Now that the authorization scheme has been implemented on the administration pages, you can test the security behavior. Only a user set up with the Administrator role on the access-control page can use Admin pages 600 through 620.

Log in to the application as the user Scott, and you can navigate all the administration functions. Logging in as any other user and clicking the Admin parent tab results in the message shown in Figure 9-23.
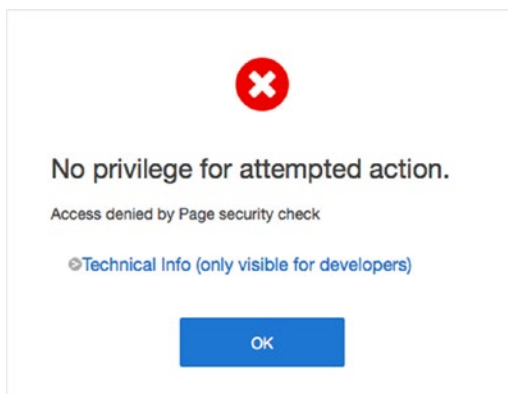

Figure 9-23. *Error message generated when the authorization scheme returns a denied result*

The error message in Figure 9-23 isn't very friendly. An application should make every effort to avoid the type of event that would cause a privilege error. In this application, the Admin menu item should be removed from the page when it doesn't meet the access restrictions. You can accomplish this by applying the same authorization scheme to the menu item itself:

23. Navigate to the Shared Components for your application.

24. In the Navigation section click the **Navigation Menu** link.

25. Click the **Desktop Navigation Menu** link to edit the navigation entries.

26. Edit the Admin menu entry by clicking on its name.

27. Under **Authorization**, set **Authorization Scheme** to **access control - administrator**, and click **Apply Changes**.

Now, when running the application, if the user isn't privileged with administrator access, the menu item doesn't display. This avoids the event that would cause the user to see the access-denied error message.

You've applied the authorization scheme at both the page level and the tab level for the administration pages. Next, let's remove the ability for a view-only user to create new records by associating the Edit authorization scheme with the button required to create tickets:

28. Edit **Page 200** of the application.

29. Edit the **Create** button by clicking its name.

30. In the **Security** section, shown in Figure 9-24, set **Authorization Scheme** to **access control - edit**, and click **Save**.
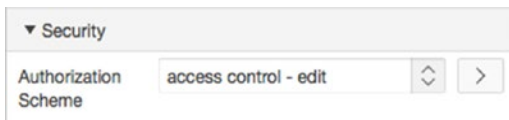

Figure 9-24. *Security setting for the buttons*

31. Repeat step **32** for the **Manage Multiple Tickets** button.

To test this change, log in with the user name Martin. This user has been granted view privileges, so the buttons on page 200 aren't shown. Does this mean that Martin can't create tickets?

Let's review the steps you applied to the Admin pages. Security was first applied to the page itself, and then additional security was applied to prevent the access-denied error. In the case of the buttons to create tickets, security to remove the buttons doesn't prevent the page from being run directly either from the Application Builder or by changing the page number in the URL to 210 or 230.

■ **Important**   Removing or hiding a button, a tab, or another link doesn't secure the target it was pointing at; it only helps reduce errors seen by users on components that are already secure.

The design for the Help Desk application has the Manage Multiple Tickets page only available to users with edit privileges, so the entire page is secured at the edit level. The single-record view of a ticket continues to be visible to all authenticated users, but without the buttons related to record manipulation:

32.   Edit **Page 210** of the application.

33.   Edit the **<u>Create</u>** button in the **<u>Manage Tickets</u>** region by clicking its name.

34.   In the **<u>Security</u>** section, set **<u>Authorization Scheme</u>** to **access control - edit**.

35.   Repeat steps **35** and **36** for the **<u>Delete</u>** and **<u>Save</u>** buttons as well as for the second **<u>Create</u>** button located in the **<u>Ticket Details</u>** region. Remember to **Save** your changes.

■ **Note**   The previous step could also be completed by using APEX 5.0's new multi-edit capability. Simply multi-select the items you wish to edit and change the Authorization Scheme once for all selected items.

---

36.   Edit **Page 220** of the application.

37.   Edit the **Create** button by clicking its name.

38.   In the **<u>Security</u>** section, set **<u>Authorization Scheme</u>** to **access control - edit**.

39.   Repeat steps **39** and **40** for the **<u>Delete</u>** and **<u>Save</u>** buttons. Remember to **Save** your changes.

40.   Edit **Page 230** of the application.

41.   Edit the page attributes by clicking the page name.

42.   In the **<u>Security</u>** section, set **<u>Authorization Scheme</u>** to **access control - edit**, and click **Save**.

Review the application now with different users. Notice how the user Martin can still navigate from the Tickets report to view the details of the ticket, but there are no buttons to modify the records in the database. Even though the form elements are editable, they aren't written back to the database without the proper form submission.

Read-Only Items

Normally, users can edit the contents of an item in APEX. There are instances where you want to prohibit them from doing so, but you don't want to hide the item entirely. At the conclusion of the previous step, the user Martin doesn't have the ability to save edits of the ticket information even though the form allows Martin to change the contents of the form items.

To assist in preventing changes, each item in APEX has a read-only attribute that you can set programmatically. The approach is similar to how item conditions are managed. Because the read-only attribute can't use an authorization scheme directly, you can use the APEX API `APEX_UTIL.PUBLIC_CHECK_ AUTHORIZATION` to determine whether a user has the rights to edit the data. This API takes a parameter of the authorization scheme name and runs the verification, returning a Boolean result that can be used in PL/SQL logic.

Although we could go and apply the read-only condition to each individual item in each region, there is a way to make an entire region read-only.

Here are the steps to use the read-only attribute and the API just discussed:

1. Navigate to and edit the regions indicated in Table 9-1 by clicking the region name on the respective page.

Table 9-1. Items That Require the Read-Only Attribute

Page Number	Page 210	Page 220
Region to Update	Manage Tickets	Ticket Details

2. In the **Read Only** section, set **Type** to **PL/SQL Function Body**, as shown in Figure 9-25. Set the value for **PL/SQL Function Body** to the following:

```
RETURN NOT APEX_UTIL.PUBLIC_CHECK_AUTHORIZATION('access control - edit');
```
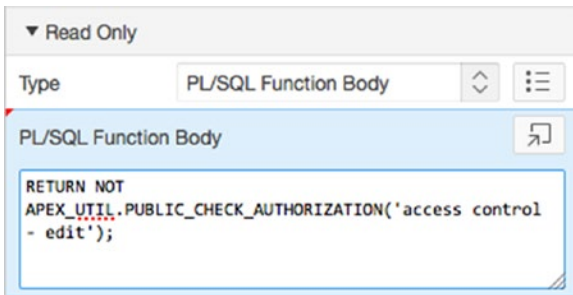

Figure 9-25. Setting the region to be read-only using the Read Only attribute

When you run the application as Martin, information about a ticket on page 210 shows data without the confusion of form elements. Authenticating as any other user shows the data in form elements and displays the corresponding buttons. Results of the read-only view are shown in Figure 9-26; compare them to the form in edit mode, shown in Figure 9-27.
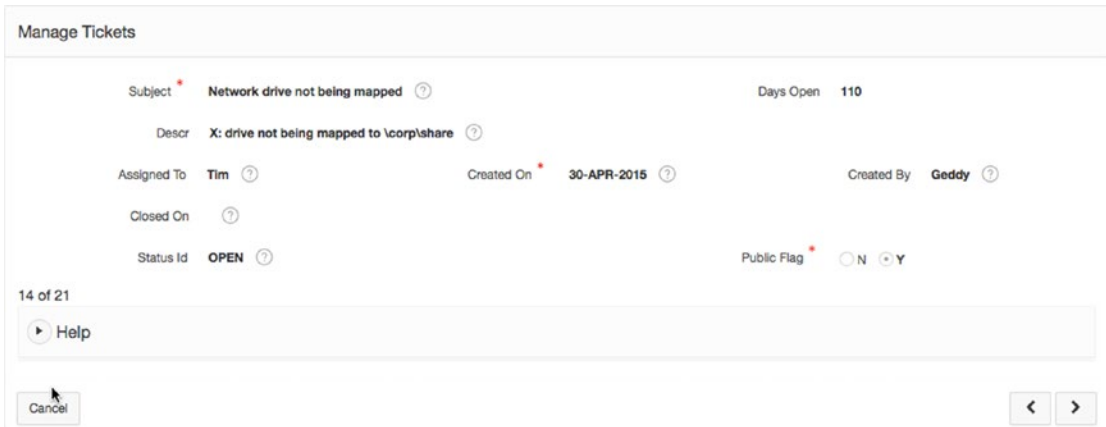
Figure 9-26. Ticket record in read-only mode

Figure 9-27. *Ticket record in edit mode*

Data Security

At this point, the majority of the application is relatively secure. What you lack is data security applied to segregate the data between application users. Any authenticated user can see and make changes to any other user's records. APEX doesn't provide a built-in construct for securing data. APEX does support and work well with other Oracle technologies that do secure data, such as Virtual Private Database, Oracle Label Security, and Transparent Data Encryption.

Although there are a number of ways to deal with data segregation and security, one of the simpler methods is to use a view to enforce the data available to a user in place of all references to the base table. This method is effective and works with all versions of the Oracle database. The process works by adding a securing function to the view that uses the current APEX user name, filtering out the data from other users.

To implement this data security, you will run a script that creates a new view named TICKET_SECURE_V and then re-create the other two views, TICKET_ACTIVITY_V and TICKET_V, so they point to the secured view rather than to the TICKETS table directly. Then you will make modifications to the other key components of the pages that access ticket data so as to also use the new secure views. Here are the steps:

1. Locate, upload, and run the script ch9_data_security_script.sql. Refer to Chapter 4 if you need step-by-step instructions. You should see three rows in the results report, all of which complete successfully.

2. Once the script completes, run the application and navigate to the Analysis page. You should notice that only tickets or ticket details that are assigned to the user you're logged in as appear.

Next, make changes to the source of several other pages so they reference the new secure objects you just created:

3. Edit **Page 200** of the application.

4. Select the **Tickets** report by clicking its name.

5. Locate and open the file ch9_report_p200.txt, and then copy its contents into the **SQL Query** text area, replacing everything that is there. **Save** your changes.

6. Run **Page 200** and notice that you can only see the tickets that are assigned to the current user.

You need to make a similar change on the Manage Multiple Tickets page:

7. Edit **Page 230** of the application.

8. Edit the **Manage Multiple Tickets** report by double-clicking it.

9. Locate and open the file ch9_report_p230.txt and copy its contents into the **SQL Query** text area, replacing everything that is there. **Save** your changes.

10. Run **Page 230** and notice that you can only see the tickets that are assigned to the current user.

Finally, you should also apply this rule to the chart, because it's still allowing you to see the status from all records in the system, which is inaccurate:

11. Edit **Page 500** of the application.

12. Under the Tickets by Status chart, edit **Series 1** by clicking its name.

13. Locate and open the file ch9_report_p500.txt and copy its contents into the **SQL Query** region, replacing everything that is there. **Save** your changes.

14. Run **Page 500** and notice that the chart only reflects the status of either unassigned tickets or tickets that are assigned the current user.

This is a huge leap forward in data security, but you're not quite finished. You may have noticed that if you edit one of the records on page 210, you can use the Next (>) and Previous (<) buttons in the lower-right corner to see records that belong to other users. Thus, you need to plug this security hole as well:

15. Edit **Page 210** of the application.

16. The location of the process **Get Next or Previous Primary Key Value** is shown in Figure 9-28. Edit the process by clicking its name.
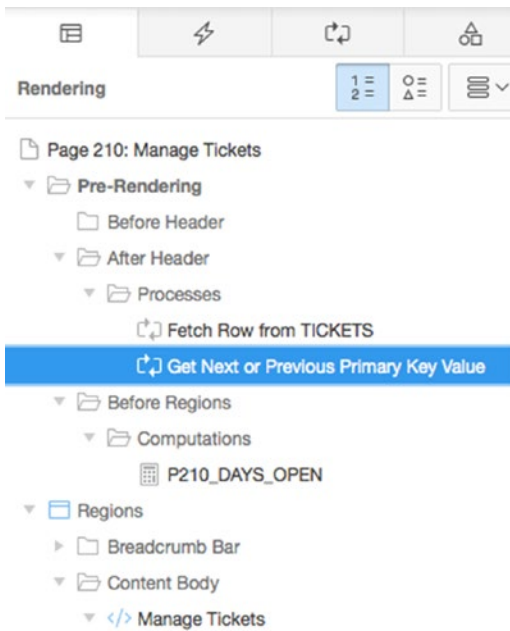
Figure 9-28. *The location of the Get Next or Previous Primary Key Value process*

17. Change the value of **Table Name** to TICKETS_SECURE_V, as shown in Figure 9-29. If you use the LOV, make sure you search the View tab. Click **Save**.
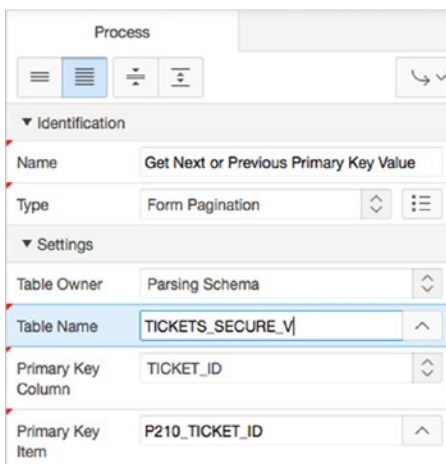

Figure 9-29. *Update the source for fetching the next record*

Now all of your data is secured based on who is signed on to the system. Or is it?

Session-State Protection

One of the most common ways to compromise a web application is through a form of attack known as *URL tampering*. You don't need to be a programmer or hacker to launch this type of attack, all you need to do is alter the URL in your browser. APEX introduced the session-state protection feature in release 2.2. When enabled, it adds a checksum value to the URL. If any portion of the URL is altered, the resulting checksum doesn't match what is expected, and the page simply won't render.

By default, APEX 5.0 enables session state protection at the application level and applies item-level session state protection to any forms and reports that are built using the wizards. Thus, if a user were to tamper with the APEX URL, it would prevent them from being able to see tickets assigned to other users.

Run the Tickets report on page 200 in the application. Hover your mouse over the Edit icon and examine the URL. Notice the &cs= portion of the URL. The &cs= parameter is the checksum that was automatically generated by APEX. Alter the value for P210_TICKET_ID in the URL, or remove &cs= and everything to the right of it, and try to run the page. You will receive an error message similar to that shown in Figure 9-30.
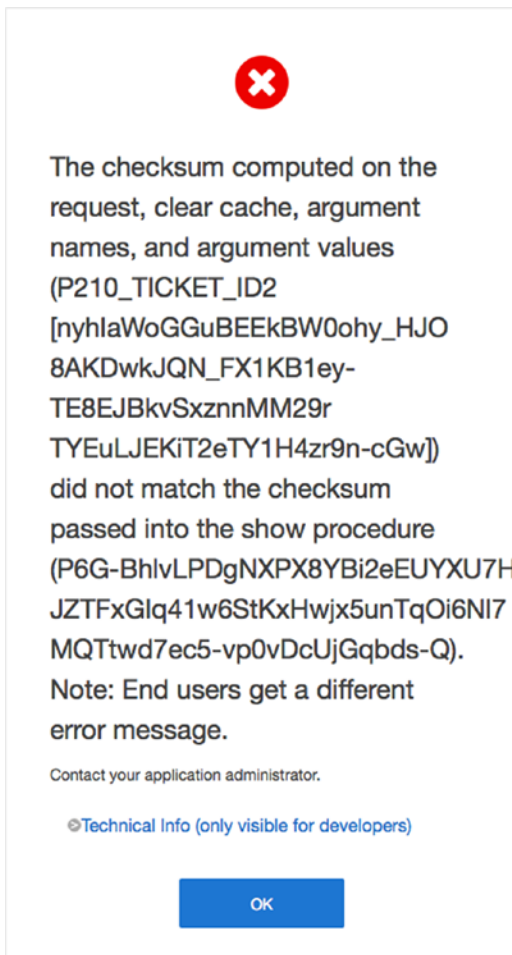
Figure 9-30. Checksum error message as a result of URL tampering

Summary

In this chapter, you've applied new security to the Help Desk application by utilizing the key features of APEX. You implemented a new custom authentication scheme to allow control over users who access the sensitive parts of the application. You also reviewed conditional security with both authenticated and un-authenticated individuals and added parameters to allow the application to be used by both.

Congratulations! You've just written your first APEX application. With the skills you've learned thus far, you have the capability to create just about any type of system you'll need. Take a moment to bask in the glory of your achievement and then get ready to learn how to migrate your application from development to other systems.

■ ■ ■ ■

Application Bundling and Deployment

The concept of application bundling and deployment is something developers should consider from the beginning when designing an application. In the case of APEX, built-in facilities help make the job easier. When it comes to application deployment, there are various ways to accomplish the same end goal, and no two IT organizations do it exactly the same way. This chapter will discuss the tools APEX provides to help you bundle and deploy applications as well as how to use them in a very APEX-centric way.

■ **Note**   Your organization may already have a standardized way to achieve many of the things being introduced in this chapter. Before implementing any of these methods, check and make sure you're not reinventing the wheel.

Identifying Application Components

Your APEX application consists of more than just the application export itself. There are underlying database objects, images, Cascading Style Sheets (CSS), and JavaScripts. And these components may or may not be stored on the same server as APEX, let alone stored in the APEX metadata repository. In essence, you need to know how to assemble everything it would take to instantiate your application from scratch. Therefore, it's important to understand all the components that make up your application, where they're stored, and how to bundle them in a way that makes migration easier.

You can break the various components into roughly four main groups:

- *External files*: Your application may access files that don't reside in the APEX repository. For instance, your company may have a common set of CSS and image files that are used by several websites in order to maintain a standard look and feel.

- *Database objects*: These include all the tables, views, PL/SQL objects, and any other database objects used by your application. Most of the time these reside in your application's "parse as" schema.

- *APEX-based files*: These are files that have been uploaded into the Files section of an application's supporting objects. They may include images, CSS, JavaScript, static files, and so on, and are stored in the APEX repository.

- *APEX application export*: This is the core of the APEX application, containing the pages, regions, items, validations, and so on.

When it comes time to deploy an application, each of these types of files needs to be treated a bit differently. The following sections address each file type and how to obtain the most recent version for migration to an alternate platform. Later, the chapter will discuss using the Supporting Object feature of APEX to bundle the appropriate items into the application export.

External Files

As already mentioned, external files exist outside of the APEX metadata repository and usually outside the Oracle database as well. In the majority of cases, these files are placed in a directory structure on the application server that provides the HTTP services for APEX. Usually they're placed in a directory under the document root (docroot) of the domain that is servicing APEX requests. Because they exist outside of APEX, they can't rightly be included in the supporting objects of an application, and thus need to be handled separately from the other file types.

You must keep careful track of what files your application uses and whether those files have changed during the development of your application. Another area of concern is whether other applications, APEX or otherwise, use these same files.

For instance, version 1 of your application may reference a JavaScript file that is stored on the application server. During the development of version 2 of the application, you may have made changes to that file that need to be moved from the development server to QA or production. But what if your colleague is working on another application that uses the same JavaScript file? You must be very careful about what you change and how you deploy it so as not to inadvertently affect other systems.

When migrating these files from a development to a QA or production environment, you likely will need to work with the people who are in charge of maintaining the application-server tier. They probably have a process in place for planning the migration from one tier to another.

If you're working on your own and are the sole person in charge of the file migration, it's good to get into the habit of maintaining a backup copy of the files you're replacing, just in case something goes wrong. You can do this simply by renaming the file currently in use to include some type of identifier for the version. Including the date in the filename works well for this. In Linux, the command looks something like this:

```
mv my_old_file.js my_old_file_2015_09_17_12_37.js
```

If you're using a source code control system and are tagging the file versions that are moved to production, you may not need to take this extra step.

The key is making sure you can recover from any issues that may arise from overwriting a file. There's nothing worse than bringing a system to its knees with no easy way to get back to the previous state.

Database Objects

It may seem that database objects should be straightforward, considering that they exist in Oracle and the SQL code for their definition can be recreated relatively easily. And for a brand-new application, this assumption is fairly accurate.

However, the minute an application goes live, if you need to change the table structure, you can't simply replace the underlying tables with new versions. The users have probably entered or manipulated data in the system, and it's your job to make sure that when new versions of the system are rolled out, the integrity of the data is maintained.

New Applications

When you're deploying a brand-new application, a couple of tools can help you generate the scripts for the underlying database objects. The Utilities menu in the APEX SQL Workshop contains a Generate DDL tool, which does exactly what its name implies. If you run it against your application's "parse as" schema, it allows you to generate a SQL script containing the underlying database objects.

As shown in Figure 10-1, the wizard asks which of the available schemas you'd like to use as the basis for the generated script.
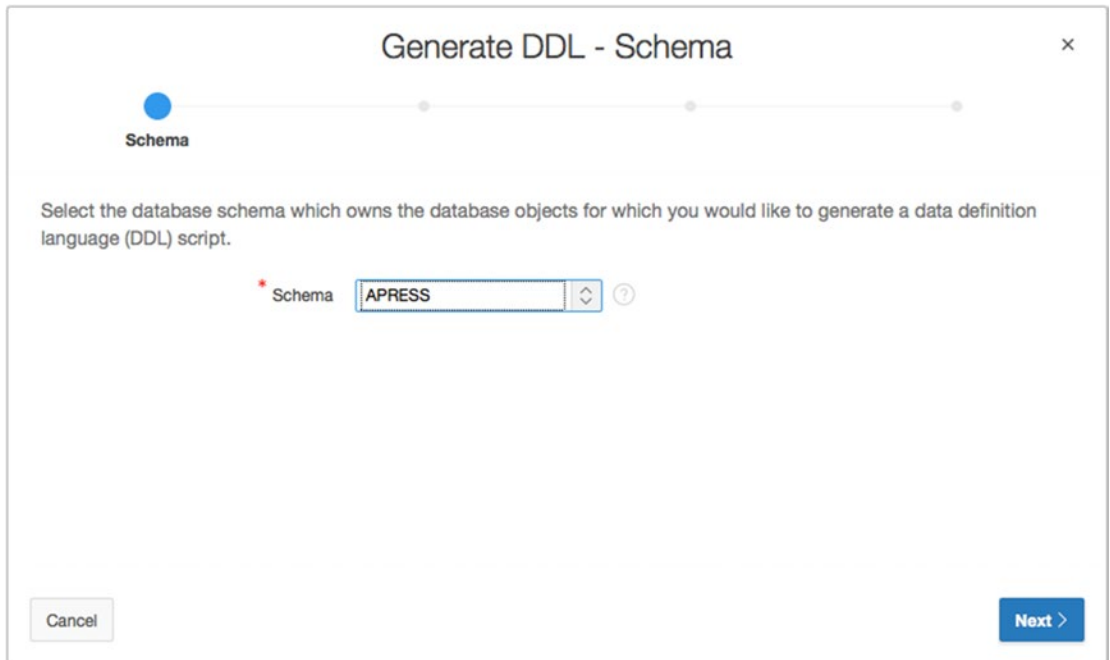
Figure 10-1. *Choosing the schema for which to generate object-definition scripts*

The wizard then lets you choose what types of database objects to include in the script (see Figure 10-2). Make sure you select all the object types that are used by the application. Selecting Check All gives you the option of generating scripts for all objects in the selected schema. At this point you may also decide whether you wish to show the generated script inline so you can copy and paste it, or save it as a script file to the APEX script repository.
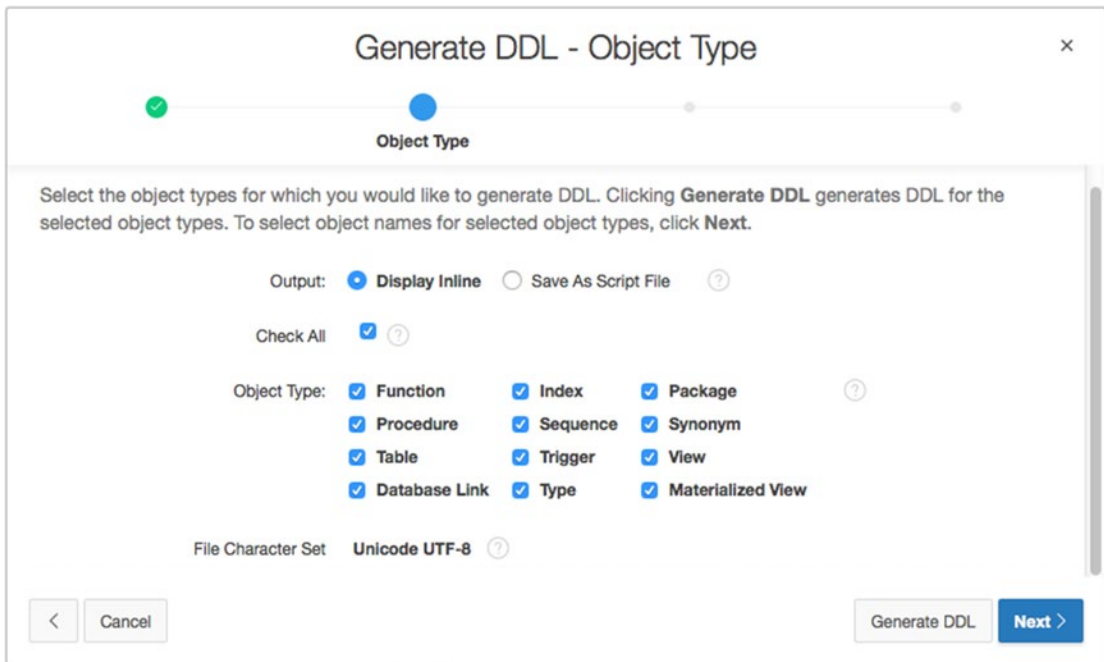
Figure 10-2. *Selecting the object types in the Generate DDL wizard*

The next step of the wizard (see Figure 10-3) lists all the objects that match the types you selected in the previous step. You can be as selective as you'd like about which objects to include. Your particular application may only use a subset of the objects within a schema, so you only need to choose those when generating the DDL.
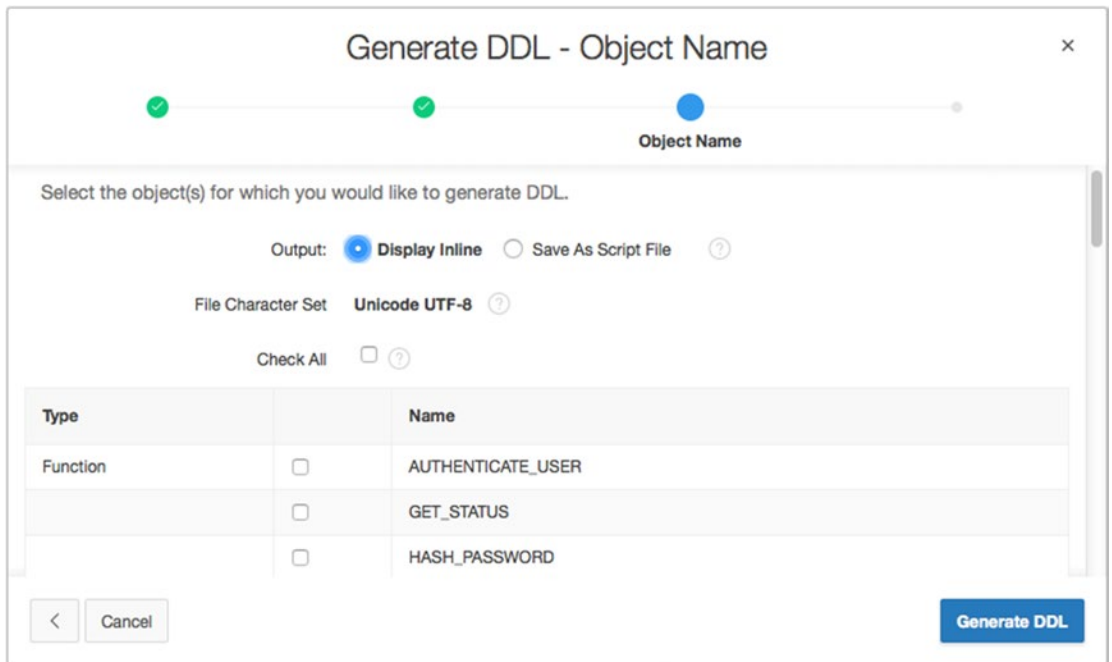
Figure 10-3. *Choosing the specific objects desired in the Generate DDL wizard*

■ **Note**   If you find yourself in a situation in which several applications are sharing the same underlying schema, you may want to apply a naming convention to the database objects so you know which objects relate to which application. A common database-object naming convention is to introduce a three-letter prefix to the object names. For instance, the table USERS for the Help Desk application would become HDA_USERS. Again, check with your company to see if it already has an object naming convention.

If you've chosen to save the script to the APEX script repository, the next step allows you to enter the name of the file to be created and a description, as shown in Figure 10-4.
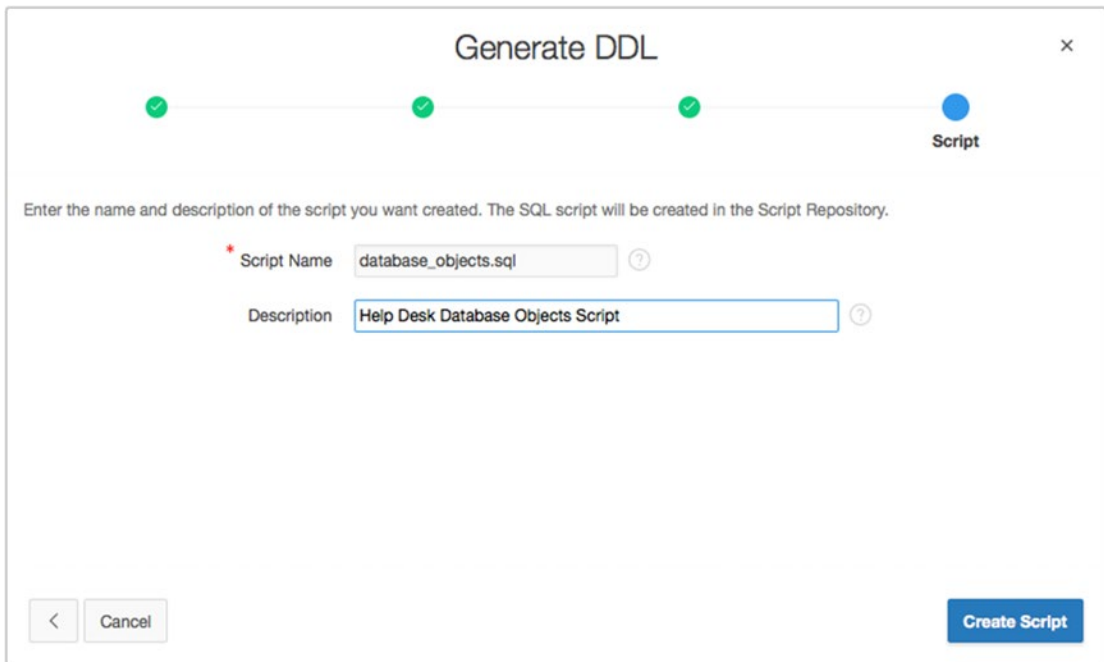
Figure 10-4. *Naming the script being created by the Generate DDL wizard*

At this point, the script is generated, containing all the chosen objects. The generation engine does a good job of creating objects that are dependent on other objects in the correct order so that no errors will occur when the script is run. However, it's always a good idea to test these scripts to make sure everything runs smoothly.

Oracle's SQL Developer product also has a tool that lets you generate DDL for a selected schema. Figure 10-5 shows the splash screen of the SQL Developer Database Export tool.
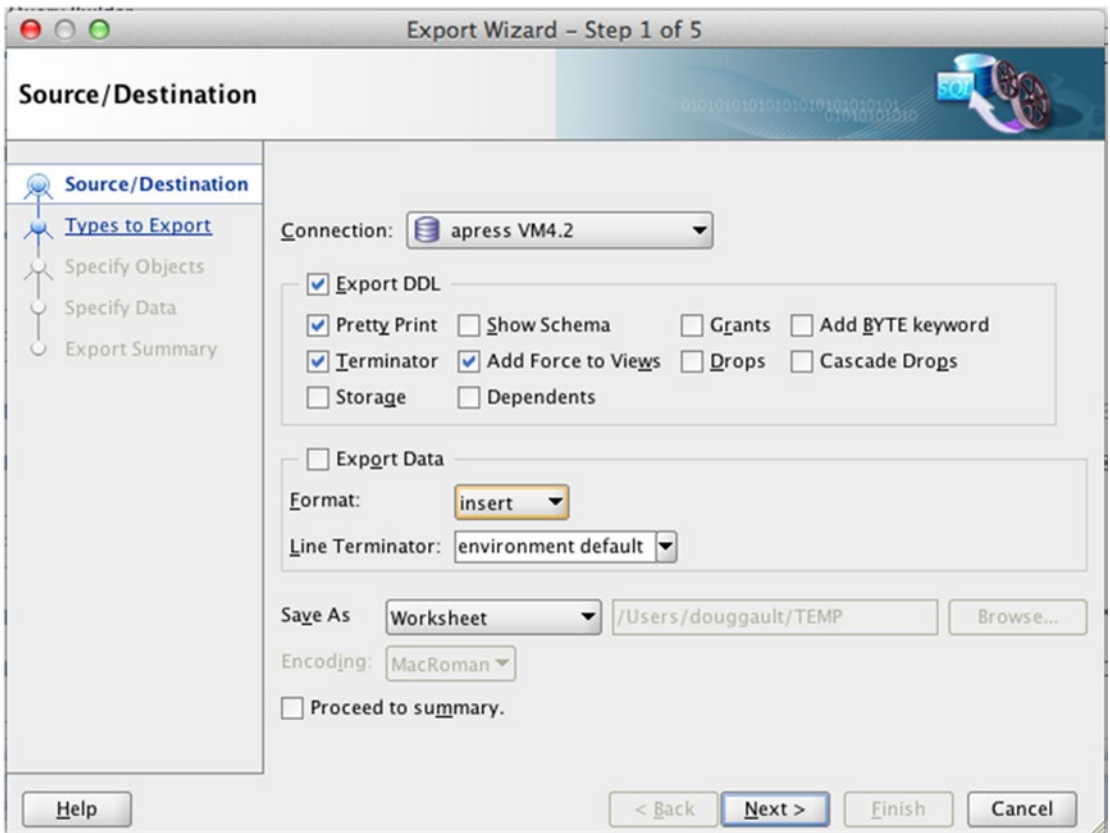
Figure 10-5. *The first screen of the SQL Developer Database Export tool*

This tool is very similar to the APEX wizard, but it gives you more control over the format and contents of the output, including whether to include schema names, storage clauses, grants, and so on. Another benefit of SQL Developer is the ability to export the data that exists in the tables. This comes in very handy for seed data that is needed for the system to function properly.

Whether you choose to use the APEX-based tool or SQL Developer, generating the object-creation scripts for a new system is straightforward.

Existing Applications

For applications that have already been released into a production environment, the deployment process can be much more complex. You need to take into account the version that is in production and how the underlying database structure may differ from the version you've created in development and are ready to deploy.

Luckily, there are tools available to help identify the differences between two schemas. These tools can also generate the necessary DDL scripts to implement the differences.

However, the unfortunate truth is that although the APEX SQL Workshop utilities do include a schema-compare tool, it has some severe limitations. For one, both schemas that are being compared must be available from the same workspace. This isn't possible if your production schema exists on a separate server, as it often does. The second limitation is that the APEX-based comparison tool identifies the objects that are different, but it doesn't say how they're different, nor does it generate the DDL that would be required to synch up the schemas.

For this type of functionality, you have to rely on external programs or scripts. The following list mentions a number of options, all of which can generate the scripts required to synchronize the production environment's database objects structure with the changes you may have introduced in development:

- *SQL Developer*: Oracle's own product can run a full schema comparison between two separate schemas on separate servers and generate a script that synchronizes one schema with another. Older versions of this tool suffered from some problems, but as of SQL Developer version 3.2, the comparison engine has been significantly upgraded, and the generated scripts are solid.

- *Oracle Enterprise Manager*: If you have the Change Management Pack and Oracle Enterprise Manager (OEM), then you can compare schemas and generate a synchronization script. However, developers are very rarely given access to OEM because it's more of a database administration tool and would potentially give them access to several sensitive utilities administrators would rather us not have access to.

- *Schema Compare for Oracle*: Red Gate Software has taken its extensive experience in creating tools for the SQL Server market and turned its attention to the Oracle database market. The result is a tool that allows you to compare, view, and generate synchronization scripts between two Oracle schemas. This is probably the best third-party tool on the market, but the one downside is that it only runs on Windows.

- *TOAD for Oracle*: TOAD (which originally stood for Tool for Oracle Application Development) is a tool written and distributed by Dell's software division (formally Quest Software). Although it can do a lot more, the schema-comparison tool that's available as part of the DB Admin module is quite sophisticated and will generate very clean and accurate scripts.

Whichever tool you use, the output is a script that, when run against the production environment, executes the required DDL to alter the underlying database objects and bring them in line with what was created in your development environment.

However, none of these tools take into account the data that may reside in the tables that are being altered. Be very careful before you implement any of the generated upgrade scripts, understand what they may do to the underlying data, and mitigate any risks of data loss or corruption.

This subject is huge and is beyond the scope of this book. There is no automated solution to the problem of data migration between versions. More often than not, it boils down to handwritten scripts and heavy testing.

APEX-Based Files

APEX provides the ability for developers to upload static files into the APEX metadata repository as part of an application's shared components. Figure 10-6 shows the Files section of the Shared Components page. There are two types of files represented: those that are tied directly to the application and those that are available to all applications within the current workspace.
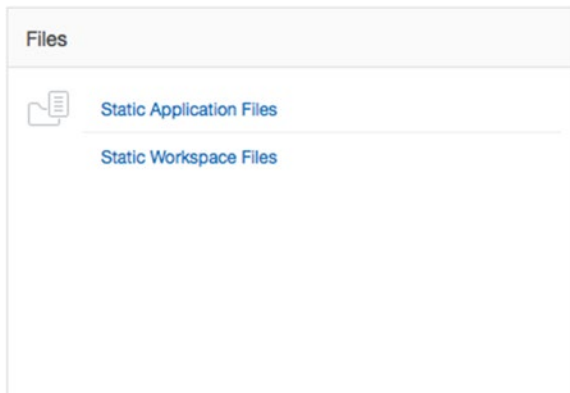
Figure 10-6. The Files section of an application's Shared Components page

Static Application Files may be any file type, such as CSS files, JavaScript files, images, documents, and so forth, that you may need as part of your application. They can be referenced from within your application by prefacing them with the #APP_IMAGES# replacement variable.

Because these items are tied directly to a specific application, they are ***automatically included*** when you perform an application export (as discussed later in this chapter).

Static Workspace Files may also be any file type, but instead of being tied directly to an application, they are made available to all applications within the current workspace by using the #WORKSPACE_IMAGES# replacement variable.

Even though the Static Workspace Files are considered shared components, they aren't included in the application export. This means that you'll need to migrate these items separately. You'll be able to do this from the same screen that allows you to upload them. You can either choose to download and migrate individual files using the download link associated with that item, or choose to simply click the Download as Zip button, after which you're presented with a dialog that lets you export a zip file of all the Static Workspace Files. Both options are seen in Figure 10-7.
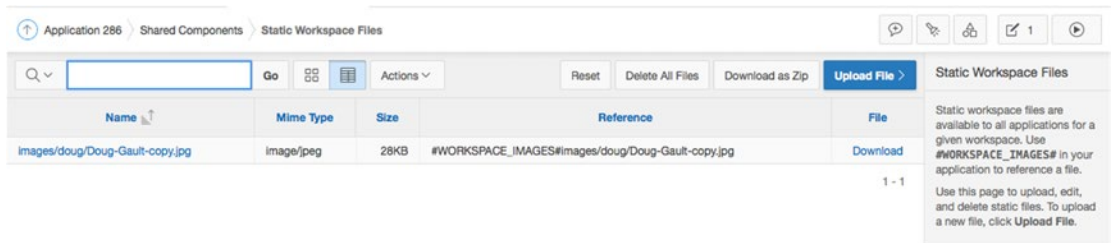
Figure 10-7. The Static Workspace Files page allows for individual or bulk download

Files that are uploaded as shared components will likely be ones that you reference throughout your application. They may represent portions of your theme, such as images for tabs or buttons, or they may represent icons that you use to show status or that, when clicked, allow end users to edit rows of data.

One key differentiation to make is that the files uploaded to this area should not be directly related to the application's data. Things such as product images, images of employees, and the like should be stored in the application's "parse as" schema alongside the data to which the image is related.

APEX Application Exports

Overall, the APEX application export is easy to execute. The interface includes a process designed to generate scripts for recreating APEX applications.

It's important at this point to know what an application export includes and what it doesn't. We've already discussed the fact that the underlying database objects aren't included, and neither is anything that is uploaded to the Static Workspace Files section of the shared components. But all other shared components, including Static Application Files, are included in the export file.

It's worth mentioning that all configured and assigned shared components are included in the APEX application export, whether they're being used by the application or not. For instance, there can only ever be one authentication scheme current for an APEX application, but more than one authentication scheme may be configured and assigned to the application. The same is true for user-interface themes.

Although this isn't strictly a problem, it's good practice to delete any shared components that aren't being used by the application so that the size of the application export stays as small and as manageable as possible. Most shared components provide a utilization report so you can see whether they're being used.

The application export capability is located on the Application Builder main page in the Icon menu at top of the page, as shown in Figure 10-8.
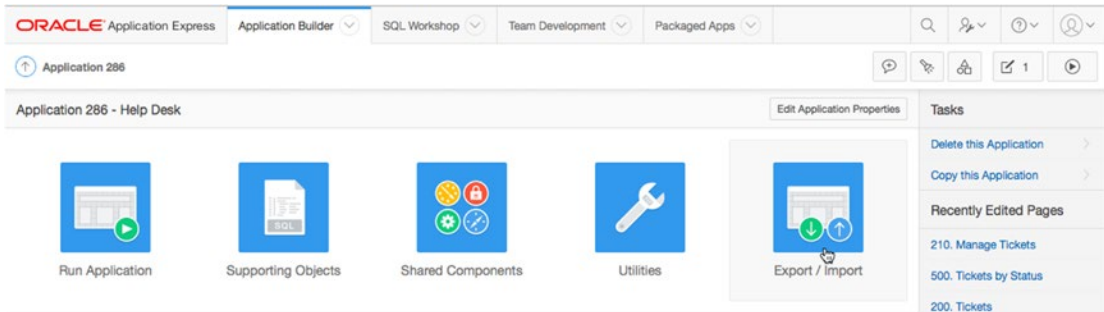
Figure 10-8. *The Export option is located in the Icon menu at top of the page*

When you initiate the wizard, it first prompts you as to whether you wish to import or export an application. Once you select Export, you are taken to the Export page, as shown in Figure 10-9.

Figure 10-9. *The Export Application page*

You can choose the application to export by using the select list near the top of the page. The Export Application section allows you to dictate how, in more general terms, the application should be exported. It includes these options:

- *File Format*: It doesn't relate to the target platform but instead to how the file is generated with regard to carriage returns and line feeds.

- *Owner Override*: Allows you to override the currently assigned "parse as" schema by either entering or selecting one.

- *Build Status Override*: Lets you select which build status is the default when the application is imported. The default is Run and Build Application, but you may set the status to Run Application Only.

- *Debugging*: Dictates whether the application is installed with debugging enabled or disabled by default. Debugging is useful for applications in development. However, as a best practice, you should turn off debugging for production applications so as to prevent users from viewing things that may only show up while in debug mode.

- *As Of*: Allows you to export the application as it existed a number of minutes ago. For this feature to work, Flashback Query must be enabled by the DBA at the database level. The amount of time you may flash back is controlled by the UNDO_RETENTION parameter at the database level.

■ **Note**   Although you can select default values for the settings in the Export Application section, it's important to understand that they can be overridden when the application is imported. At this point, you're merely setting the defaults for the import.

In the Export Preferences section, several options allow you to decide what is included in the application export. The following options are available:

- *Export Supporting Object Definitions*: Dictates whether any supporting objects that have been uploaded are exported with the application. See the "Supporting Objects" section later in this chapter for a full description.

- *Export Public Interactive Reports*: Dictates whether report definitions saved by end users and marked as public are exported as part of the application.

- *Export Private Interactive Reports*: Dictates whether report definitions saved by end users and marked as private are exported as part of the application.

- *Export Interactive Report Subscriptions*: Dictates whether user subscription information for interactive reports is exported as part of the application.

- *Export Developer Comments*: Dictates whether any comments developers have entered against APEX components are exported as part of the application.

- *Export Translations*: Dictates whether translation mapping information is exported as part of the application. Translation text messages and dynamic translations are always included in the application export, regardless of the setting chosen here.

- *Export with Original IDs*: Dictates whether the export file should contain the application component IDs as of now or as of the last import of this application.

Once you've chosen the appropriate settings, click the Export button to produce the application export. You're prompted to save it to your local machine. The export file name consists of the letter *f* followed by the application ID, with a `.sql` extension. For example, an application with an ID of 9239 is named `f9239.sql`.

The downloaded file contains a large text script that defines all the contents of the application built in APEX. Along with the application pages are the shared components, including authentication schemes, authorization schemes, themes in the application, UI settings, reports, and so on. This script can, in turn, be imported into the same workspace, a different workspace, or even a different server.

Supporting Objects

The application export captures the complete definition of your application, including most shared components, but it doesn't contain everything you would need to completely reconstitute your application on another server. However, APEX provides a feature that allows you to bundle the scripts for things such as the underlying database tables inside the application export. This feature is called Supporting Objects.

The Supporting Objects feature actually gives you a great deal more functionality than that. It also provides the ability to create and control the installation, as well as to upgrade and uninstall anything that can be scripted using SQL.

You reach the Supporting Objects management interface by navigating to the Shared Components page for an application and selecting the Manage Supporting Objects option from the Tasks menu. Figure 10-10 shows the Supporting Objects home page.

Figure 10-10. *Supporting Objects management home page*

The page is broken down into several regions. The Summary region at the top shows what is currently defined in the supporting objects, and the three regions below (Installation, Upgrade, and Deinstallation) allow you to edit the scripts and define the actions that are available during each phase.

Clicking any of the links takes you to a tabbed definition page, as shown in Figure 10-11.
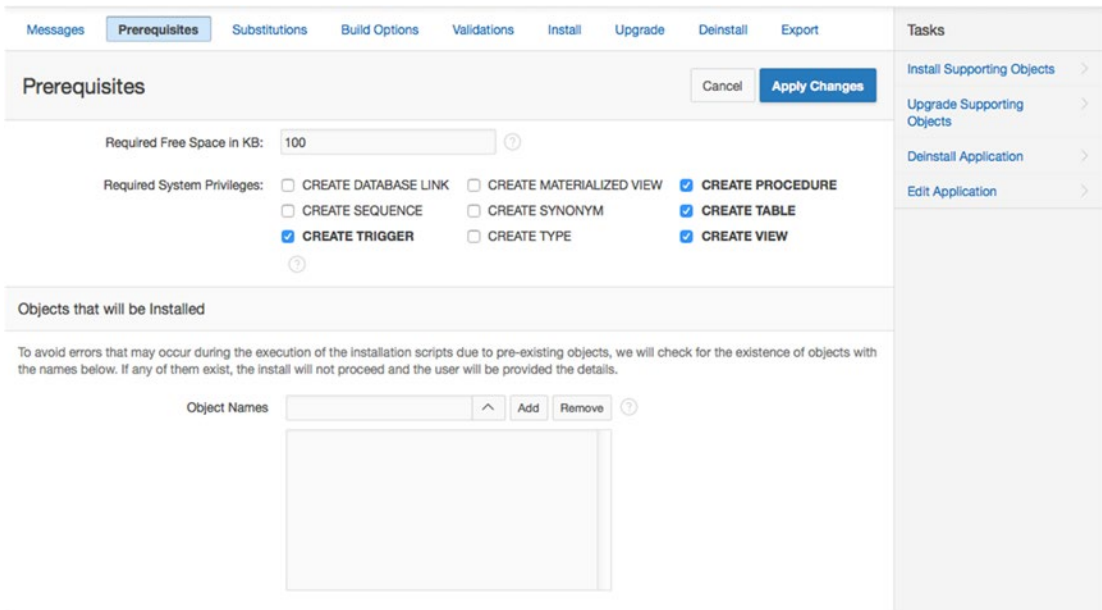
Figure 10-11. *Supporting Objects tabbed definition screen*

Working through the tabs on this page lets you define the actions that are taken and any scripts that should be run during each of the three phases. Although we won't show a picture of the contents of each tab here, in this section we will walk through each of them and discuss their contents and purpose, saving the Messages tab for last.

Prerequisites

This section defines what built-in checks should be run to ensure that the database schema into which the application is being installed has the appropriate privileges. You can provide a minimum amount of space that is required for the application to work correctly. At installation time, the "parse as" schema's default tablespace is checked to be sure the appropriate amount of space is available. You can also check to make sure the schema has any of the following specific privileges:

```
CREATE DATABASE LINK
CREATE MATERIALIZED VIEW
CREATE PROCEDURE
CREATE SEQUENCE
CREATE SYNONYM
CREATE TABLE
CREATE TRIGGER
CREATE TYPE
CREATE VIEW
```

The bottom region of the page allows you to list all objects that will be created by the supporting object-installation scripts. At installation time, if any of the listed objects already exist, the install won't proceed, because there could be a clash. The user installing the application is given the details of which objects are found to already exist.

This section may seem a bit limited in its scope, but the Validation section, discussed later, allows for more free-form prerequisite checks.

Substitutions

This section provides the ability to allow the installing user to define the value for application-level substitution strings at install time. Although substitution strings are meant to be used like static variables, you may not always know what the value of these strings should be prior to installation. From this interface, you can choose which substitution variables you want to let the installing user define and what the prompt for each variable should be.

Substitution variables aren't used very often, so this feature is also unlikely to be used. However, it's good to know that it's there if you need it.

Build Options

In Chapter 13 we will speak about build options and the fact that they can be used to exclude or hide assigned functionality. This section allows you to select whether build options you've defined are available to the installing user. By selecting a build option, the user will be prompted as to whether they wish to include or exclude the functionality associated with the build option.

Most of the time, when moving applications to production, you want to exclude all build options.

Validations

This section lets you define any number of pre-installation validations to be run. These validations are similar to normal page validations and allow full control over whether the application installation can proceed. You may have as many validations as you wish, and the validations may be conditional as well.

If any validation fails, the installation is halted, and the user is presented with the error message(s) defined in the failing validation(s).

Install

This is the core of supporting objects and is where you define what scripts to run and in what order to install all the objects your applications need to work properly. Here you can create and manage scripts that install database objects, workspace or application images, CSS files, static files, and so on. Depending on the type of scripts you're including, you may be able to create them in different ways.

When it comes to scripts that create the underlying database objects, you've probably used a tool such as SQL Developer or the SQL Workshop's Generate DDL tool to generate a script to a file.

You can choose to upload a pre-created script file, to create the script from scratch, or to create a script based on the definition of a database object in the application's "parse as" schema. You do so via the Create Script wizard shown in Figure 10-12.
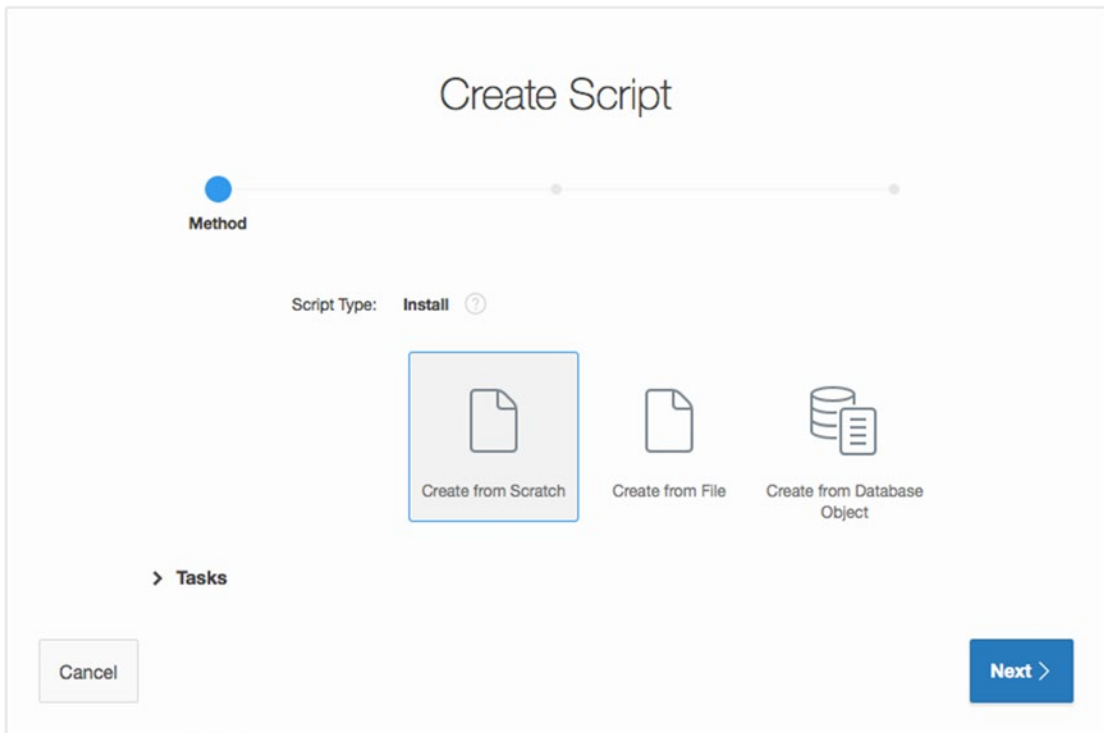
Figure 10-12. *Create Script wizard*

Choosing Create from Scratch presents you with a script-editing screen where you can type in the script steps from scratch or copy and paste the script from a text editor. However, if you already have the script stored in a file, you may want to use the Create from File option, which allows you to upload the script from your local computer.

The Create from Database Object option presents you with a wizard that allows you to choose which objects that exist in the application's "parse as" schema to include in the script. Once you choose the objects, the wizard builds the creation script for you and presents it for editing. You can then save the script as part of the Supporting Objects installation sequence.

The main difference when creating scripts from the database objects is that APEX keeps a record of which objects you chose and allows you to go back and refresh the script against the underlying "parse as" schema, add new objects you may need, or even delete ones you don't. This is potentially much more useful that using an external script-generation tool as it integrates directly into the application export. However, some IT departments require the underlying database object–creation scripts to be separated from the application export as a point of control. Again, make sure you know what your company's procedures are for moving applications to production and follow the standards.

However it is generated, once a script has been created, you're allowed to alter the script's name, its sequence of execution, and the condition under which it will be run.

Whether you have several scripts, one for each object or object type, or one large script that creates all the required objects is completely up to you. Just make sure that if you choose to have several scripts, you test their execution in the order they're listed in the interface to make sure any dependencies are accounted for.

Upgrade

The Upgrade tab is very similar to the Install tab, as it allows you to create or upload scripts. But in this case, the scripts are used to upgrade an existing application's supporting objects if the installer finds that the application is already installed in the workspace.

The installer does this by letting you write a query to check for the preexistence of supporting objects in the schema. If the query returns one or more rows, then the upgrade script set is run in place of the install script set.

Deinstall

This section allows you to define a single script that drops the objects created by the install or upgrade scripts. When you generate install scripts for supporting-object files, API calls to deinstall these files are added to the deinstall script automatically. However, you need to add the necessary code to drop the appropriate database objects manually.

Export

The Export tab simply lets you set the default for whether the supporting objects are included when you export the application. This option is also available on the Supporting Objects main screen.

Messages

The Messages page gives you control over the verbiage presented to the installing user during the installation of the application. The section text that you can edit is as follows:

- *Welcome*: After successfully importing and installing an application definition, the installation wizard prompts the user to install supporting objects for the application. This message introduces the application and describes the actions of the installation scripts.

- *License*: If the use of this application requires the user to accept a license, enter the license text here. The user is prompted to accept the message before installing supporting objects. If there is no text for the license, this step is skipped in the installation wizard.

- *Application Substitutions*: Introduces the application-substitution prompts. It should probably state that these values aren't easily changed and to be sure of their values before entering them. If there are no application-substitution variables to be entered, this message doesn't display.

- *Build Options*: Introduces the build options that may be available for the user to select. If no build options are available, the step is skipped and the message doesn't display.

- *Validations*: Introduces the validations that will be performed prior to installing the supporting objects. If there are no validations, the step is skipped and the message doesn't display.

- *Confirmation*: Displayed just prior to the installation scripts being run and the configuration options being applied.

- *Post-Installation Success*: Shown after the application's supporting objects have been installed successfully with no errors.

- *Post-Installation Failure*: Shown after the application's supporting object scripts have run, but only if errors were generated. The user can view the errors that occurred.

- *Upgrade Welcome Message*: Provides a message informing the user that the installer has detected preexisting supporting objects and that the Upgrade wizard will now be run.

- *Upgrade Confirmation Message*: Presents a message prior to running the upgrade scripts to allow the user to choose whether to continue.

- *Upgrade Success Message*: Shown after the suporting objects upgrade script is run successfully with no errors.

- *Upgrade Failure Message*: Shown after the suporting objects upgrade script is run, but only if errors were generated. The user can view errors that occurred.

- *Deinstallation Message*: Presented just prior to running the supporting objects deinstallation script.

- *Post-Deinstall Message*: Presented just after running the suporting objects deinstallation Scrip.

■ **Note**   Because all script types are standard SQL and PL/SQL, you have the option of writing quite complex logic that can decide within the script what steps to take. However, there is no interactivity or shared session state between the individual scripts, so you can't decide in the first script whether to run the second or third scripts. Every script in the set will be run regardless of the result of the previous scripts. Errors are shown only after all scripts have been run.

The process of building a packaged application that includes supporting objects can be daunting. The good news is that, in a standard IT environment, the scripts to migrate database objects are rarely processed using supporting objects. Although supporting objects are very useful, they tend to lend themselves to situations, such as shrink-wrapped software, in which applications are sent to remote sites where there is little or no direct interaction with the installing user.

For applications that are being developed and deployed in a single organization, rules and guidelines are probably in place for migrating applications to production. Make sure you check with your organization and adhere to those standards.

Importing

APEX applications can be imported by providing the application-export script. You can import into a different workspace or into the original workspace. The Application Import wizard is available from the Application Builder home page. Figure 10-13 shows the initial page of the wizard.
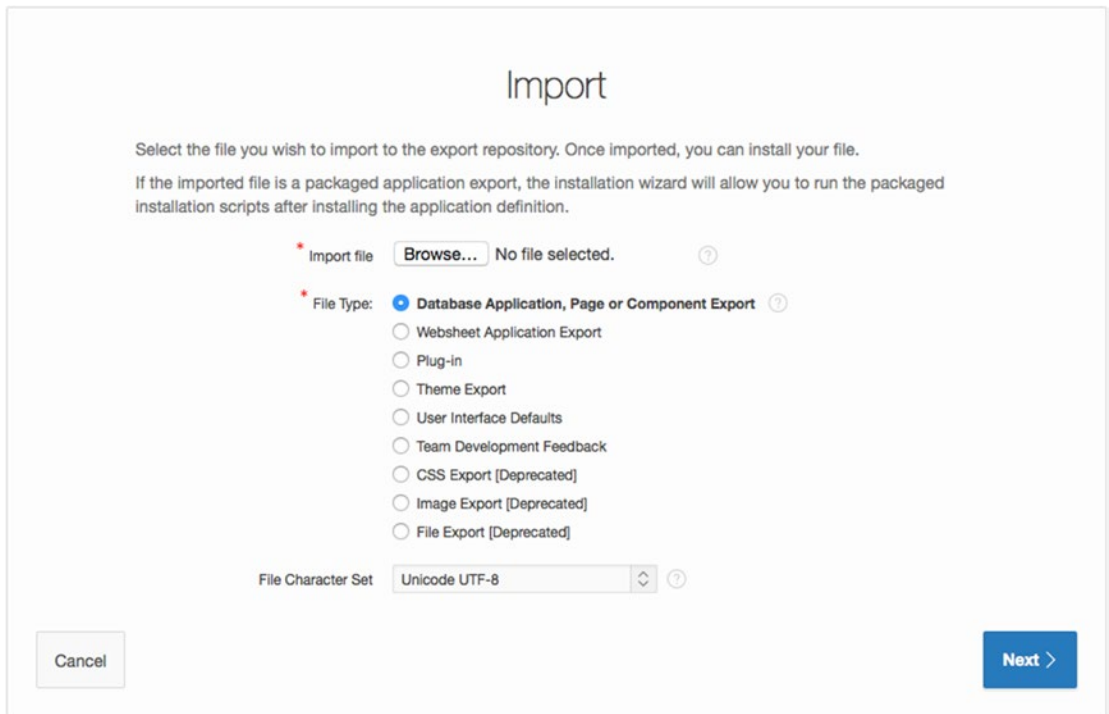
Figure 10-13. *Import file identified as a database application*

As you can see, the wizard allows you to import many different types of APEX export scripts. Make sure you choose the right type for the file you're trying to import. When importing an application export script, click Browse to choose the application export file, and be sure to choose Database Application, Page, or Component Export.

The page in Figure 10-14 indicates that the application export file has been uploaded from your computer to the server. Remember that the application file is a script. Although it has been uploaded at this stage, it hasn't yet been run; therefore, the application isn't installed.
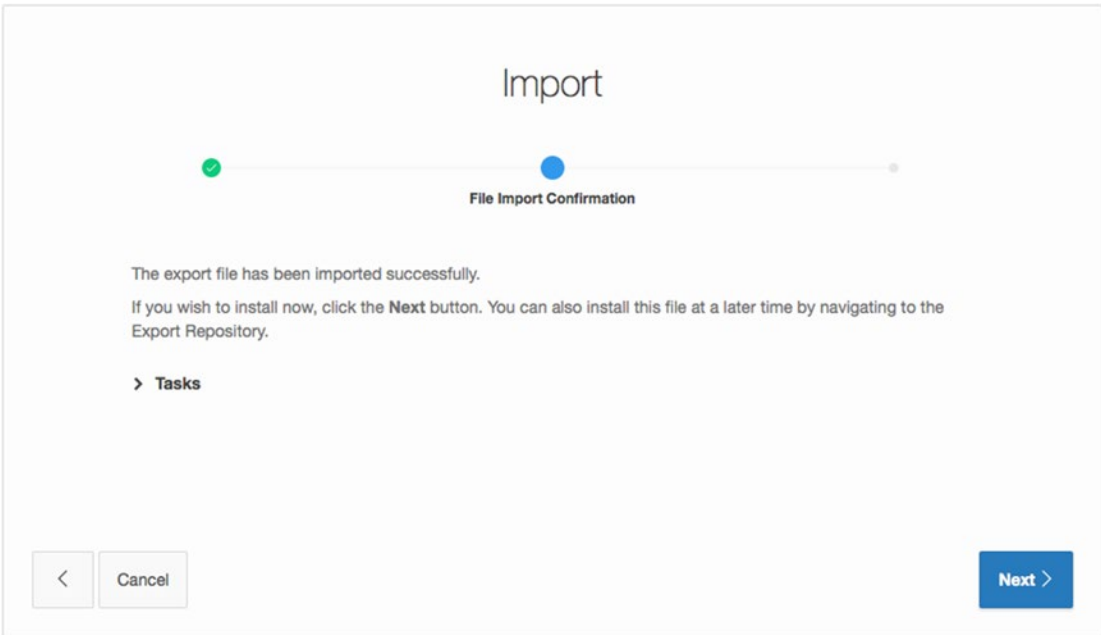
Figure 10-14. *File upload success. Continue to install the application*

Clicking the Next button initiates the steps to install the application into the current workspace. APEX prompts for a few key pieces of information, as shown in Figure 10-15.
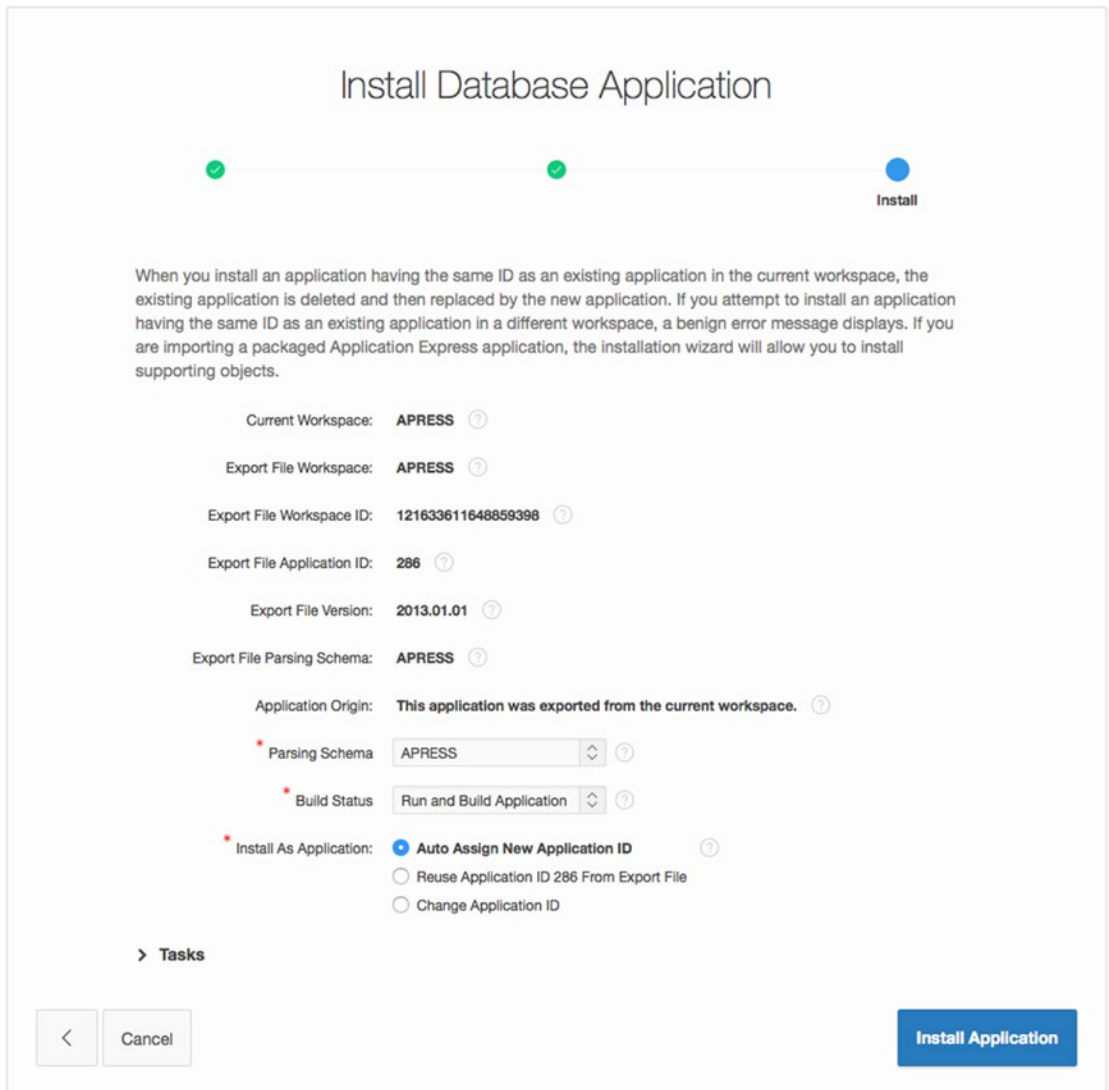
Figure 10-15. *Installing the application into the workspace*

At this point, choose the parsing schema and the build status, and then decide how to treat the application ID. The parsing schema can be any of the database schemas associated with the workspace. The build status lets the application be set to a runtime mode, which is useful for production environments; the default allows run and build (or edit) mode. The final option pertains to the application ID values; the default option is to assign a new application ID when installed, which lets the same application exist in the workspace multiple times—each time under a different ID.

If you choose to reuse the application ID from the export file or to change the application ID to one of your choosing, APEX checks to see if an application with that ID already exists. If an application with that ID does exist in the same workspace, you're prompted as to whether you wish to replace the application currently assigned to that application ID with the one you're importing. If an application with the selected ID exists but is in a different workspace, you're prohibited from using that application ID. This protects you from accidentally overwriting applications in other workspaces.

307

If the application has supporting objects, the next screen asks whether you want to install those supporting objects. It also gives you the option of previewing the supporting object scripts that will be run.

To continue installing the supporting objects, select the Yes radio button and click the Next button. The wizard then walks through all the steps that were set up when you created the supporting objects. It performs any prerequisite checks and validations and decides whether to run the install or upgrade scripts. The user is presented with any choices and options related to substitution strings and build options.

Finally, you're asked to confirm the installation (or upgrade) of the supporting objects. Continuing with the wizard runs the appropriate scripts. If there were errors during the scripts, the errors are presented to you to view. If there weren't any errors, you're given the opportunity to see the installation summary or to edit or run the application.

Summary

As you've seen, APEX has a robust, built-in migration capability. The export and import tools are easy to use and very functional. The additional ability to construct installation scripts to manage the database side of an application goes a long way toward your being able to deploy self-standing applications in one process. But remember that some files may need to be migrated manually because they don't fall into the realm of what APEX can handle via supporting objects.

CHAPTER 11

■ ■ ■

Understanding Websheets

Websheets were a new marquee feature of APEX 4.0 and deliver end-user control over both web content and structure. In the early days of APEX, when it was still known as Project Marvel and later as HTML DB, some people thought that end users could use APEX to develop their own applications. Although this was true for simple spreadsheet-like applications, most end users weren't comfortable building web applications that needed an underlying normalized database together with snippets of SQL, PL/SQL, and JavaScript. Websheets now fulfill the early promise of end-user development for web content like blogs, wikis, and very simple business applications. Websheets give end users this power without forcing them to learn how to normalize a database or write code. Everything in websheets, except a few optional, advanced features, is declarative.

Websheets have been designed so that they're easy to use. However, like all computer tools, there is an associated learning curve. That's the bad news. The good news is that the learning curve is very shallow. The tool relies heavily on wizards that lead you intuitively through the content-creation processes.

This chapter will outline the underlying structure of websheets, describe the navigation style, and highlight some of the handy features that will make you productive. This chapter will concentrate on *what* websheets can do, while Chapter 12 will focus on *how* websheets are built by leading you through some step-by-step scenarios. After reading this chapter and working through the next chapter, you will be able to quickly create professional-looking web content.

■ **Note** As you read this chapter, you may find yourself wondering how to create some of what is discussed. Not to worry–the examples in the next chapter will provide an in-depth look at the major tasks involved in creating a websheet. This chapter will provide the background that will enable you to follow along with and fully understand the upcoming examples.

Websheet Structure

The fundamental building blocks of a websheet (see Figure 11-1) are simple to envision. A websheet is a container for web pages. The web pages, in turn, are containers for sections. A section, which is similar to a region in an APEX database application, contains your content. Annotations are used to enhance both the content and the search functionality.
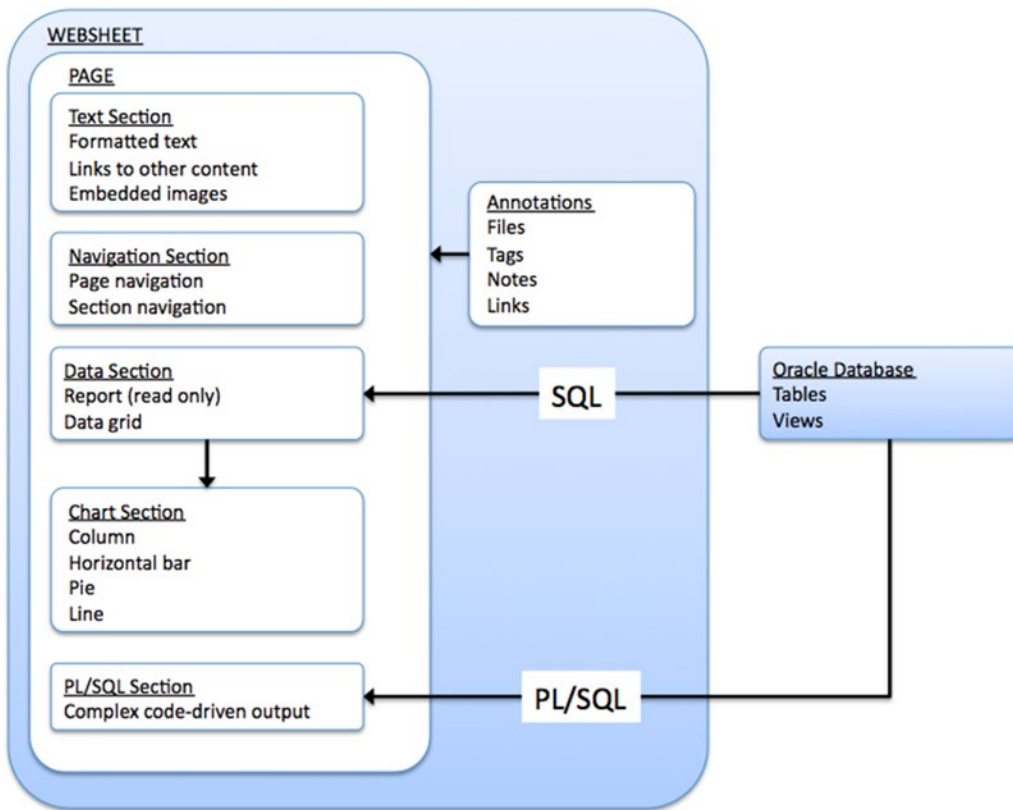
Figure 11-1. *Websheet structure*

There are five section types:

- *Text*: Text sections contain text that is easily formatted. Links to other content and images are embedded within the text by using very simple markup syntax.

- *Navigation*: Navigation sections help you navigate through your hierarchy of pages. Creating these sections requires very little thought or effort on your part. You can also set up navigation within a long page by using section navigation.

- *Data*: Data sections are used to display data in a row-and-column format that is similar to a spreadsheet. There are two types of data sections: report and data grid. A report is used to display read-only data from outside your websheet (i.e., from tables in a schema assigned to the workspace). Data grids are spreadsheet-like objects that you build. You're responsible for defining the columns, adding data-entry business rules, providing default values, and so on. If you've used spreadsheets, you'll find this work relatively easy to do.

- *Chart*: Chart sections are used to display graphs. Chart sections get their data from data sections. You link a chart section to a data section by using a simple and intuitive wizard.

- *PL/SQL*: Users with PL/SQL knowledge can create PL/SQL sections and write their own code against the associated schema. PL/SQL sections are available only if the websheet application developer has enabled the Allow SQL and PL/SQL attribute on the Websheet Properties page.

Navigation

We speak of websheet navigation in two contexts. First, we discuss navigating through a websheet's content. Second, we discuss navigating through the pages that are used to build a websheet. Be mindful that, in practice, you frequently flip back and forth between these two contexts.

In both contexts, there are usually several ways to navigate to a given page or section. The duplicate navigation choices might cause you a bit of confusion at first. However, after you work with websheets for a while, the navigation choices become helpful and intuitive. This chapter doesn't document every possible navigation path; instead, it shows you where to look for navigation links so that you can quickly find a comfortable navigation style that works for you.

Content Navigation

Content navigation enables you to quickly go to pages and sections within a page. Page navigation is mandatory and is created for you as you create the websheet hierarchy. Section navigation is optional and is useful on long pages that require a lot of vertical scrolling.

Page navigation is created by the websheet itself automatically by adding hierarchical breadcrumbs. In Figure 11-2 (a screenshot of the websheet you will build–a soccer team management application), the hierarchical breadcrumbs are found in the drop-down menu at left, which contains links to Players, Results, and Schedule. For small websheets, the breadcrumbs and the right-side navigation sections might be all you need.
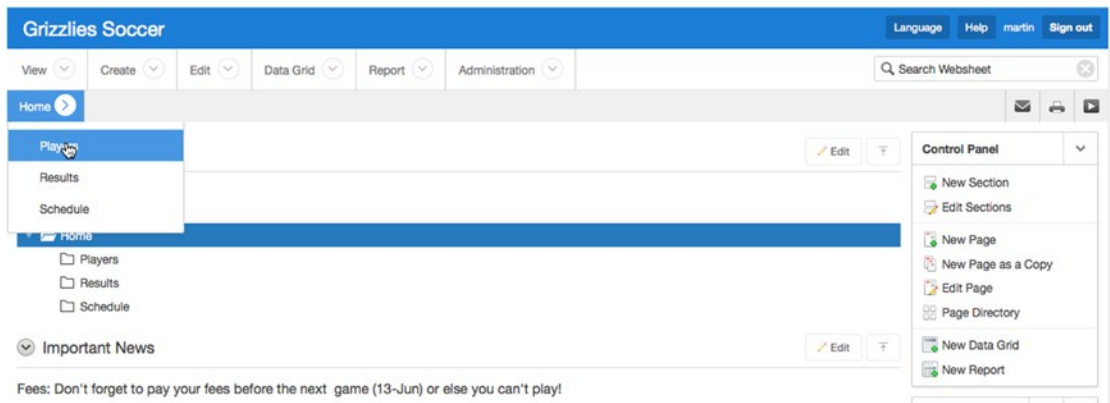
Figure 11-2. *Page navigation created by the websheet*

Second, you can add page navigation manually by creating a page-navigation section. You can also embed explicit page links in the page content (see Figure 11-3). The details for adding a page-navigation section are discussed later under the heading "Navigation Sections." Embedded links are discussed in detail in the "Markup Syntax" section.
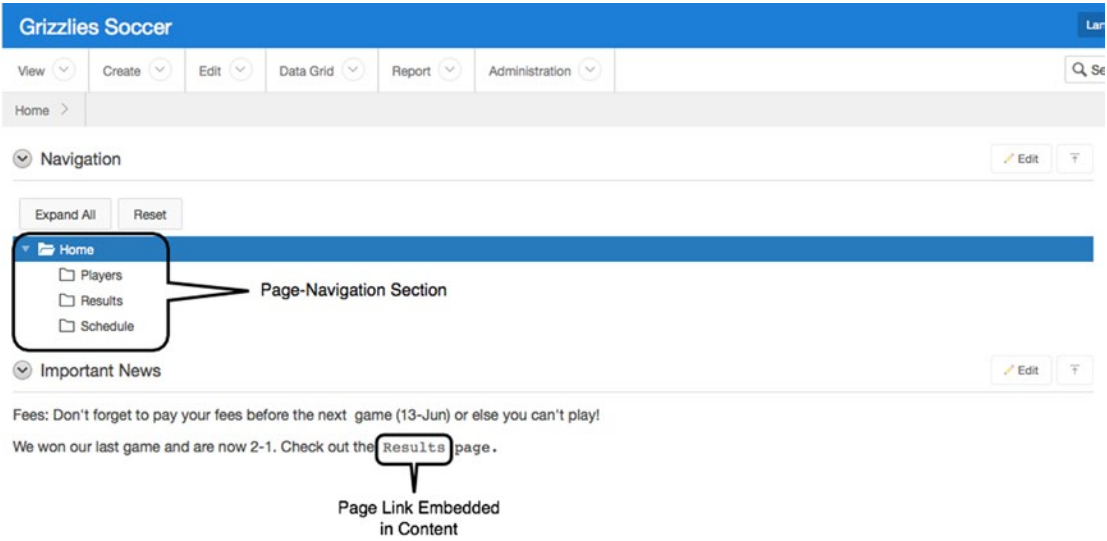
Figure 11-3. *Page navigation created by the user*

Section navigation is optional. It's useful for content-heavy pages that require a great deal of scrolling to reach the bottom of the page. Section navigation is almost identical to page navigation (see Figure 11-4). The main difference between page and section navigation sections is the lack of hierarchy in section navigation; sections have no children.
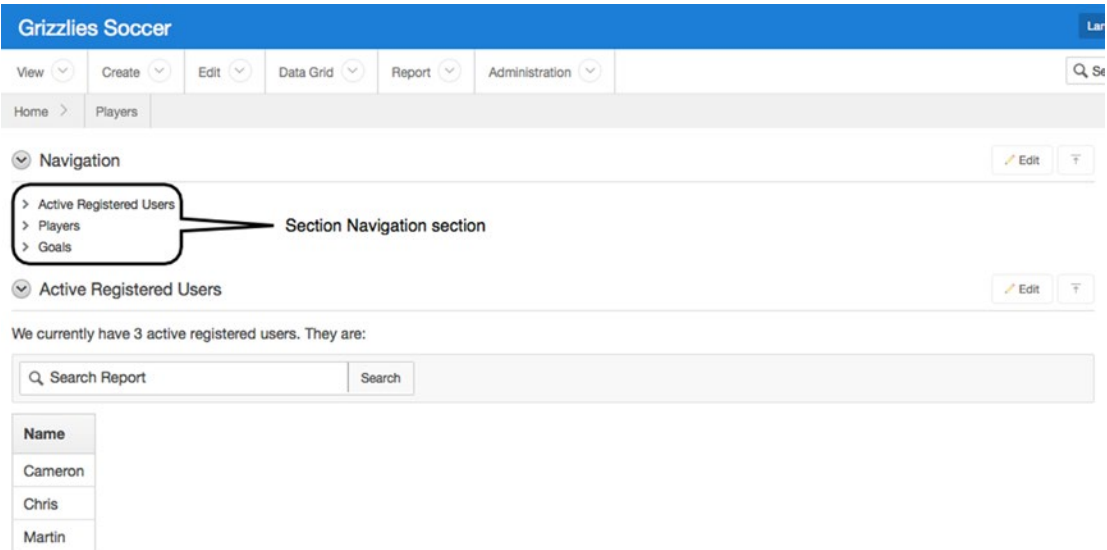
Figure 11-4. *Section navigation created by the user*

Structural Navigation

Structural navigation is utilized to access the pages that are used to build and update the structure and content of websheets. On most websheet pages, two areas enable you to access the structural pages (see Figure 11-5). The first area contains the drop-down menus at the top of the page. These menus don't change from page to page. The second area is located on the right side of every page. This area contains a set of sections that, in turn, contain links to the various structural pages. These sections and links vary from page to page and are tailored to the page's context.
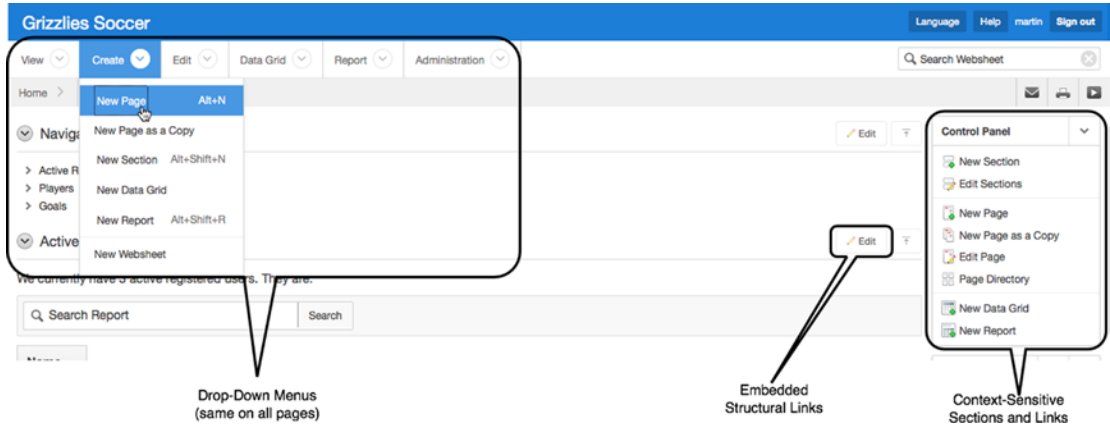
Figure 11-5. *Structural navigation*

In addition to these areas, some structural links are embedded in the content sections. These embedded links are convenient for getting to the Edit page for the content that is currently being displayed.

Help

Don't overlook the Help link (see Figure 11-6). The help is clear, concise, and useful. Clicking the Help link invokes a context-appropriate pop-up page that contains mostly static information that you can read at your leisure. We strongly recommend that you do so; it takes only a few minutes.

Figure 11-6. *Help link*

In addition to the static information, one tab contains dynamic information. The Application Content tab contains complete lists of all the websheet's pages, sections, files, images, data grids, and reports (see Figure 11-7). These lists are presented in Interactive Reports that you can tailor to suit your needs.
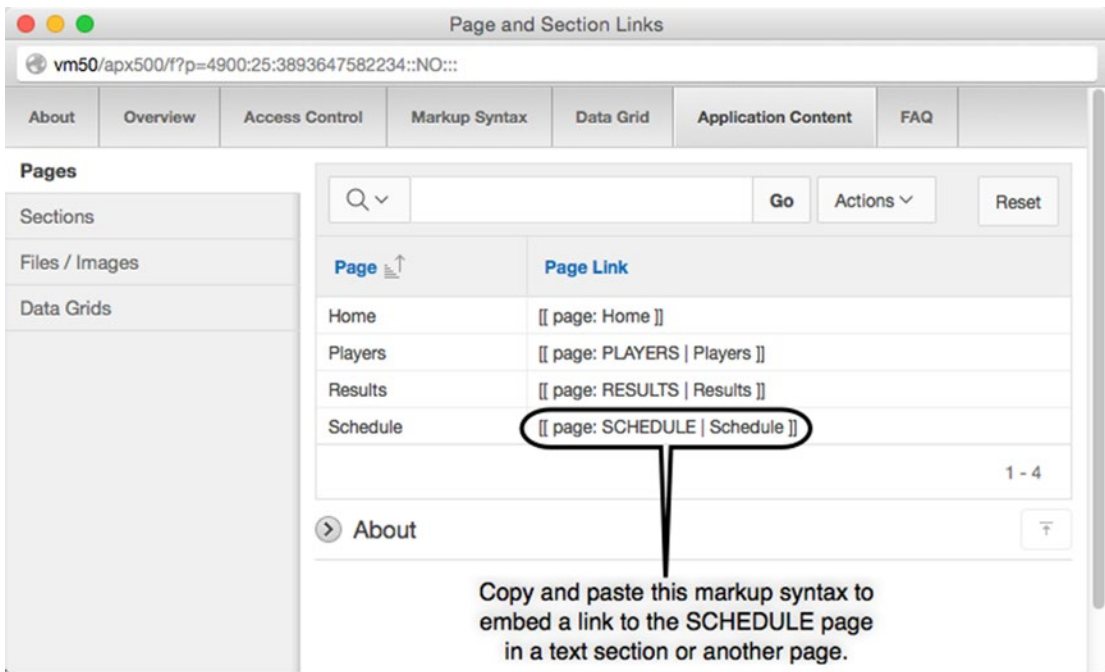
Figure 11-7. *Application Content tab*

All of the Interactive Reports contain a column that displays the explicit markup syntax that enables you to embed links to the listed objects directly in your content. This saves you the effort of having to remember the details of the markup syntax and type it manually; you also avoid the aggravation of debugging typos.

An example of how the markup syntax is used is shown in Figure 11-8. A link to the Results page is embedded in the content found in the Important News text section. Clicking the Edit link in the Important News text section takes you to the corresponding Edit page (see Figure 11-9), where the underlying markup syntax for this example is illustrated.

Figure 11-8. *Resulting content of markup syntax*
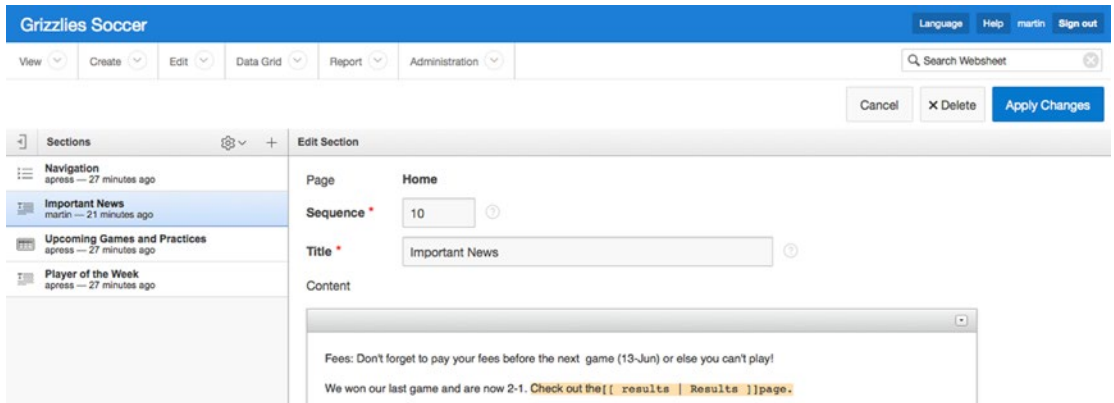
Figure 11-9. *Embedded markup syntax*

Markup Syntax

The markup syntax that is used to embed links in your web content looks a bit like a computer language. End users might find the syntax a bit intimidating; however, the syntax structure is simple, forgiving, and well documented on the Help page:

```
[[ LINK_TYPE: LINK_TARGET | LINK_NAME ]]
```

The opening and closing delimiters are two square brackets that are easy to read. LINK_TYPE is a keyword with a trailing colon. The available link types are described in Table 11-1.

Table 11-1. *LINK_TYPEs and Descriptions*

page:	Links to a page in the websheet
section:	Links to a section in the websheet
url: and popupurl:	Links to a URL
file:	Downloads the target file to the user's computer
image:	Displays an image on the page in a text section
data grid: or datagrid	Links to a data grid's Edit page
report:	Links to a read-only report page
sql:	Displays the result of an SQL statement in a grid
sqlvalue:	Displays a single value from an SQL statement

LINK_TARGET specifies the object that is displayed when you click the link. For a page link, LINK_TARGET is the page alias. For a file, LINK_TARGET is the file name or alias. The only exception to this pattern is the sql: LINK_TYPE. The sql: LINK_TYPE's LINK_TARGET isn't a link; it's an SQL statement that returns data in rows and columns. The SQL data is automatically displayed when the page is displayed. The sql: syntax is also referred to as SQL tags, and this feature must be turned on by an administrator in the application properties area. This is covered later in the "Reports: Setup" section.

A vertical bar character separates LINK_TARGET and LINK_NAME. The only fussy part of the syntax is the single spaces that must precede and follow the vertical bar.

LINK_NAME contains the text that is embedded in the page's content. The user clicks this text to follow the link. There are two exceptions to this pattern. First, the image LINK_NAME is optional and can be replaced by HTML markup. For example, you can use HTML markup to resize the image. Second, the sql: LINK_TYPE has no LINK_NAME. LINK_NAME isn't required, because the SQL data itself is automatically embedded in the page content.

The markup syntax is forgiving. It's case insensitive, and the websheet code makes several friendly assumptions. For example, if you omit LINK_TYPE, the websheet scans its metadata for LINK_TARGET. If an exact match is found, the websheet assumes that this is the target for which you were looking. In other words, you can be a bit sloppy with the syntax and still get the correct result.

User Authentication

User authentication governs how users log on to a websheet. There are four options:

- *Application Express Account*: Websheet users log on to the websheet by using the IDs and passwords that have been set up in the APEX workspace that hosts the websheet.

- *Single Sign-On*: Oracle's single sign-on (SSO) technology enables users to sign in to their computing environment one time and then access all their applications, such as websheets, without having to re-enter their username and password. This is an advanced feature that is out of scope for a beginning book.

- *LDAP*: Lightweight Directory Access Protocol (LDAP) is used by websheets to provide SSO capability in computer environments that use non-Oracle authentication schemes. This is an advanced feature that is out of scope for a beginning book.

- *Custom*: This advanced feature will be explicitly explained and illustrated in Chapter 12.

The authentication method is normally chosen when the workspace administrator initially creates the skeleton websheet. If the websheet has been set up using Application Express Account authentication *and* you're logged in to the workspace as an Application Express developer, you can edit the authentication type from the Websheet Properties, as shown in Figure 11-10.
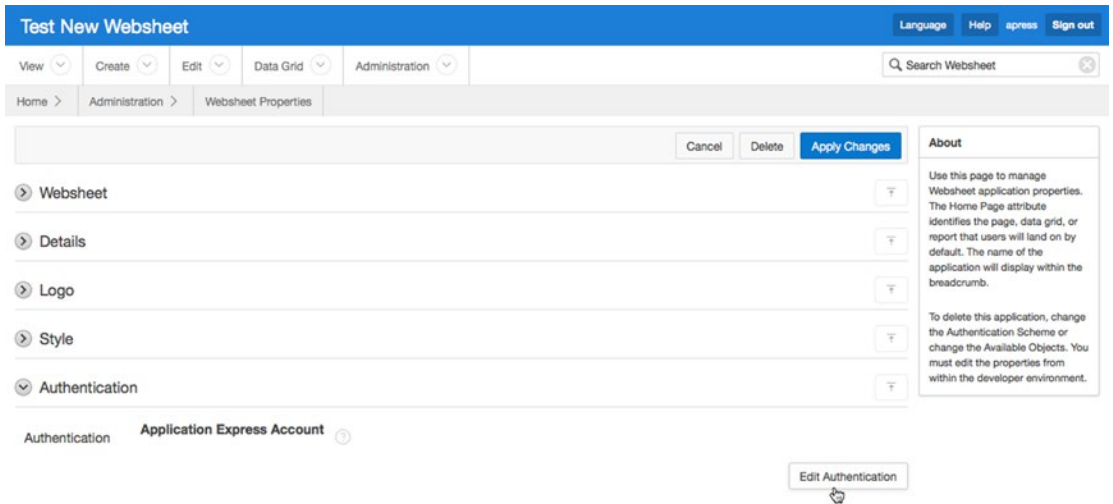
Figure 11-10. *Authentication set to Application Express Account for a websheet*

When you click the Edit Authentication button, you're redirected to the websheet's Application Properties page in the Application Builder (see Figure 11-11).
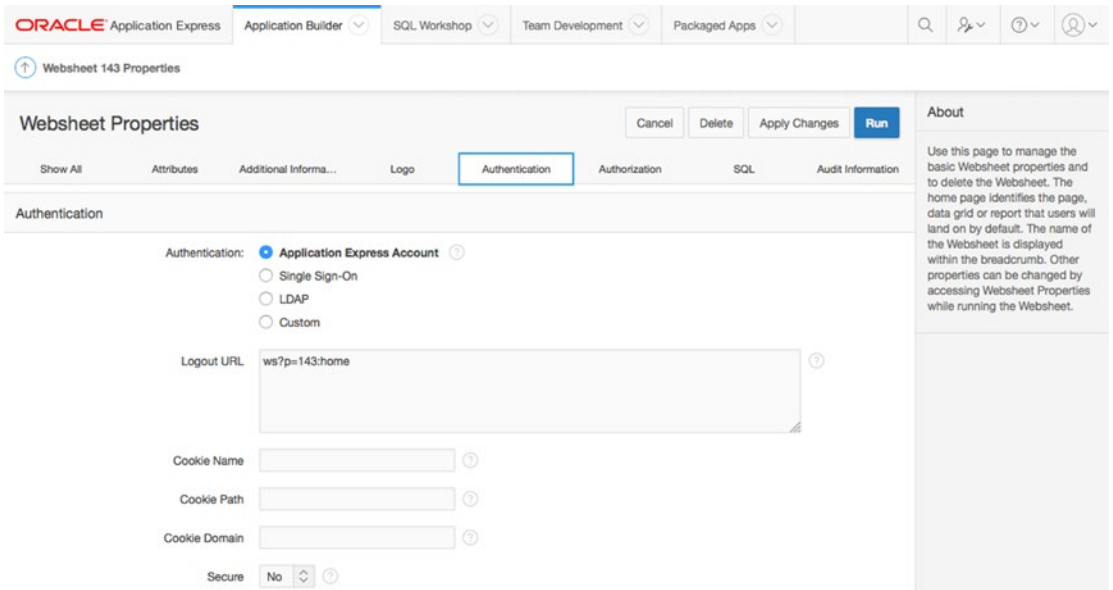
Figure 11-11. *User-authentication options displayed in the Application Builder*

When you're using the Application Express Account authentication option and also are logged in to the Application Builder, clicking the Edit Authentication button takes you out of the websheet and directly into the Application Builder. This transition may not be obvious to you at first, because the page bodies are similar. You need to verify your context by looking at the top of the page and the menus. See Figure 11-11.

If the websheet is using SSO, LDAP, or custom authentication, you must log in to the APEX Builder as either an APEX administrator or a developer. After you log in, navigate to the websheet's Application Properties page, where you can pick the desired authentication scheme and configure it. The details of the example shown in Figure 11-11 will be discussed in Chapter 12.

User Authorization

Websheets have three authorization roles:

- *Reader*: This is the read-only role. Figure 11-12 is a websheet home page as seen by a reader. The page contains content together with navigation objects. When you drill down into data pages, you can see the data but not the buttons that are used to add, change, and delete data.

Figure 11-12. *Websheet home page as seen by a reader*

- *Contributor*: This role is allowed to add, change, and delete a websheet's content plus manipulate the structure. Figure 11-13 is the websheet home page as seen by a contributor. Notice the rich set of functionality that is added for this role. The top drop-down menus contain links to the structural pages. The text sections contain Edit links. The right-side sections contain links to the structural pages. When you drill down into the data pages, you can see the buttons that allow you to add, change, and delete the content.
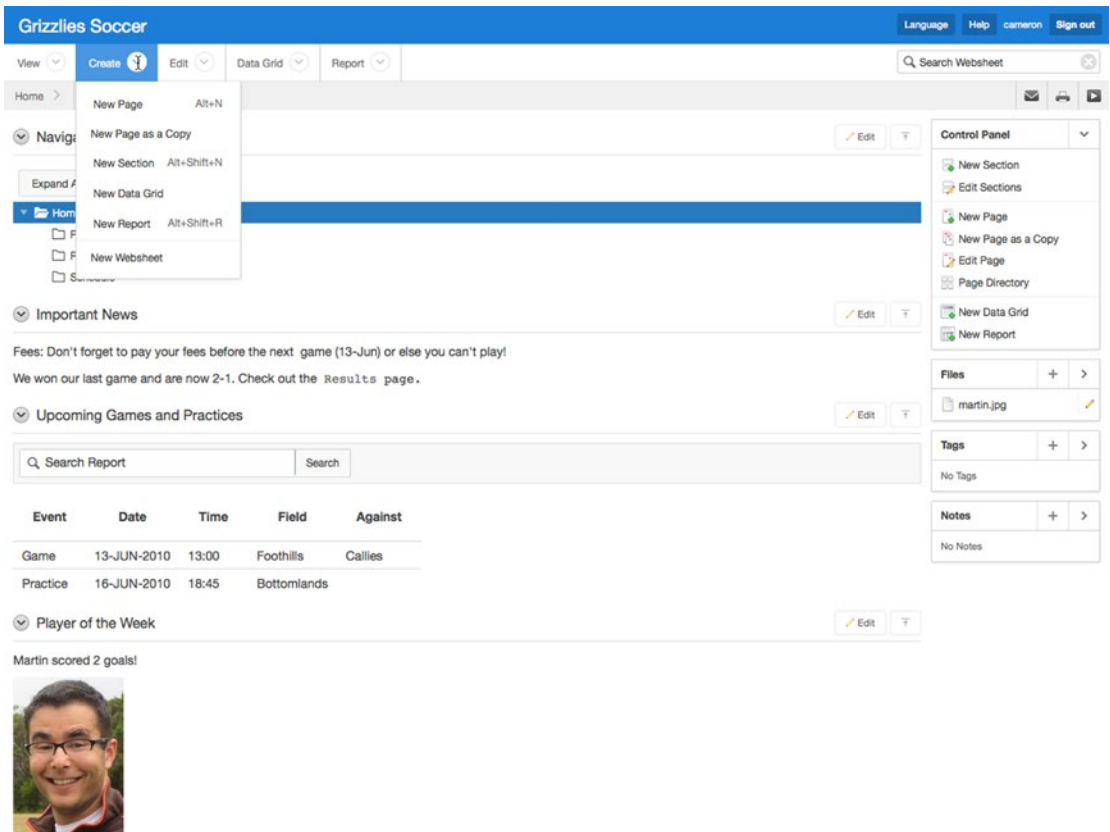
Figure 11-13. *Websheet home page as seen by a contributor*

- *Administrator*: This role can create and delete websheets. It's responsible for maintaining a websheet's global properties and maintains the list of users who can access the websheet. Figure 11-14 is the administrator's view of the websheet home page. The only addition to the contributor's view is the Administration drop-down menu at the top of the page.
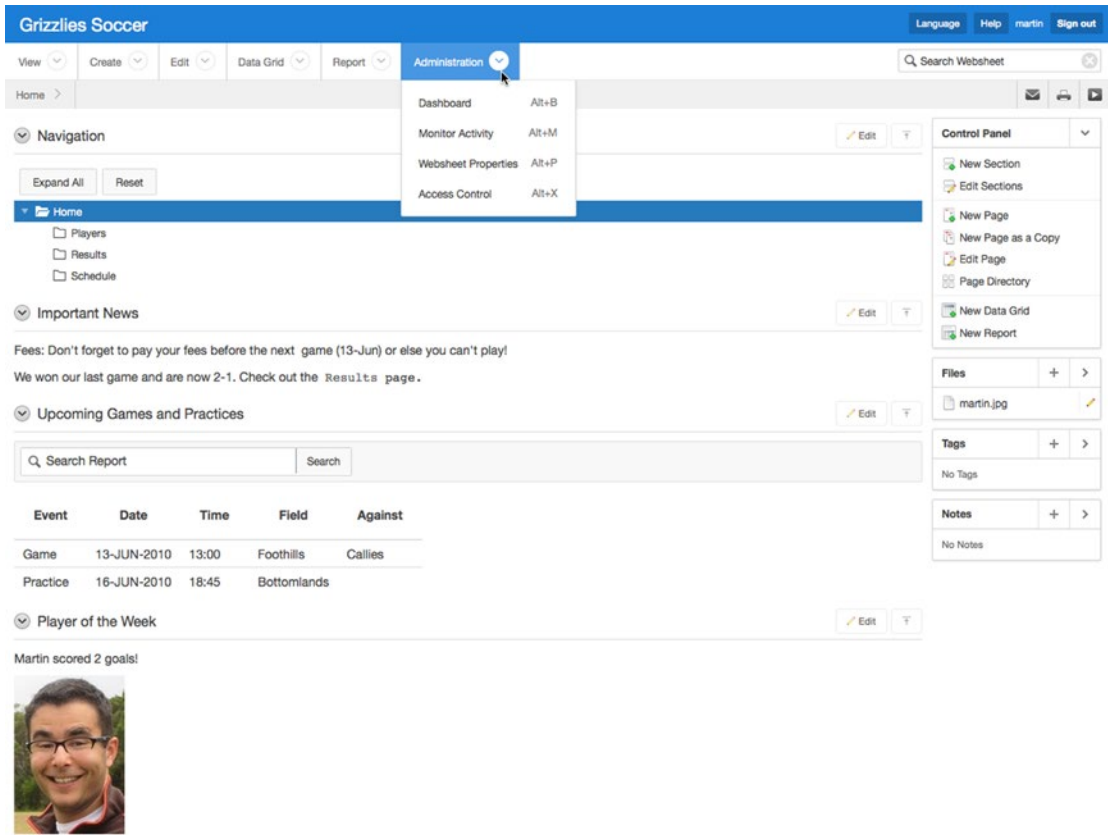
Figure 11-14. Websheet home page as seen by an administrator

The websheet administrator configures user privileges through the Access Control list, which is found under the Administration drop-down menu (see Figure 11-15). This task is usually done after you set up the authentication scheme.
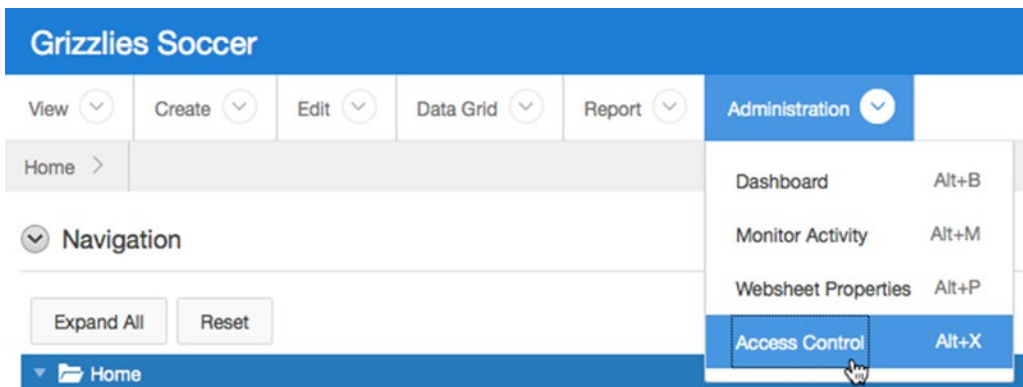
Figure 11-15. Navigating to the Access Control list

The Access Control list is simple to create and maintain (see Figure 11-16). Create a new entry by clicking the Create Entry button. Change an existing entry by clicking the pencil icon in the list. In both cases, you're taken to the Entry Details page, which has only two fields: the username and the privilege level (see Figure 11-17).
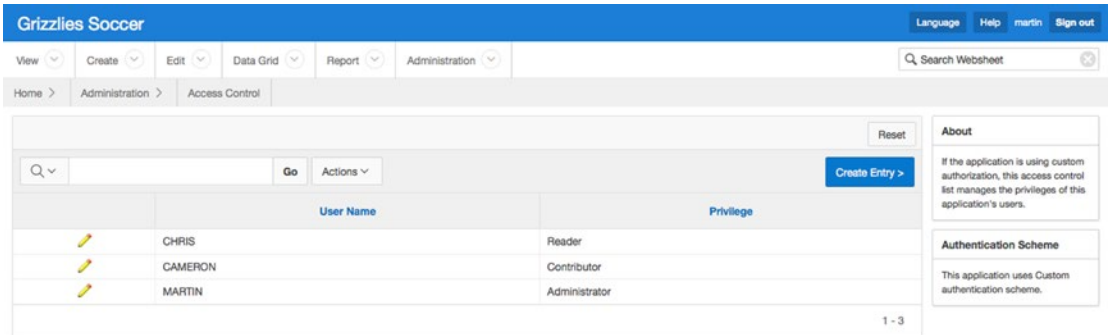
Figure 11-16. *Access Control list*
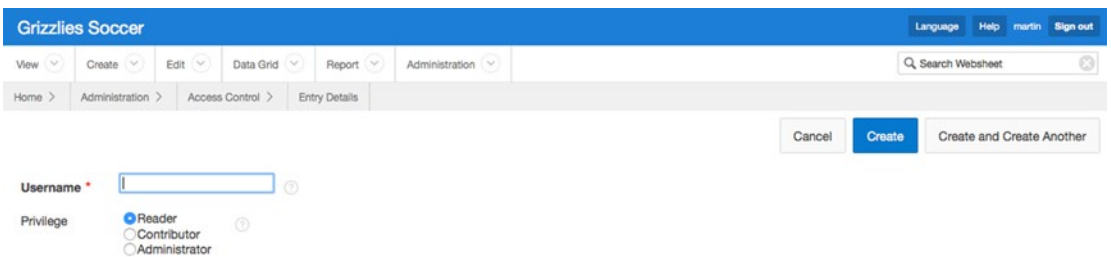
Figure 11-17. *Entry Details page*

The Access Control list is somewhat sensitive to the authentication scheme that is used. When you use SSO, LDAP, or a custom authentication scheme, the Access Control list is mandatory. In this context, it's easy to understand and build. You build the Access Control list as a duplicate of the list of users in the authentication scheme. The hook between the two lists is the username. The websheet username must match the user ID in the authentication scheme.

When you use the Application Express Account authentication scheme, things get a little harder to understand. In this instance, using the Access Control list is optional. Because the websheet is inside an Application Express workspace, the websheet can directly use the existing APEX user accounts. The websheet privileges are inferred from the APEX user-account privileges. Table 11-2 illustrates the translation between the APEX workspace privileges and the websheet privileges. You can override the default translation by adding the APEX users to the Access Control list. For example, you might want an APEX workspace administrator to have the reader privilege on a given websheet. To do this, you add the APEX workspace administrator's ID to the Access Control list and set the privilege to reader.

Table 11-2. *Access Control Configuration*

Authentication Scheme	No Access Control List	With an Access Control List
Application Express Account	APEX administrator = websheet admin APEX developer = websheet contributor APEX end user = websheet reader	The Access Control list overrides the inferred APEX websheet privileges.
Single Sign-On	NA - Access Control list is mandatory.	Access Control ID must match SSO ID.
LDAP	NA - Access Control list is mandatory.	Access Control ID must match LDAP ID.
Custom	NA - Access Control list is mandatory.	Access Control ID must match custom ID.

Websheets can be set up for public access. This means that anyone who invokes the websheet's URL is allowed to access the websheet with the reader role. Logging in with an ID and password isn't required. You can set this up by going to the websheet's Properties page in the APEX Application Builder and changing the Allow Public Access control under the Authorization section. See Figure 11-18.
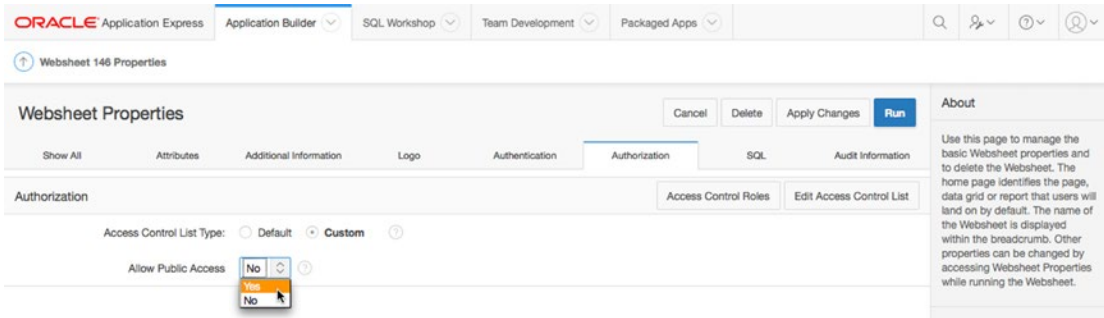
Figure 11-18. *Navigating to the Application Properties page*

Select Yes in the Allow Public Access drop-down menu and click the Apply Changes button. Now, when you run the websheet application, you're automatically logged in as the user "nobody" with reader privileges. Administrators and contributors who need to update the websheet's content can log in by using the Sign In link that appears at the upper right on all the websheet's pages (see Figure 11-19).
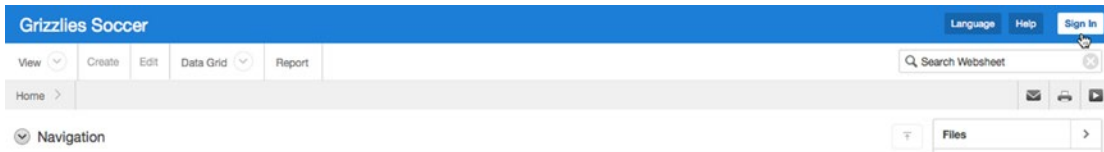
Figure 11-19. *Public websheet with Sign In link*

Sections

Sections contain your content. The following chapter sections will illustrate useful features found in the websheet structural environment by showing you the Edit pages for existing objects. Step-by-step procedures for creating new websheet objects will be covered in Chapter 12.

Text Sections

Text sections contain text, embedded links, and images. Text sections can be used to create wikis and blogs. To start a wiki, the original author creates a text section and then invites contributors to edit the text section's content. To start a blog, the original author creates a text section and then invites contributors to add more text sections in reply to the first section.

To access a text section's Edit page, you click the Edit link in the upper-right corner of the text section (see Figure 11-20).

Figure 11-20. *Navigating to a text section's Edit page*

The Edit page for a text section is simple and clean when it's invoked. By default, the collapsible regions are collapsed (see Figure 11-21). The upper-right collapsible region link, a small arrow icon, contains the text-formatting controls. Expanding this section by clicking the icon displays an edit palette that is similar to what you might expect from your favorite word processor.
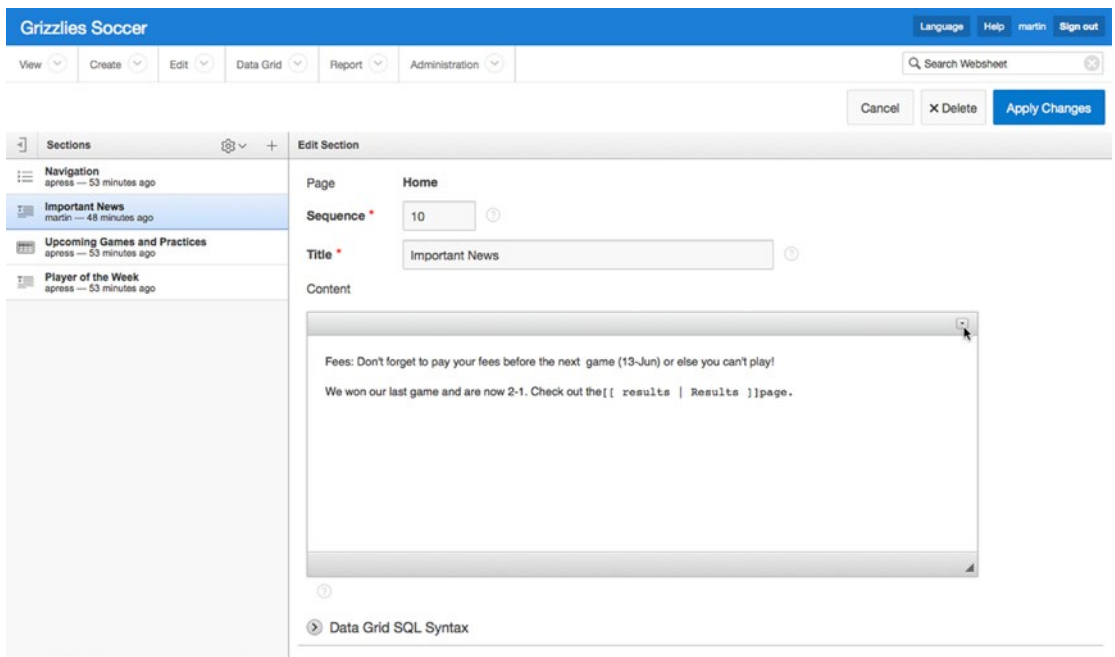
Figure 11-21. *Edit page for a text section, collapsed*

The lower-left link, Data Grid SQL Syntax, contains links to information on how to access data from the data grids in your websheet. Clicking the links presents a pop-up page with cut-and-paste syntax for many data queries and links.

Finally, the section on the left contains a list of all the sections on the page. Clicking any one of the section names allows you to edit the properties and/or content for the section. This region also has a toolbar across the top that lets you perform various tasks. See Figure 11-22.
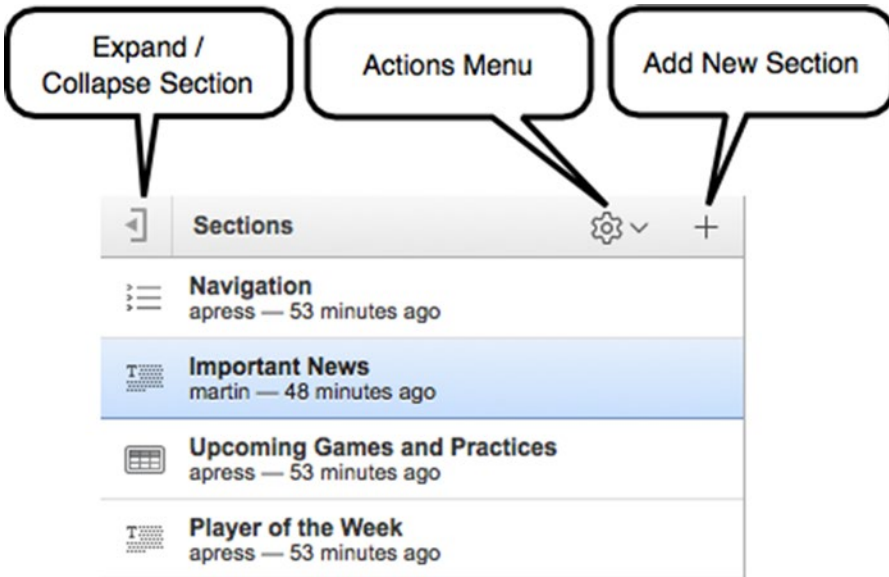
Figure 11-22. *The Sections toolbar on the Edit Section page*

When you expand the collapsible regions, you can see the considerable scope available for enhancing your content (see Figure 11-23). The upper-right icon expands into a section that contains a number of formatting icons. We don't describe each formatting icon's function in detail; they're intuitive because they're similar to other tools, like a word processor, that you probably use regularly.
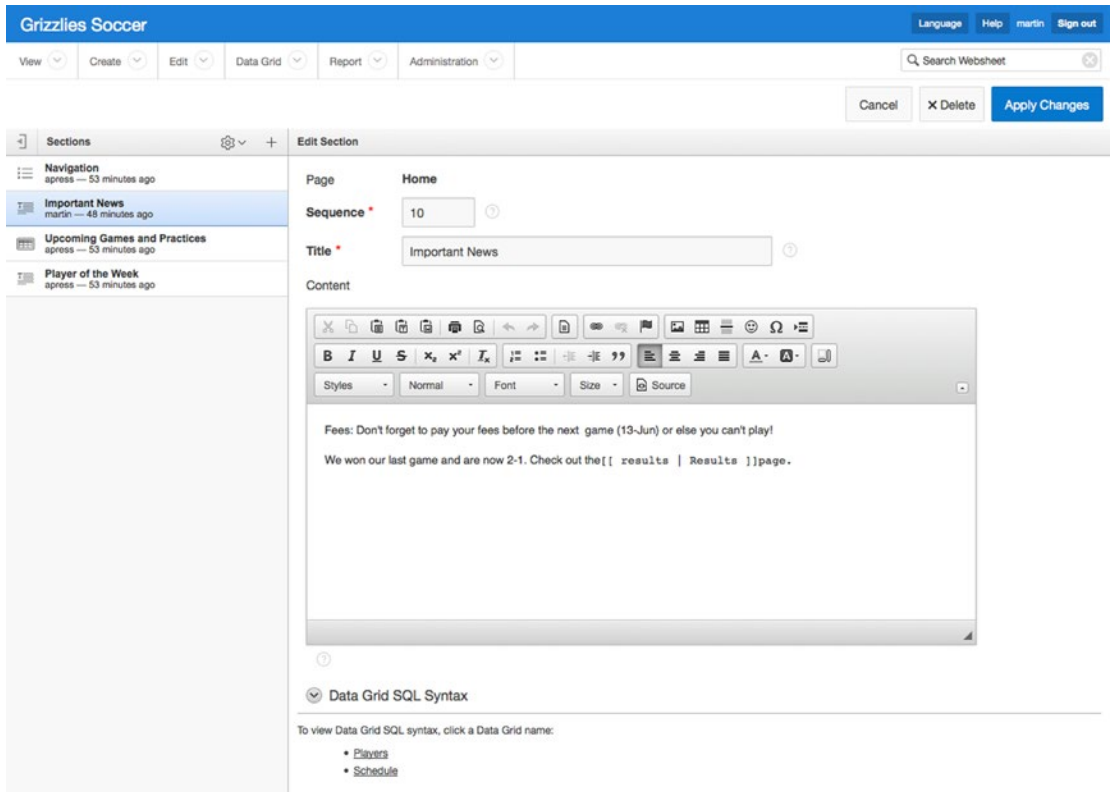
Figure 11-23. *Edit page for a text section, expanded*

Other elements available include the following:

- The list of page sections at the bottom of the page shows you where the current section fits within the page relative to the other sections.

- The Help section contains direct links to the Help page tabs.

- The Tasks section contains links to processes that automate moving the section to an existing or new page.

The Show History link will be explained in the "Administration" section later in this chapter.

■ **Note** Many pages contain collapsible regions at the bottom of the page. Some of the regions contain help text; others contain lists of things that help give you a sense of place and context, which in turn helps you to get your content right. In all cases, the collapsible regions found at the bottom of pages are useful.

Navigation Sections

The most important aspect of adding navigation sections to your web content is the fact that adding them takes very little effort or thought on your part. Navigation sections are easy to use because they're declarative, and the websheet takes care of virtually all the details like the page links and formatting. The details of adding navigation sections will be discussed in Chapter 12.

A page-navigation section is shown in Figure 11-24. Clicking the Edit link takes you to the Edit page (see Figure 11-25). The Edit page has five inputs:

- *Sequence*: Positions the section among the other sections on the page

- *Title*: The title of the section

- *Starting Page*: Lets you start the navigation tree on pages that are below the home page in the page hierarchy. In this example, a user could add a page-navigation section to the top of their personal page that shows only the pages below their personal page in the hierarchy.

- *Maximum Levels*: Limits the number of levels displayed in a page-navigation section. In this example, if you set the Maximum Levels value to 3, you would see only the pages shown in Figure 11-24 even if a user adds child pages under their personal page.

- *Order Siblings By*: Allows you to choose the sort order of all pages at the same level. Options are Page Name, Created Date, and Updated Date.

Figure 11-24. *Page-navigation section*
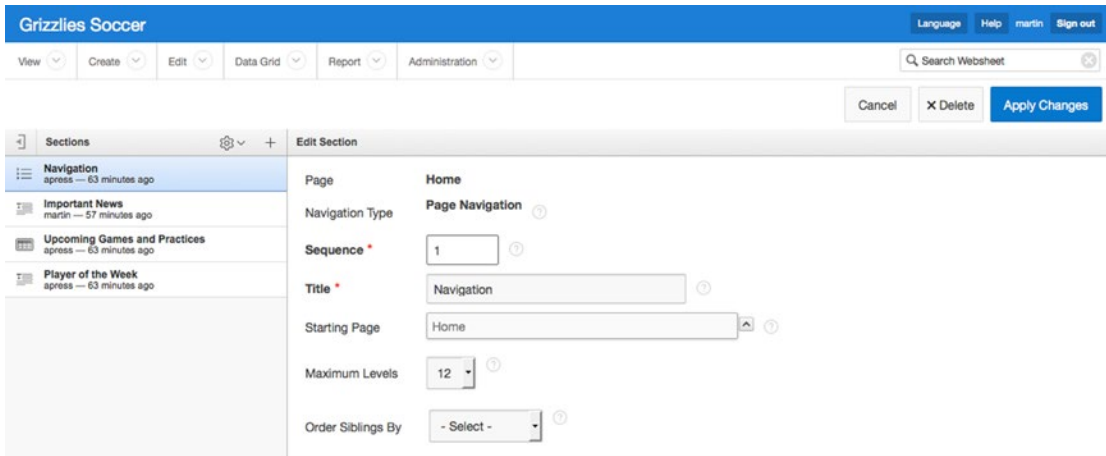
Figure 11-25. *Edit page for a page-navigation section*

A section-navigation section is shown in Figure 11-26, and you can see the Edit page in Figure 11-27. The only inputs are the sequence and title. In this case, the websheet takes care of all the other details. You don't have to do any work.

Figure 11-26. *Section-navigation section*
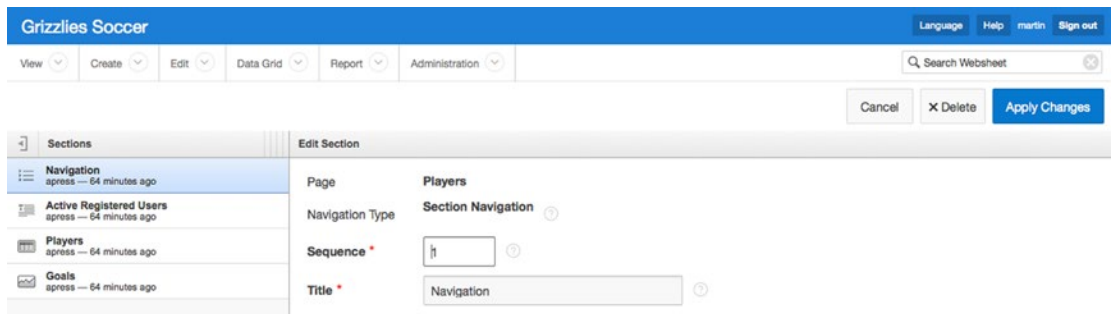
Figure 11-27. *Edit page for a section-navigation section*

Data Sections

There are two types of data sections: Data Grids and Reports. Data Grids are spreadsheet-like objects that you create entirely within a websheet. Reports display read-only data from external database tables or views that are located outside the websheet.

We will look at data grids first because they're native to websheets and are what you'll likely use the most.

Data Grids

A data grid is the most complex part of a websheet. However, if you've had a bit of experience with a spreadsheet, you probably won't have much difficulty learning how to use a data grid in a websheet.

This section will highlight some of the features of data grids. Chapter 12 will walk you through the steps that are required to create a data grid from scratch.

Data grids are used to organize data in a column-and-row format. Both the design and the data are held in the websheet environment; no external configuration is required.

Both data grids and reports can be put into data sections, and you can embed links to them in text sections. Figure 11-28 shows a data grid that is displayed in a data section. In this context, the data grid is read-only, like a report. The Edit link goes to a page that allows you to edit the data section, but not the data grid itself.

Figure 11-28. *Data grid displayed in a data section*

You can navigate to the context where you maintain the data grid's data by using the Data Grid drop-down menu (see Figure 11-29). You can either navigate to an individual data grid directly from the menu or click its link from the View All report.

Figure 11-29. *Navigating to a data grid's data-entry context*

Figure 11-30 shows the Schedule data grid. The search text box, the Go button, the Reports drop-down list, and the Actions drop-down menu are standard Interactive Report features that were discussed previously.
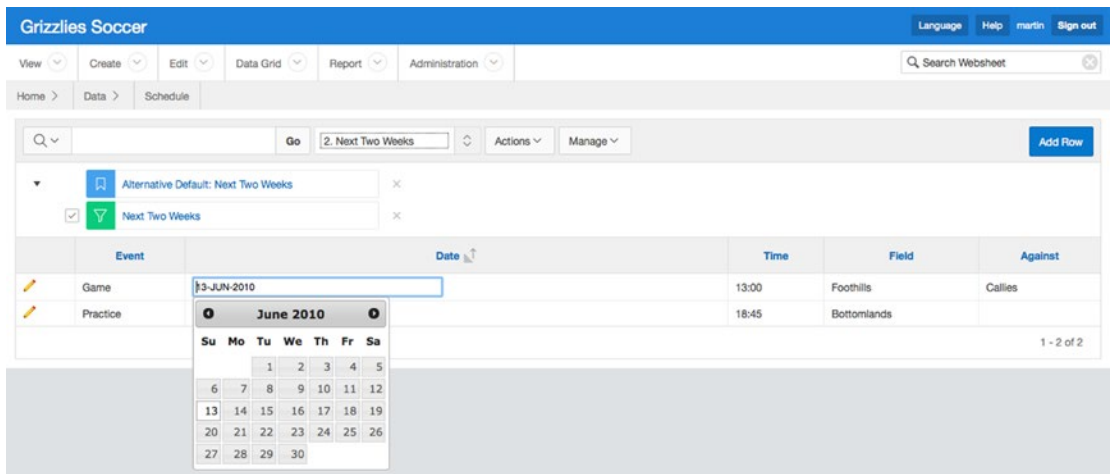
Figure 11-30. *Editing data in a data grid*

However, unlike traditional Interactive Reports, here you can change the data directly in the data grid. When you click a cell, the cell turns into an editable item, and you can type data directly into it. If you've configured a column as a date, a pop-up calendar automatically appears to help with accurate date entry. You can also configure a column to have a list of values; when this is defined, the cell contains a drop-down list. In addition, you can use the pencil icon to the left of the grid to link to a Form page that edits an individual row (see Figure 11-31). This is convenient when the data grid contains many columns and is too wide for your computer screen.
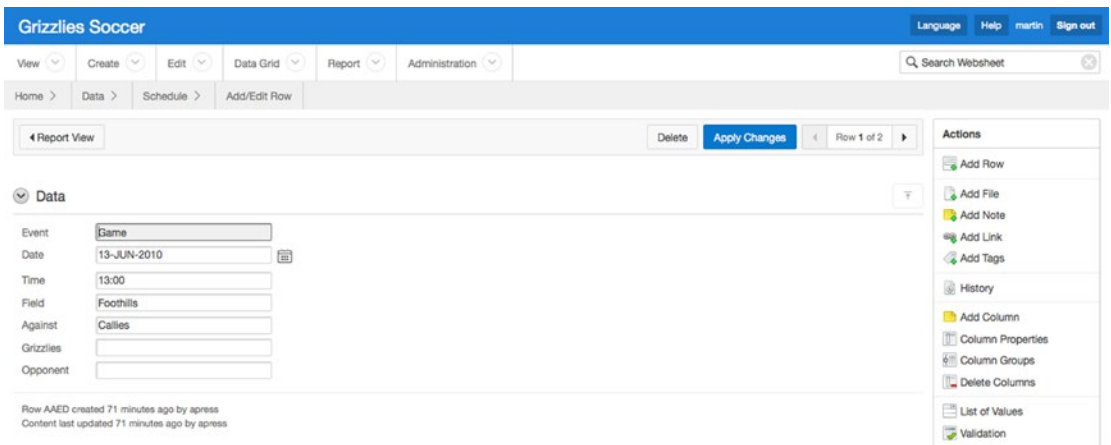
Figure 11-31. *Editing data on a Form page for one row*

You manage the structure of a data grid by selecting options from the Manage drop-down menu. Figure 11-32 and Figure 11-33 show all of the data-grid configuration options.
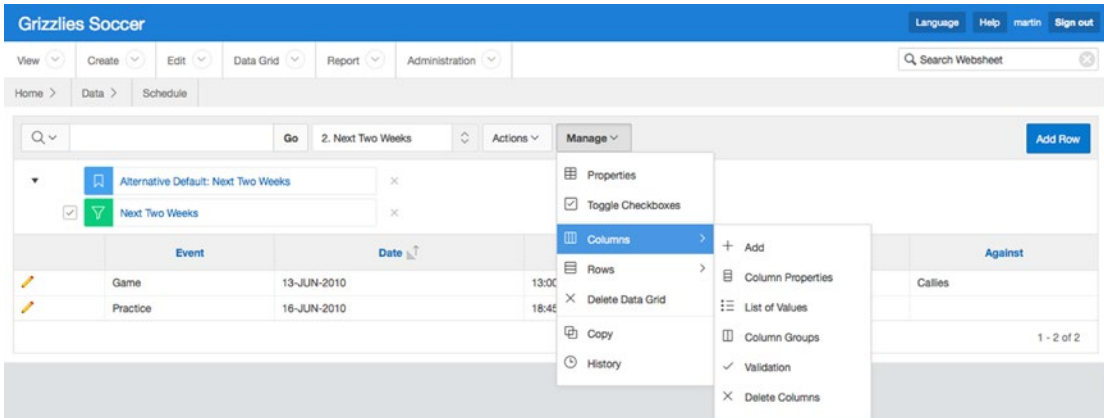
Figure 11-32. Data-grid management, column options expanded
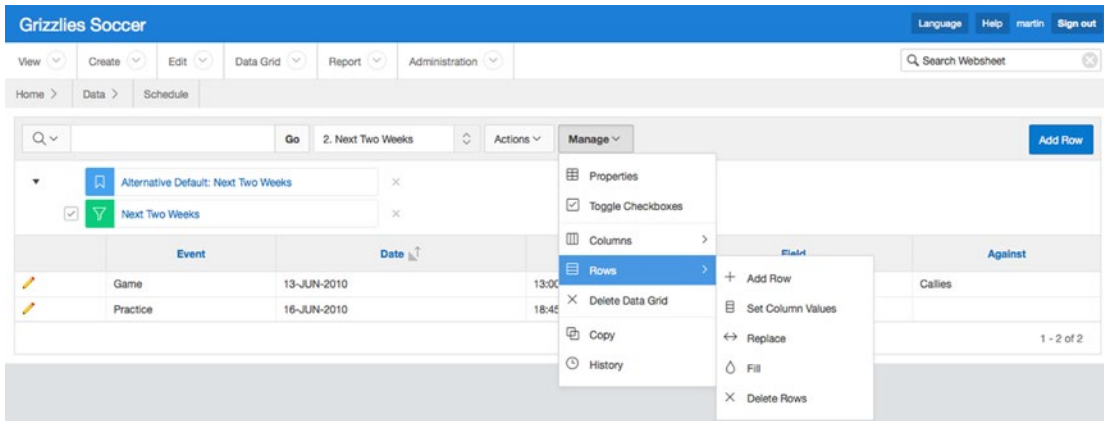
Figure 11-33. Data-grid management, row options expanded

An interactive edit section is displayed as a modal dialog when you select one of the Manage menu options. This is shown in Figure 11-34. The edit sections all contain their own Cancel and Apply buttons. You must click the Apply button to save your changes.
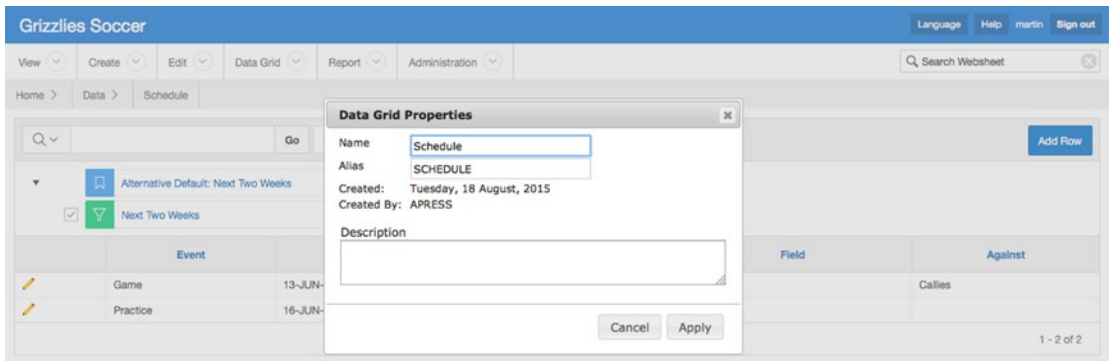
Figure 11-34. *Data-grid management, Data Grid Properties menu option*

The Manage drop-down menu gives you a rich set of options that allow you to use a data grid as a simple and friendly spreadsheet-like application. Most of the options are simple to use; they're illustrated in more depth in Chapter 12. The Manage menu options are summarized next:

- *Properties*: Lets you edit the overall data-grid properties.

- *Toggle Checkboxes*: Toggles the row-selection checkboxes on and off. The row-selection checkboxes are used to perform bulk updates on selected rows.

- *Columns*:

    - *Add*: Adds a new column to the data grid.

    - *Column Properties*: Changes the properties of a column after it has been created.

    - *List of Values*: Creates a named list of values. A named list of values can be used in more than one data grid.

    - *Column Groups*: Creates column groups.

    - *Validation*: Adds data-entry validations. Validations are chosen from a defined list of business rules. You can use several validations simultaneously to achieve a result. For example, to make sure a number is greater than zero, you can use the Column Specified Is NOT Zero validation together with the Column Specified Doesn't Contain Any of the Characters in Expression validation, and you would enter a minus-sign character in the Validation Expression text area. This last point illustrates the fact that all of the underlying data in a data grid is text, and that sometimes you need to use more than one validation rule to achieve a single result.

    - *Delete Columns*: Deletes one or more columns.

- *Rows*:

    - *Add Row*: Displays the data-entry page, where you can enter new data.

    - *Set Column Values*: Enters data into many rows in a single column. A value can be set for All Rows, Selected Rows, or Empty Rows.

    - *Replace*: A find-and-replace feature that is similar to that found in text editors and word processors. It can be applied to All Rows or Selected Rows.

331

- *Fill*: Fills a column's null cells with the value found above them.

- *Delete Rows*: Deletes rows from the data grid. This is done for All Rows, Selected Rows, or Rows with Empty Columns.

- *Delete Data Grid*: Deletes the data grid from the websheet.

- *Copy*: Makes a copy of the data grid.

- *History*: An audit trail for the data.

Reports: Setup

The Report feature and the related SQL Tags feature require a small amount of setup by the administrator before contributors can use them. Start the setup process by navigating to the APEX Application Builder and editing the websheet's properties. In the SQL and PL/SQL section (see Figure 11-35), set Allow SQL and PL/SQL to Yes. After you do this, click the Add Object button to display the page shown in Figure 11-36. This page enables you to create a list of suggested objects. The list of suggested objects is an optional convenience that automatically creates a drop-down list of database objects together with helpful comments.
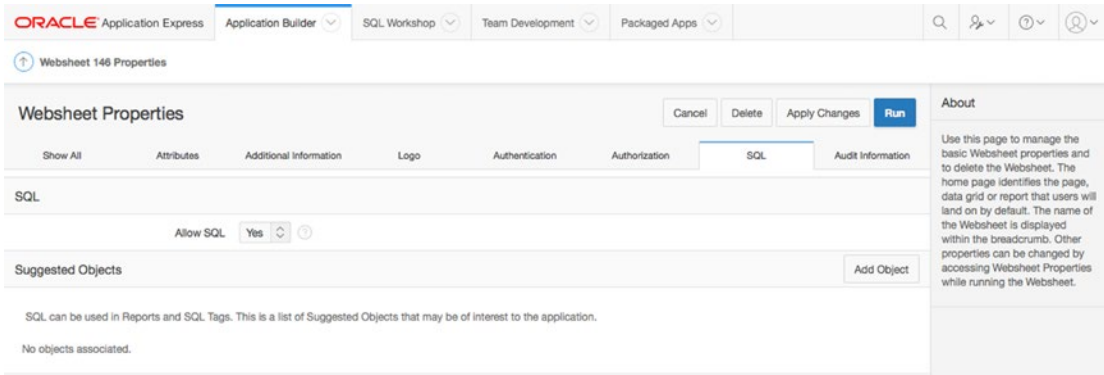
Figure 11-35. *Websheet report and SQL tag setup*
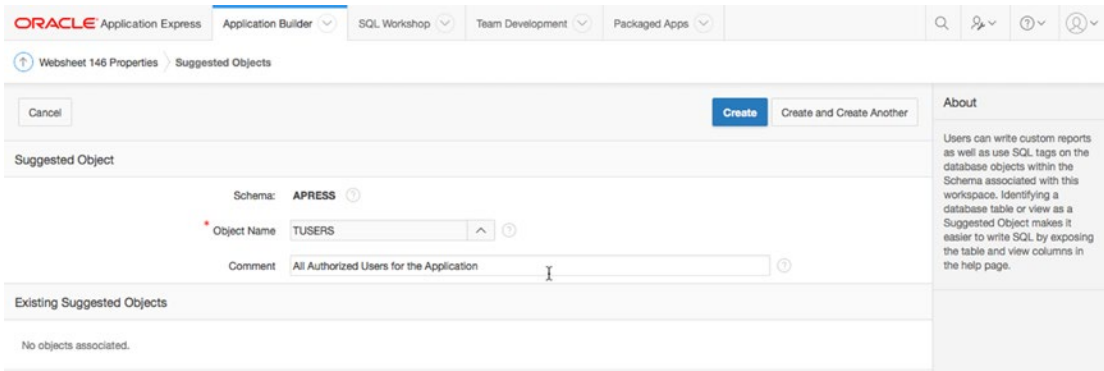
Figure 11-36. *Creating the list of suggested database objects*

■ **Note** When you set Allow SQL and PL/SQL to Yes, you're giving contributors access to all the database objects in the websheet's default schema. This is a potentially serious security issue. It's imperative that you chat with your Oracle Database Administrator (DBA) before you use this feature to make sure sensitive data isn't accidentally exposed.

Reports: Creation

After you've set up the Allow SQL and PL/SQL feature in the Application Builder, return to your websheet. You create a report by selecting New Report from the Report menu (see Figure 11-37) or by clicking the Create Report button from the View All report.
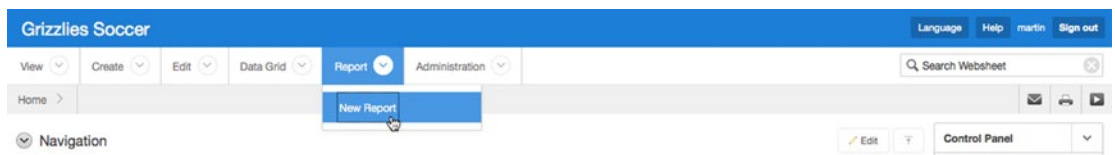
Figure 11-37. *Choosing the New Report option*

The Create Report page prompts you for one of two report sources (see Figure 11-38). The Table report source creates a report that contains all the columns in the selected table or view. The SQL Query report source (see Figure 11-39) creates a report based on an SQL statement. Using an SQL statement gives you a tremendous amount of flexibility in tailoring a report to your needs. In both cases, click Next to go to a confirmation page and check your input before creating the report.
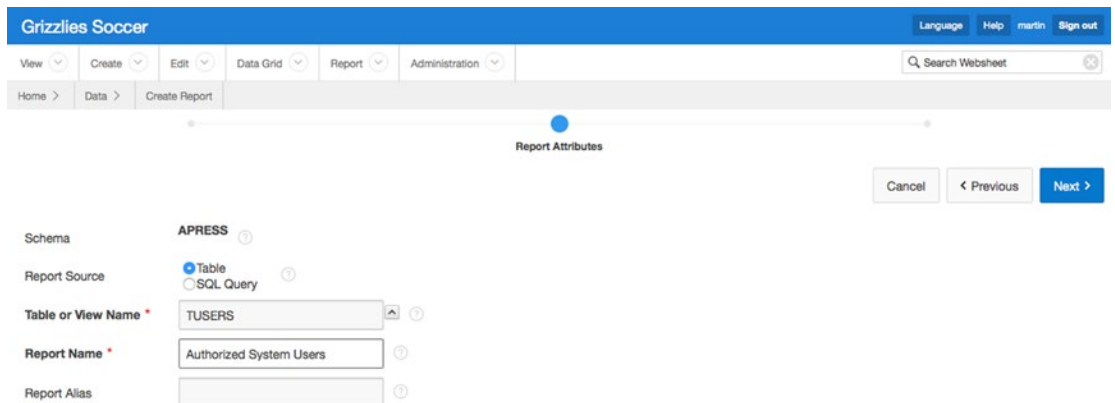
Figure 11-38. *Creating a report based on a table or view*
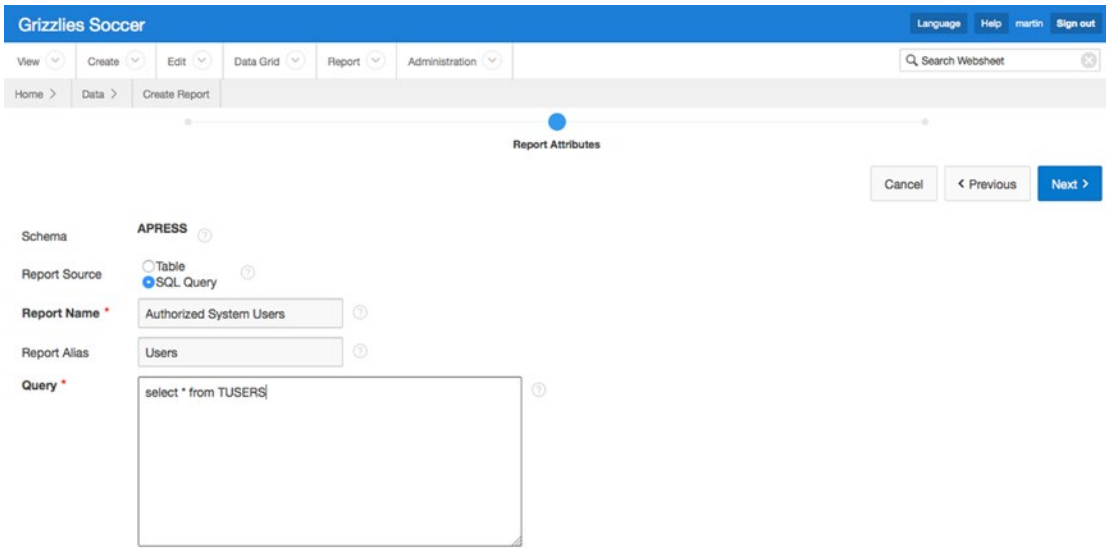
Figure 11-39. Creating a report based on an SQL statement

Reports: Accessing the Data

Users want, of course, to see the report data. Report data is exposed in three ways:

- *Navigate to a report*: Figure 11-40 shows you where to find the list of reports under the Report drop-down menu. Clicking the report's Name link in the menu takes you to the Report Data page. Figure 11-41 shows the Authorized System Users report from this example.
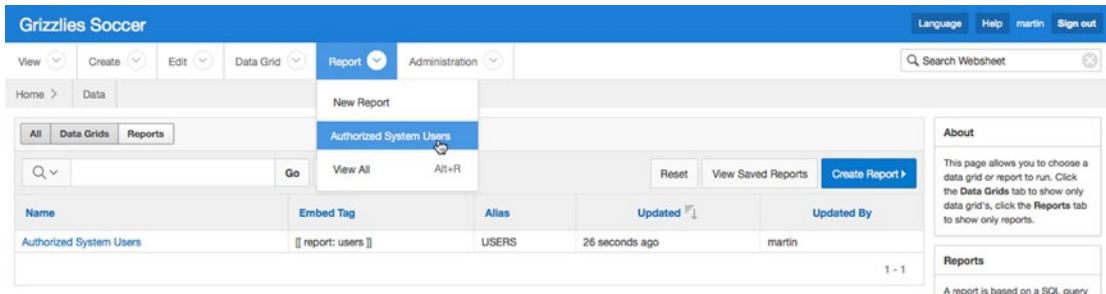
Figure 11-40. Navigating to a report

Figure 11-41. *Report data page*

- *Embed a link to the report in a text section*: The list of reports in Figure 11-40 contains an Embed Tag column. This column contains the markup syntax that you would need to add a link to the report in a text section. You simply copy and paste the markup syntax into the text section to add a link that takes you to the Authorized System Users report shown in Figure 11-41.

- *Create a data section*: Creating a data section based on a report is easy. A wizard walks you through the steps. You first navigate to the page that will contain your report and click one of the New Section links in either the drop-down menu or the Control Panel at the right side of the page (see Figure 11-42). This starts the Create Section wizard. Select the Data icon on the first page and click Next (see Figure 11-43). Now, link the data section to its data source (see Figure 11-44). In this example, you're linking the data section to the Authorized System Users report. This page allows you to link your data section to any data grid or report that you've previously created. Click Next, which takes you to the confirmation page. Once you click the Create button on the confirmation page, you will find yourself back on the content page, and your report is displayed in the new data section (see Figure 11-45).
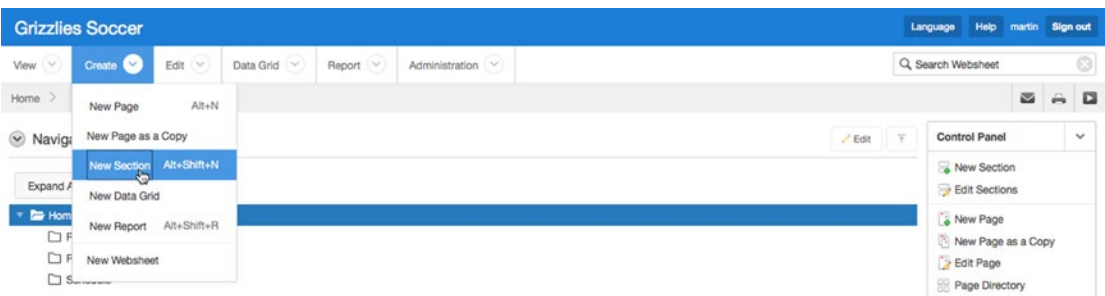
Figure 11-42. *New Section link*
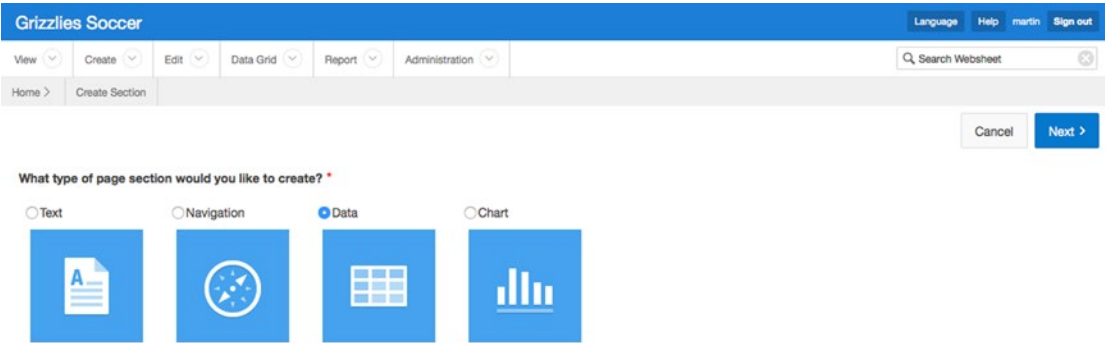
Figure 11-43. *New Section wizard: choosing a section type*
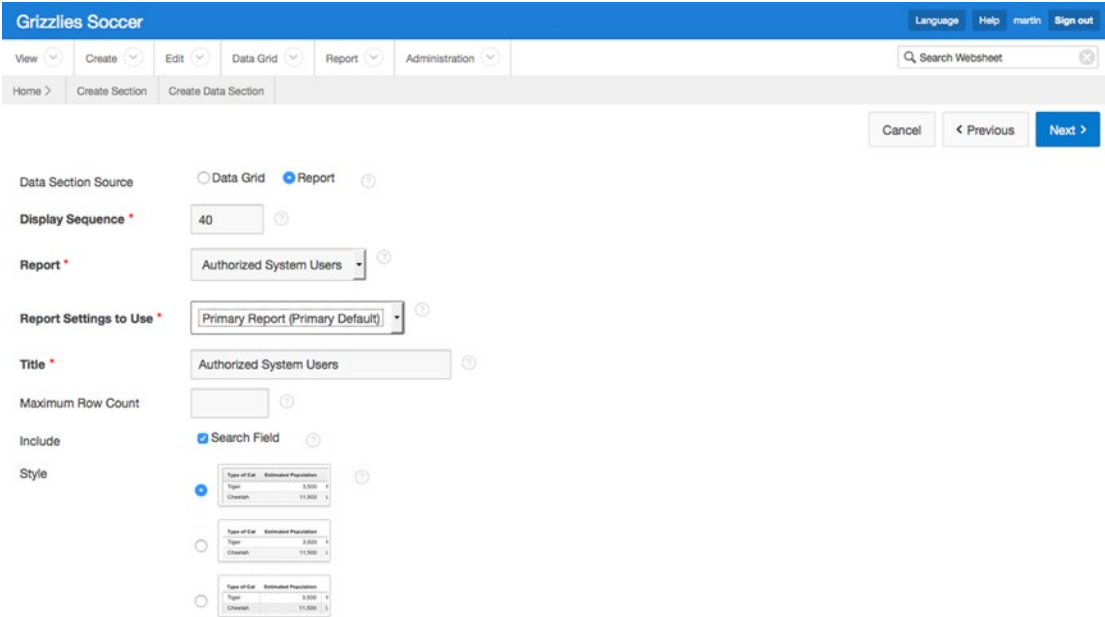
Figure 11-44. *New Section wizard: data source*
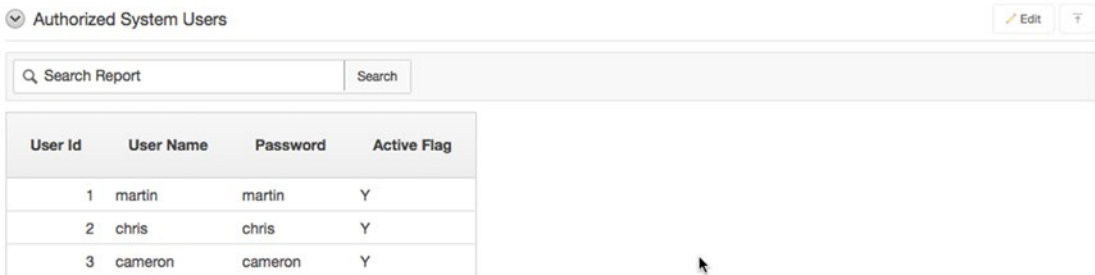
Figure 11-45. *Data section containing a report*

Chart Sections

Chart sections are an easy way to add graphics to your content. First, you must create a report or data grid that contains at least one numeric column. Second, you run the Create Chart wizard. The wizard links the report or data grid to the chart and sets up the axis labels. The details and visual result of this simple process will be covered in Chapter 12.

Annotations

Annotations are used to add additional content to your pages or individual rows in data grids. There are four types of annotations:

- *Files*: Two types of files can be uploaded into your websheet. Image files are displayed within text sections. Other file formats, such as PDFs, can be uploaded to a websheet and then downloaded to the end user's computer. In both cases, you use markup syntax to achieve the result.

- *Tags*: Tags are free-form text words that are attached to content to enhance the websheet's Search feature.

- *Notes*: Notes are like sticky notes. When you add a note, it appears in a section on the right side of the page.

- *Links*: Links allow users to navigate to any valid URL. Annotation links are associated with rows in a data grid; they can't be associated with a page. To add a link to a page, you would use markup syntax, not an annotation.

Figure 11-46 shows the Annotation section that appears on a websheet page. Clicking the various links allows you to add, change, and delete annotations.
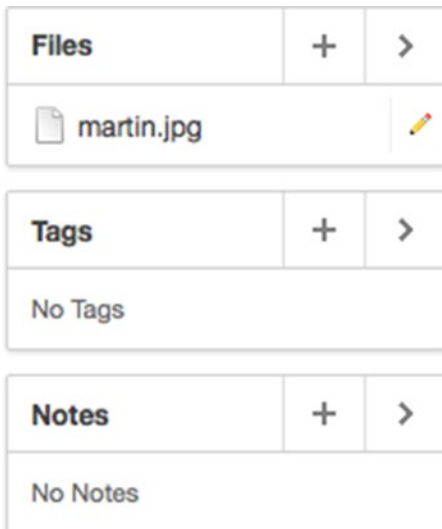
Figure 11-46. *Annotation section at lower right on a page*

Administration

Websheets, like any blog or wiki, should be reviewed periodically by a moderator who has been given the administrator role. This allows the moderator to view the Dashboard and Monitor Activity pages (see Figure 11-47). These pages and the underlying reports give the moderator the tools to see which pages are the most and least popular, how long it takes to render a page, who is using the websheet, and many other parameters that help the moderator make sure the websheet is working optimally.
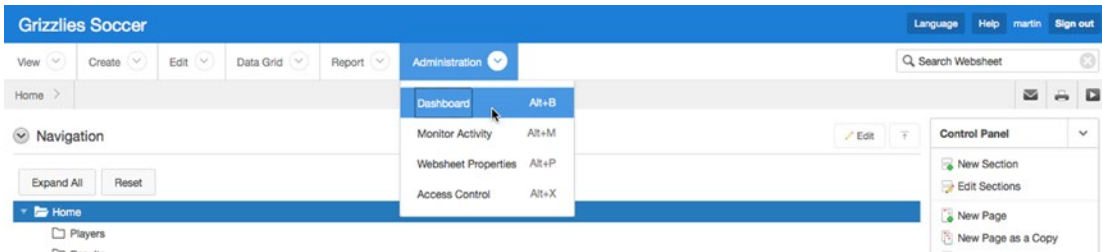
Figure 11-47. *Navigating to the Dashboard and Monitor Activity pages*

As you work through Chapter 12, notice the links that are labeled History. These History links are placed throughout the websheet's structural navigation areas. They take you to context-sensitive report pages that contain comprehensive audit trails of websheet changes. Contributors, when they know their changes are audited, will make an effort to hold their contributions to a high standard.

Summary

This chapter has given you a good look at websheets so that you can use them in innovative and creative ways. Chapter 12 complements this chapter by working through, on a step-by-step basis, the construction of a websheet from the ground up.

CHAPTER 12

■ ■ ■

A Websheet Example

The previous chapter covered many of the different features of websheets. In this chapter, you will use these features to build an application from scratch.

The example application manages a corporate soccer team. Currently, the player roster and schedules are maintained in spreadsheets. When the schedule is updated, the spreadsheet is emailed to all the players. As you can imagine, it would be very frustrating to manage a team this way.

A websheet is a good way to manage the soccer team, because websheets can be built with minimal developer or DBA assistance. All the files, along with a copy of the final application, can be found in the example download described in the introduction to this book.

Setup

In order to highlight the capability of websheets to interact with objects in your database, let's create some database objects that are referenced throughout the chapter. These objects simulate an existing users table and login function for your organization:

1. Run the following code, which is included in the example download as a script named ch12_database_objects.sql, in SQL*Plus or directly in APEX using SQL Workshop:

```
-- Create Users Table
CREATE TABLE tusers (
  user_id       NUMBER (5, 0) PRIMARY KEY,
  user_name     VARCHAR2 (10) NOT NULL UNIQUE,
  password      VARCHAR2 (10) NOT NULL,
  active_flag   VARCHAR2 (1) NOT NULL
);

-- Create sequence for IDs
CREATE SEQUENCE sn_users;

-- Create Users
-- Note: You should not store passwords in clear text.
-- This was done for demonstration purposes.
INSERT INTO tusers ( user_id, user_name, password, active_flag)
     VALUES (sn_users.NEXTVAL, 'martin', 'martin', 'Y');
```

```
INSERT INTO tusers ( user_id, user_name, password, active_flag)
     VALUES (sn_users.NEXTVAL, 'chris', 'chris', 'Y');

INSERT INTO tusers ( user_id, user_name, password, active_flag)
     VALUES (sn_users.NEXTVAL, 'cameron', 'cameron', 'Y');

-- Authentication Function
CREATE OR REPLACE FUNCTION f_login (p_username IN VARCHAR2, p_password IN VARCHAR2)
  RETURN BOOLEAN
AS
  v_count   PLS_INTEGER;
BEGIN
  SELECT COUNT (user_id)
    INTO v_count
    FROM tusers
   WHERE LOWER (user_name) = LOWER (p_username)
     AND password = p_password
     AND active_flag = 'Y';

  IF v_count = 1 THEN
    RETURN TRUE;
  END IF;

  RETURN FALSE;
END f_login;
/

COMMIT;
```

Creating and Configuring a Websheet Application

To create a websheet application, you need to have access to APEX Builder. Once you've logged in, proceed
with the following steps to create the websheet application:

1. Navigate to the Application Builder.

2. Click the Create button.

3. Select Websheet for the application type, as shown in Figure 12-1, and click Next.
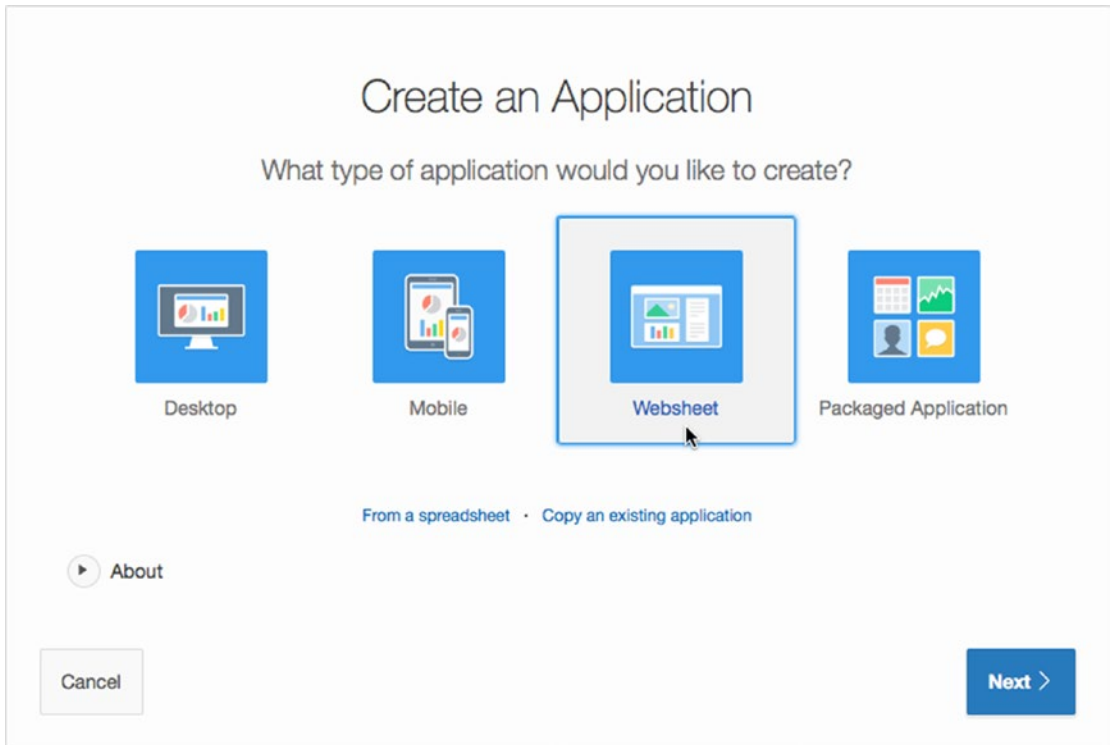
Figure 12-1. *Creating a websheet application*

4. On the next screen, enter `Grizzlies Soccer` for **Name** and use the default ID for **Websheet**. Deselect the **Include Getting Started Guide** checkbox.

5. Click the **Create Websheet** button.

You should now see a success page with the option to run the websheet (see Figure 12-2).
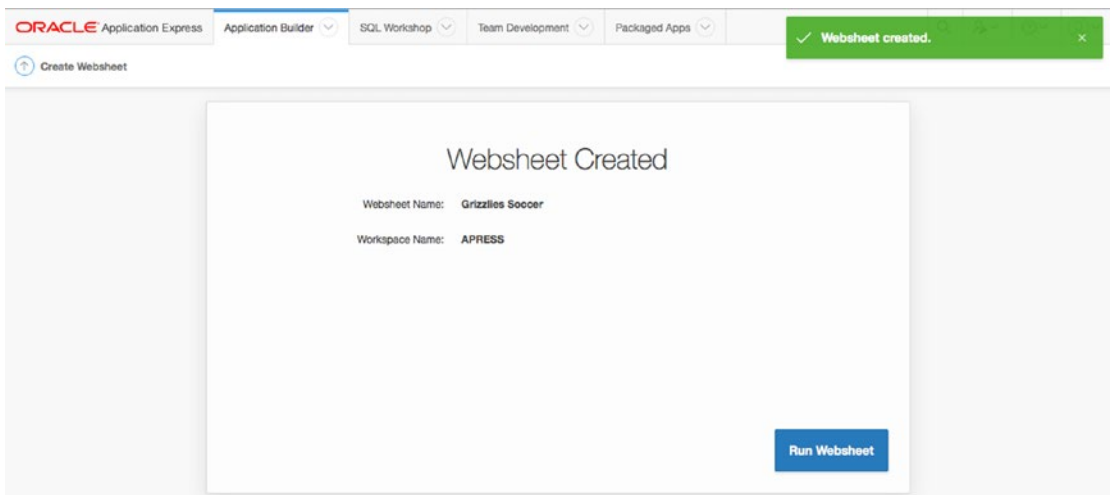
Figure 12-2. *Websheet Created success page*

The Grizzlies are your corporate soccer team, so you'd like to be able to have users log in using their current corporate accounts. To use the corporate authentication, you need to configure the application and modify the authorization scheme.

Because you haven't defined an authentication scheme, you need access to APEX Builder to modify the application properties. This is the last portion of the process which requires APEX Builder access. The following steps describe how to modify the application properties:

1. Click the **Application Builder** tab at the top of the page to return to the Application Builder home page.

2. Edit the new application you created (**Grizzlies Soccer**) by clicking on either the name or the icon.

3. Modify the items in the following sections:

   - **Attributes**: Enter DD-MON-YYYY for **Application Date Format**, as shown in Figure 12-3.

Figure 12-3. *Application date format*

   - **Authentication**: Select **Custom** as the authentication type and replace -BUILTIN - with the statement **return f_login** in the **Authentication Function** field, as shown in Figure 12-4. Leave all the other inputs at their default values. (f_login refers to the function you created in the "Setup" section of this chapter.)

Authentication

Authentication:	○ Application Express Account ⓘ
	○ Single Sign-On
	○ LDAP
	● **Custom**
Logout URL	ws?p=146:home
	ⓘ
Page Sentry Function	ⓘ
Session Verify Function	ⓘ
Pre-Authentication Process	ⓘ
Authentication Function	return f_login ⓘ
Post-Authentication Process	ⓘ
Invalid Session URL	f?p=4900:101:&SESSION.::NO::WS_ ⓘ
Cookie Name	ⓘ
Cookie Path	ⓘ
Cookie Domain	ⓘ
Secure	No ⌄ ⓘ

Figure 12-4. *Custom authentication*

- **SQL**: Select **Yes** for **Allow SQL**. This lets you reference tables and views in the underlying schema. After you select Yes, an Add Object button appears. Click **Add Object** and enter TUSERS for **Object Name**, as shown in Figure 12-5. This will allow users to quickly select the TUSERS table when creating reports. Click the **Create** button, which brings you back to the application properties page.
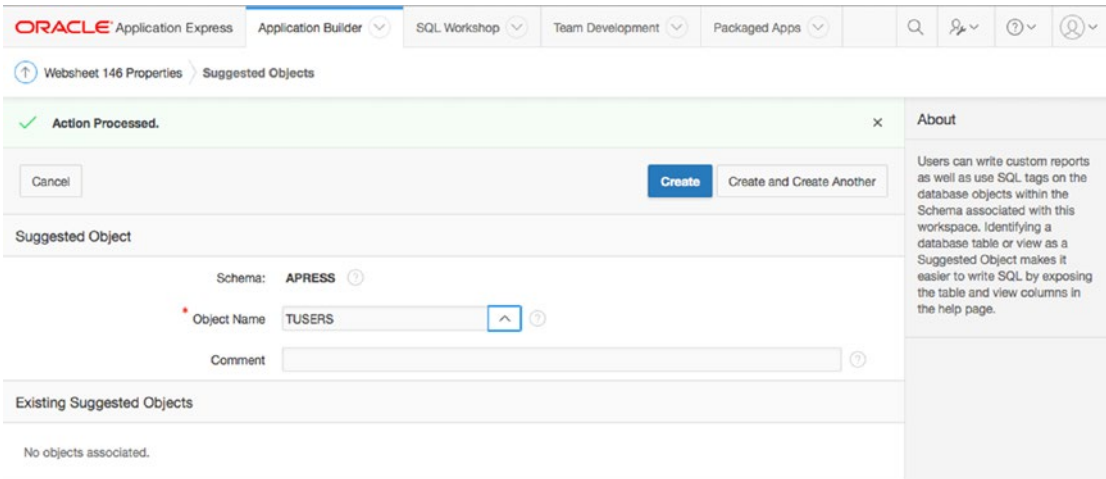
Figure 12-5. *Adding an object*

4. **Authorization**: Before you can use a custom authentication scheme, you need to define an administrator for the application. To do so, click the **Edit Access Control List** button.

On the new page, click the **Create Entry** button. In the **Username** field, enter martin, and select **Administrator** for **Privilege**, as shown in Figure 12-6. Click **Create** to register martin as an administrator. You're brought back to the Access Control List page. Click the **Cancel** button to return to the application properties page. Click **Apply Changes** button to save your changes.
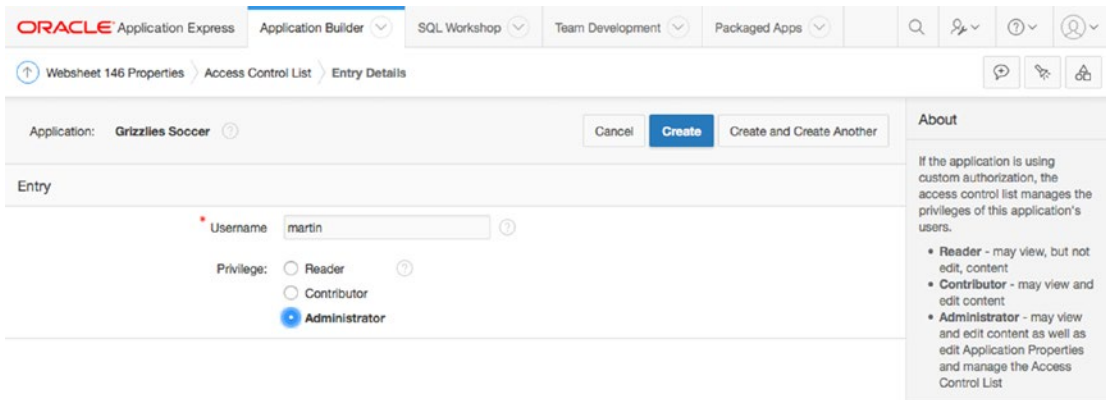
Figure 12-6. *Access Control List: adding an administrator*

Adding Content to a Websheet

In the previous section, you created and configured a websheet application to manage your corporate soccer team. In this section, you will create data grids and add content to the application.

Run the websheet application and log in as martin/martin, the site administrator. Once you've logged in, the page should look like Figure 12-7. You're now ready to create your first data grid.
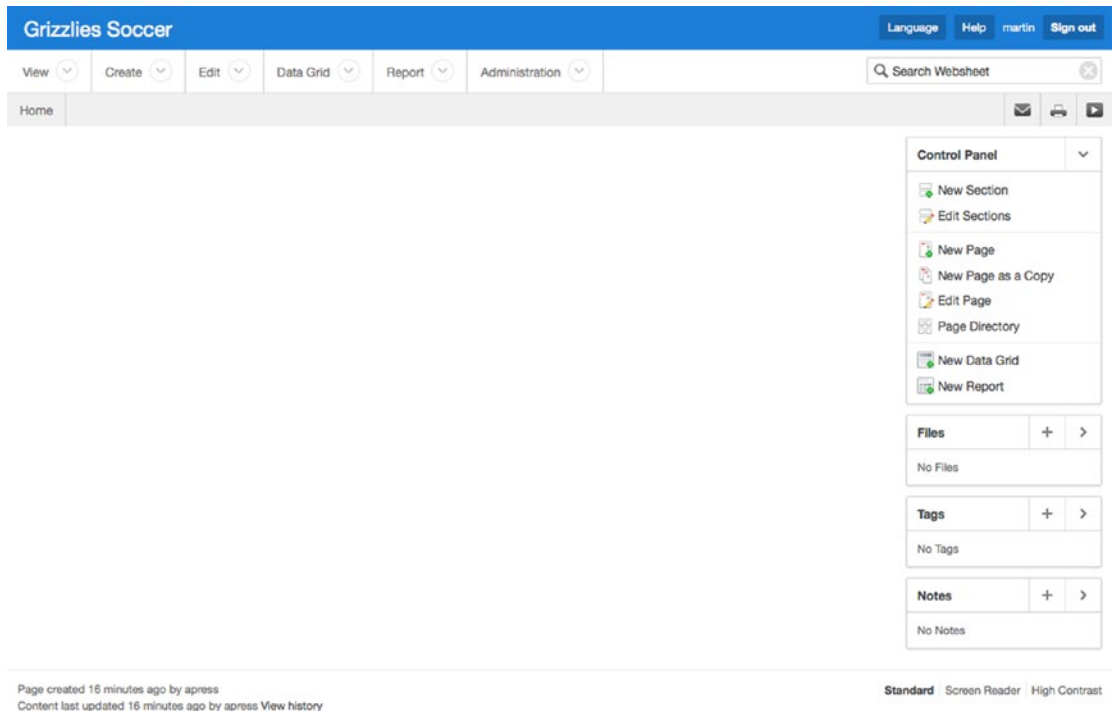
Figure 12-7. *Initial websheet application*

■ **Note**   The URL to run the websheet application is `<apex_url>/ws?p=<web_sheet_id>`. For example: `http://www.example.com/apex/ws?p=103`, where 103 is the websheet application ID.

Creating Data Grids

The first thing to do is to create some custom tables called *data grids*. Just a reminder: data grids exist only in the context of the websheet application. They don't exist as tables in a schema. There are two ways to create data grids: by pasting in existing data from a spreadsheet or from scratch by manually defining each column. In this section you will create data grids with both methods.

You currently keep the game and practice schedules in a spreadsheet. You can import data and simultaneously create a data grid using copy and paste. Here is the process to follow:

1. Click the **Data Grid** tab at the top of the application.

2. Click the **New Data Grid** option in the drop-down menu.

3. Select **Copy and Paste** as the input method and click **Next**.

4. Enter Schedule in the **Name** and **Alias** fields.

5. Open Grizzlies_Schedule.csv, which can be found in the sample code for this chapter, in Microsoft Excel and select all the fields, including the header. Copy these values and paste them into the **Paste Spreadsheet Data** text area. Ensure that the **First Row Contains Column Headings** checkbox is checked and click the **Upload** button.

6. You should now see the data in an interactive report, as shown in Figure 12-8.

	Event	Date	Time	Field	Against	Grizzlies	Opponent
✏	Practice	11-MAY-2010	18:45	Bottomlands			
✏	Practice	18-MAY-2010	19:00	Bottomlands			
✏	Practice	22-MAY-2010	10:00	Forrest Lawn			
✏	Game	26-MAY-2010	20:15	Airdrie	Airdrie	2	1
✏	Game	02-JUN-2010	20:15	Tom Brooks	Chinooks	2	3
✏	Practice	04-JUN-2010	21:00	Bottomlands			
✏	Game	08-JUN-2010	19:00	Bottomlands	Blizzard	4	1
✏	Game	13-JUN-2010	13:00	Foothills	Callies		
✏	Practice	16-JUN-2010	18:45	Bottomlands			
✏	Game	30-JUN-2010	19:15	Bottomlands	Darts		
✏	Game	07-JUL-2010	18:45	Glenmore	Vipers		
✏	Game	18-JUL-2010	15:30	Mount Royal	Darts		
✏	Game	30-JUL-2010	19:00	Bottomlands	Wolves		
✏	Practice	01-AUG-2010	11:00	Bottomlands			
✏	Game	09-AUG-2010	20:00	Oakridge	ASP		
✏	Game	20-AUG-2010	21:00	Tom Brooks	Storm		
✏	Practice	23-AUG-2010	19:15	Bottomlands			
✏	Game	25-AUG-2010	19:15	Bottomlands	Tigers		

1 - 18 of 18

Figure 12-8. *Data grid result*

You also need to create a data grid to keep track of the number of goals each player scores. You don't have any existing data ready to copy and paste from a spreadsheet, so this time it makes sense to create the data grid manually. Here are the steps to follow:

1. Click the **Data Grid** tab at the top of the application.

2. Click the **New Data Grid** menu option.

3. Select **From Scratch** as the input method and click **Next**.

4. Fill out the data grid definition, as shown in Figure 12-9, and click the **Create Data Grid** button to finish creating the data grid.

Figure 12-9. *Creating a data grid from scratch*

Applying Constraints

Now that you have data grids, you need to add some constraints to them. Because data grids aren't database objects, you must use the websheet interface to apply constraints.

In the Players data grid, you need to ensure that all the fields contain data and that the default for the Goals column is 0. To apply these constraints, follow these steps:

1. Click the **Data Grid** tab.

2. Click **Players** in the drop-down menu.

3. Click the **Manage** button and select **Columns ➤ Column Properties**, as shown in Figure 12-10.
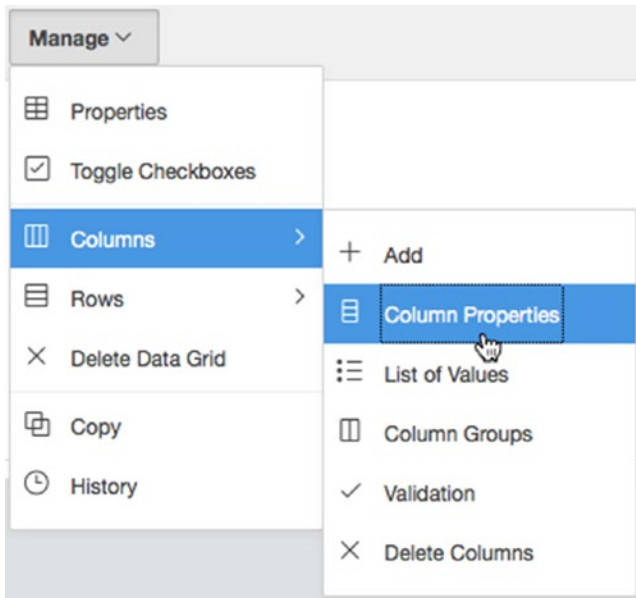
Figure 12-10. *Data grid column properties*

4. Select **Name** in the **Column Name** select list.

5. Select **Yes** for **Value Required**.

6. You must explicitly save your changes before modifying another column. Click the **Apply** button at the bottom.

7. Open the **Column Properties** section again (repeat Step 3).

8. Select **Goals** in the **Column Name** select list.

9. Select **Yes** for **Value Required**.

10. In the **Default Text** field, enter 0.

11. Click **Apply**.

Now, when you create players, both Name and Goals are required values. Goals defaults to 0.

Adding Players

To add players to the Players data grid, click the Add Row button. For this example, add the players and their goals as shown in Figure 12-11.

Figure 12-11. *Player data*

Creating Alternate Default Reports

Now that you have data in the data grids, you can create alternate default reports, which you can reference when creating sections. Data grids allow you to save reports just like interactive reports do. Alternate default reports are saved data grid reports that can be displayed throughout your websheet application.

■ **Note** Some of the reports that you'll create are date sensitive. Normally, you'd use SYSDATE as a reference point. Instead, you'll use a static date of 10-Jun-2010 to simulate a common SYSDATE. This ensures that you will see the same data as that shown in this book.

The first alternate report highlights the games and practices for the next two weeks. To create this report, follow these steps:

1. Navigate to the **Schedule** data grid using the **Data Grid** tab.

2. Click the **Actions** button and select **Filter**.

3. Select **Row** for **Filter Type**.

4. Enter Next Two Weeks for **Name**.

5. Enter the following for **Filter Expression**:

   ```
   B >= to_date('10-jun-2010', 'dd-mon-yyyy')
         and B < to_date('10-jun-2010', 'dd-mon-yyyy') + 14
   ```

6. Click the **Apply** button.

7. Order the **Date** column as ascending by clicking the **Date** column heading and choosing the **Ascending** sort icon.

8. Hide the **Grizzlies** and **Opponents** columns, as you don't have scores for future games. To do this, choose the **Select Columns** option from the **Actions** menu.

9. Choose **Actions ➤ Save Report**.

10. Select **As Default Report Settings** in the **Save** select list.

11. Select **Alternate** for **Default Report Type** and enter Next Two Weeks for **Name**.

12. Click **Apply**. The report should now look like that in Figure 12-12.

Figure 12-12. *The next two weeks*

Because you've already learned how to manipulate and create saved interactive reports, create the following alternate default reports for the Schedule data grid:

- *Remaining Games*: This report lists all the games left in the season.

- *Remaining Practices*: This report lists all the practices left in the season.

- *Results*: This report lists all the games that have been completed along with the scores.

Creating Page Sections

In this section, you will modify existing pages and create new sections. You will cover some of the different types of content that you can add to websheets.

To start, you'll modify the home page by creating several sections that help players get the most important information right away. Modify the Welcome section to contain important news by following these steps:

1. Click the **View** tab at the top.

2. Select the **Home** page in the drop-down list.

3. In the **Control Panel** at the right of the page, click **New Section**.

4. Choose **Text** as **Section Type** and click **Next**.

5. Enter Important News for **Title**.

6. Enter the following in the **Content** section:

```
Fees: Don't forget to pay your fees before the next game (13-Jun) or else you
can't play!
We won our last game and are now 2-1. Check out the[[ results | Results ]]page.
```

The special notation involving the square brackets creates a link to the Results page that you will create later in this chapter.

7. Click the **Create Section** button.
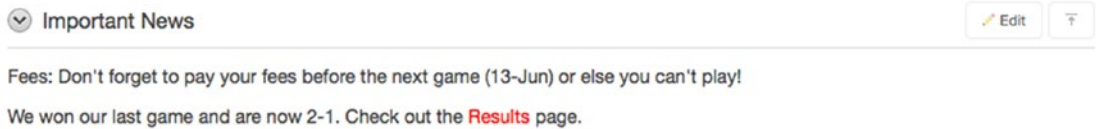
Figure 12-13. *Important news*

The home page should now look like Figure 12-13.

■ **Note**   Notice that the link to the Results page is in red. That's because you created an invalid link. Once the Results page is created, the link will turn grey.

Next, you'll create a new section on the home page to highlight the upcoming games and practices. This section references one of the alternate default saved reports that you created earlier. To create the section, follow these steps:

1. While viewing the **Home** page, click the **New Section** link at the right, located in the **Control Panel** region.

2. Select **Data** for **Section Type** and click **Next**.

3. Select **Schedule** for **Data Grid** and **Next Two Weeks (Alternative Default)** for **Report Settings to Use**. Change **Title** to Upcoming Games and Practices.

4. Select a style (use **2** for all sections in this example), and click **Next**.

5. On the confirmation page, click the **Create Section** button.

The new section should look like Figure 12-14.

Figure 12-14. *Upcoming games and practices*

Each week, the coach likes to highlight a player of the week. The coach wants to include a picture along with some text in this section. This week, Martin is the lucky recipient of the Player of the Week award. To create the Player of the Week section, follow these steps:

1. First, you need to upload the player's picture. In the __File__ region at the right, click the **Plus** link, as shown in Figure 12-15.

Figure 12-15. *Adding a file*

2. Click the **Browse**/**Choose File** button and select martin.jpg from the files associated with this chapter. Click the **Add File** button. You're brought back to the home page.

3. Click the **New Section** link.

4. Select **Text** as __Section Type__ and click **Next**.

5. Enter Player of the Week for __Title__.

6. Enter the following in the __Content__ text area:

   Martin scored 2 goals!

   [[image: martin.jpg]]

7. Click **Create Section**.

The Player of the Week section should now contain an image, as shown in Figure 12-16. Each week, the coach can easily upload a new picture and modify this section.

⌄ Player of the Week   ✎ Edit   ⊤

Martin scored 2 goals!

Figure 12-16. *Player of the Week section*

You also need a page to display the list of players and that includes a graph to show the top scorers on the team. To create and modify the player's page, follow these steps:

1. Click the **New Page** link at the right.

2. In the **Name** field, enter Players. For **Page Alias**, enter PLAYERS. Select **Home** for **Parent Page**. Click the **Create Page** button.

3. Add a new section called **Players**, which is a **Chart** section referencing the **Players** data grid.

4. To add the graph, click the **New Section** link.

5. Select **Chart** for **Section Type** and click **Next**.

6. Select **Column** for **Chart Type** and click **Next**.

7. Select **Players** for **Data Grid**, select **Primary Report (Primary Default)** for **Report Setting to Use**, and enter Goals for **Section Title**. Click **Next**.

8. Modify the chart section as shown in Figure 12-17 and click **Next**.

Figure 12-17. *Chart definition*

9. On the confirmation screen, click the **Create Section** button.

The new chart region should look like Figure 12-18.
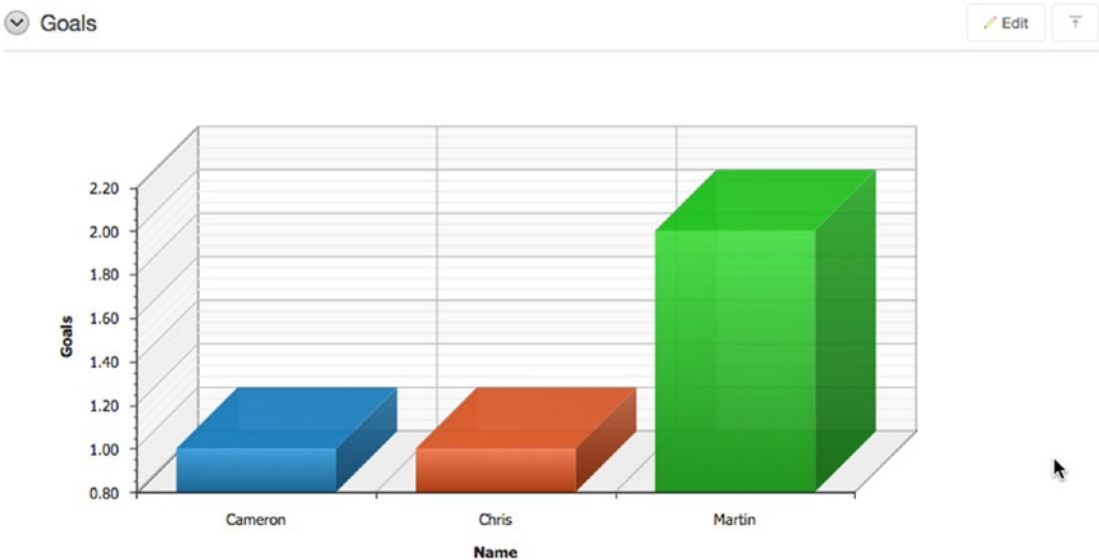
Figure 12-18. *Goals chart*

The last modification you need to make for the Players page is to add a navigation section. This allows users to quickly go to each section on the page rather than having to scroll down the page. To add the navigation section, follow these steps:

1. Click the **New Section** link.

2. Select **Navigation** for **<u>Section Type</u>** and click **Next**.

3. Select **Section Navigation** for **<u>Navigation Type</u>** and click **Next**.

4. Enter 1 for **Sequence**. Setting the sequence to 1 makes it the first section on the page. Click the **Create Section** button.

When someone views the page, they can quickly navigate to each section via the navigation section.

You should now be comfortable creating and modifying pages. Before you create the final section, create the following pages, which are child pages of the home page:

- *Results*: This page displays the Results saved report. The Results page should look like Figure 12-19.

Figure 12-19. *Results page*

- *Schedule*: This page contains two sections. The first section displays the Remaining Games saved report, and the other section displays the Remaining Practices saved report, which you created earlier. The Schedule page should look like Figure 12-20.
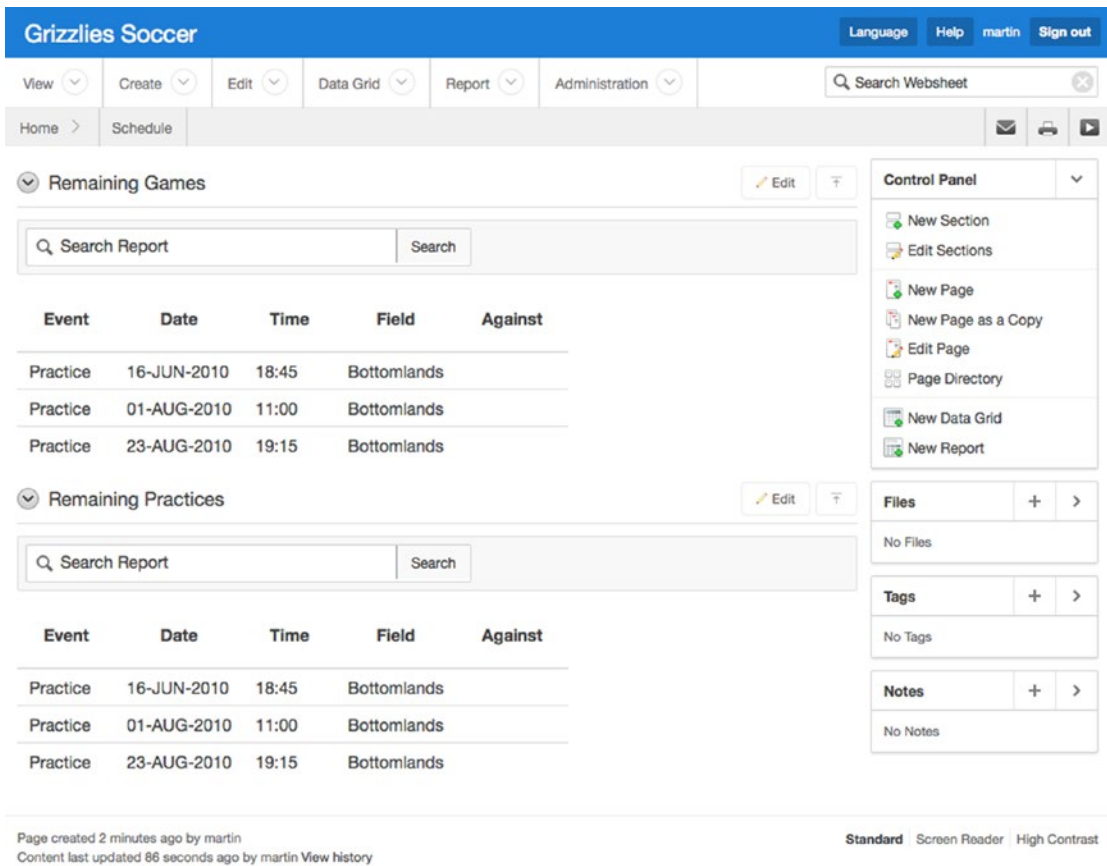
Figure 12-20. *Schedule page*

The next section you will create provides a list of all the pages, along with links to them. To create this navigation section, follow these steps:

1. Go to the **Home** page.

2. Click the **New Section** link.

3. Select **Navigation** for **Section Type** and click **Next**.

4. Select **Page Navigation** for **Navigation Type** and click **Next**.

5. If you want to, modify **Title** and set **Sequence** to 1, and then click the **Create Section** button.

The new section should look like Figure 12-21.

Figure 12-21. *Page navigation*

SQL Tags

Websheets allow administrators and contributors to query tables and views in the schema. They can create reports that are similar to data grids, except they're read-only. They can also include query results called *SQL tags* directly within sections.

On the Players page, let's add a section to display the number of registered users who have access to the application, and include a SQL tag in the section. To create the section, follow these steps:

1. Go to the **Players** page.

2. Click the **New Section** link.

3. Select **Text** for **Section Type** and click **Next**.

4. Set **Sequence** to 5 and **Title** to Active Registered Users.

5. Enter the following text in the **Content** section and click the **Create Section** button:

```
We currently have[[sqlvalue: select count(*) from tusers where active_flag = 'Y' ]]active
registered users. They are: [[sql: select initcap(user_name) "Name" from tusers order by
user_name ]]
```

The new section should look like Figure 12-22. Notice the select count query in the first line of the preceding code. That query generates the value 3 that is shown as the number of active users in the figure. Similarly, the second select statement in the preceding code generates the list of player names.

When using SQL tags, you need to explicitly define whether the query will return a single value or multiple rows and columns. A SQL tag defined as **sqlvalue:** means a single value will be returned and be embedded within a sentence such that the single value appears as a word in the sentence. Using **sql:** means multiple rows and columns will be returned. When a query returns multiple rows, its results are displayed in the spreadsheet-like format you see in Figure 12-22.
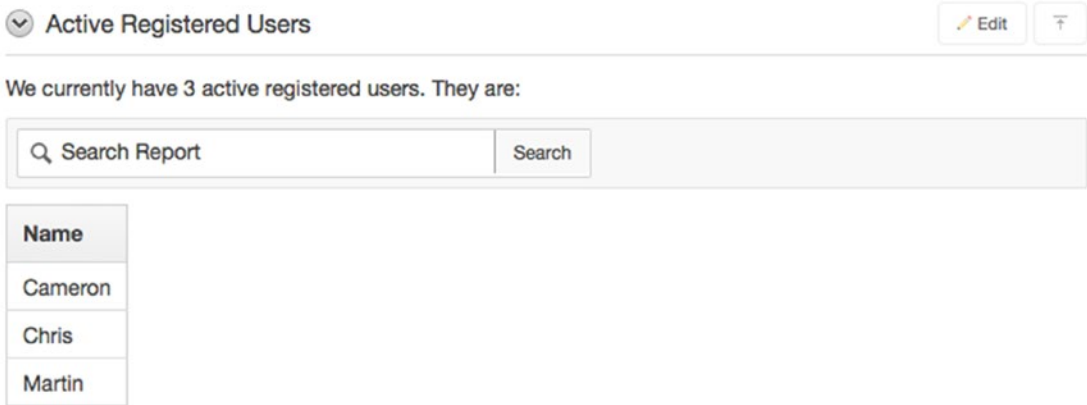
Figure 12-22. Section with SQL tags

The search box in Figure 12-22 is a result of the second query returning multiple rows. Whenever a query's results are displayed as a spreadsheet-like grid, that grid is preceded by a search box that you can use to quickly find specific result rows.

Access Controls

The last thing you need to do for the application is to give the other players on the team access to it. You already gave access to Martin when you created the application. You need to give the other players, Chris and Cameron, access to view the application. To do so, follow these steps:

1. Click the **Administration** tab at the top of the screen.

2. Click the **Access Control** option in the drop-down menu.

3. Click the **Create Entry** button.

4. Enter Chris for **Username** and select **Reader** for **Privilege**. Click the **Create and Create Another** button.

5. Enter Cameron for **Username** and select **Reader** for **Privilege**. Click **Create**.

Now Chris and Cameron can log in to the application. They can't modify any of the sections or data grids, however. If you need to give someone access to modify the ap plication, you can grant them the Contributor role in the Access Control section.

Summary

The last two chapters introduced websheets and what they can do. From here, you can go on to make complex websheet applications without having to know much about databases or SQL. Now that you have a base knowledge of websheets, installing and analyzing the websheet sample application would be a good next step to understanding the capabilities of a websheet.

CHAPTER 13

■ ■ ■

Extended Developer Tools

While developing the sample application in the previous chapters, you saw many features of the APEX development tool. This chapter will highlight advanced development features in APEX that weren't covered in the previous chapters. These features or tools may help when you're developing large applications in a corporate environment.

■ **Note**   This chapter assumes that you're comfortable with APEX and understand the fundamentals. If you're still not comfortable developing an APEX application, we strongly recommend that you revisit the examples from Chapters 5 through 9 in order to become more at ease with APEX and its development environment.

Page Locks

When developing in larger teams, development conflicts can occur. A *development conflict* is when two developers are working on the same object at the same time and overwrite each other's changes.

■ **Note**   For the remainder of this chapter, references to APEX objects imply page items, regions, lists, pages, and so on.

Conventional web development tools, such as ASP, PHP, and JSP, contain multiple files that each represent a page or a set of functions in the web application. When developing with these tools, it's common practice to use a source-control tool, such as Subversion, to manage all the changes. Source-control tools can easily manage development conflicts between multiple developers, because the conflicts are isolated to a single file.

APEX is different than the scripting languages just mentioned because developers don't work with files. All the information is stored in tables in the database. When you create an export of an application, you get a single SQL file that loads the metadata for the application into these tables. Because APEX stores its content in the database, you can't use traditional source-control tools to manage conflicts when developing in teams with multiple developers.

APEX Conflicts

To demonstrate a development conflict in APEX, imagine that you have two developers, Mina and Natalie, working on the same page in an application. If they're both adding and modifying page components at the same time, the page may not behave as expected for either of them.

APEX prevents developers from modifying the same object at the same time by performing *optimistic locking*. Table 13-1 shows the sequence of events that occurs as the two developers edit the same object. You can see at the end how APEX prevents Natalie from overwriting the changes made by Mina.

Table 13-1. *Optimistic Locking Scenario*

Step	Mina	Natalie
1	Edit `P1_EMPNO`	--
2	--	Edit `P1_EMPNO`
3	Edit help text to: "Mina's Help"	--
4	--	Edit help text to: "Natalie's Help"
5	Apply changes	--
6	--	Apply changes
7	--	Receive error message: `Current version of data in database has changed since user initiated update process. Current row version identifier = "A08A505E601932E33BC1074BEA1A3B4C" application row version identifier = "AECE767E4BDDC737A7823083A31D564F" Contact your application administrator.`

Optimistic locking only works when developers modify the same object. The problem occurs when multiple developers are modifying different objects on the same page at the same time. Modifying one object may affect the process of the entire page, which other developers may not be aware of. *Pessimistic locking* helps prevent trouble in that scenario. The next section will discuss how to do pessimistic locking.

Locking an APEX Page

The easiest way to prevent issues from occurring when developing an application with multiple developers is to lock a page before working on it. Locking a page prevents other developers from modifying the page while you're working on it. Developers can still view the page and its components while a page is locked; they just can't make any modifications to the page.

The following process locks a page:

1. In the **Page Designer**, click the **Lock** icon in the Page Designer Toolbar, as shown in Figure 13-1.
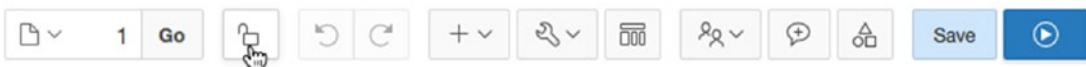
Figure 13-1. *Locking a page*

> You will see a pop-up dialog that allows you to enter a comment or reason as to why you are locking the page.

2. Enter a value for **Comment** (all page locks require a comment) and click **Lock**. The dialog will be dismissed, and the lock icon will have changed to solid green with a closed padlock.

Entering meaningful page-lock comments is important, because a history of page locks is maintained. If you use a case-management tool, it's smart to reference the case number that you're working on when locking a page.

If another developer views a locked page, they see the lock icon as solid red with a closed padlock. Clicking on the lock icon will open a pop-up dialog indicating who locked the page and the comment they entered at the time. Figure 13-2 shows an example of a locked page and the pop-up dialog.
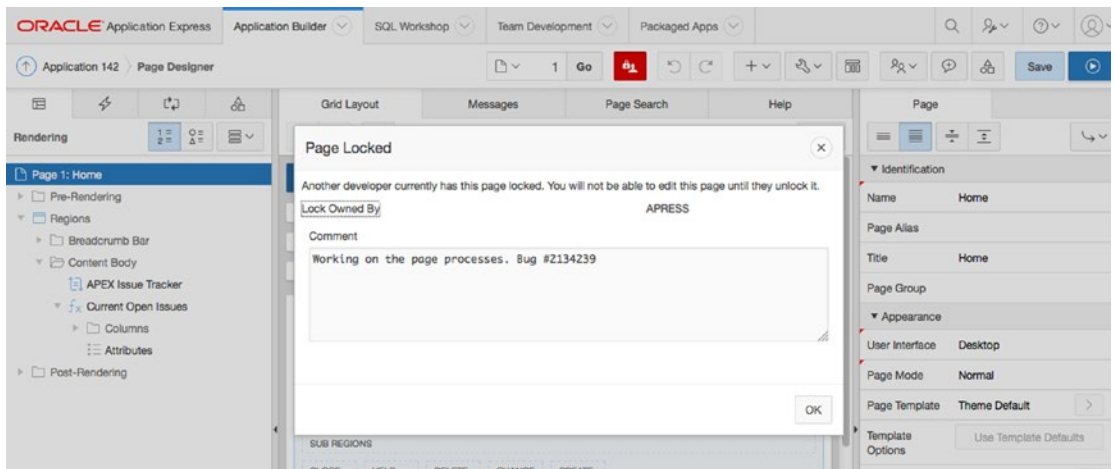
Figure 13-2. *Locked page*

Unlocking a Page

Only APEX administrators, workspace administrators, and the developer who locked a page can unlock it. If you're the developer who locked a page, the following process demonstrates how to unlock it:

1. Go to the locked page (page 1 from the previous example) and click the **Lock** icon in the Page Designer Toolbar.

2. You'll see the pop-up dialog showing the current lock comment. From here you can either alter your comment and save the changes or click the **Unlock** button to unlock the page.

Administering Page Locks

Developers may want to see all the pages that are locked, or they may want to lock/unlock multiple pages at the same time. APEX provides tools to handle multiple page-lock requests. To view the Page Locks report, in the Page Designer Toolbar go to Utilities ä Cross Page Utilities and then choose Page Locks, as shown in Figure 13-3.
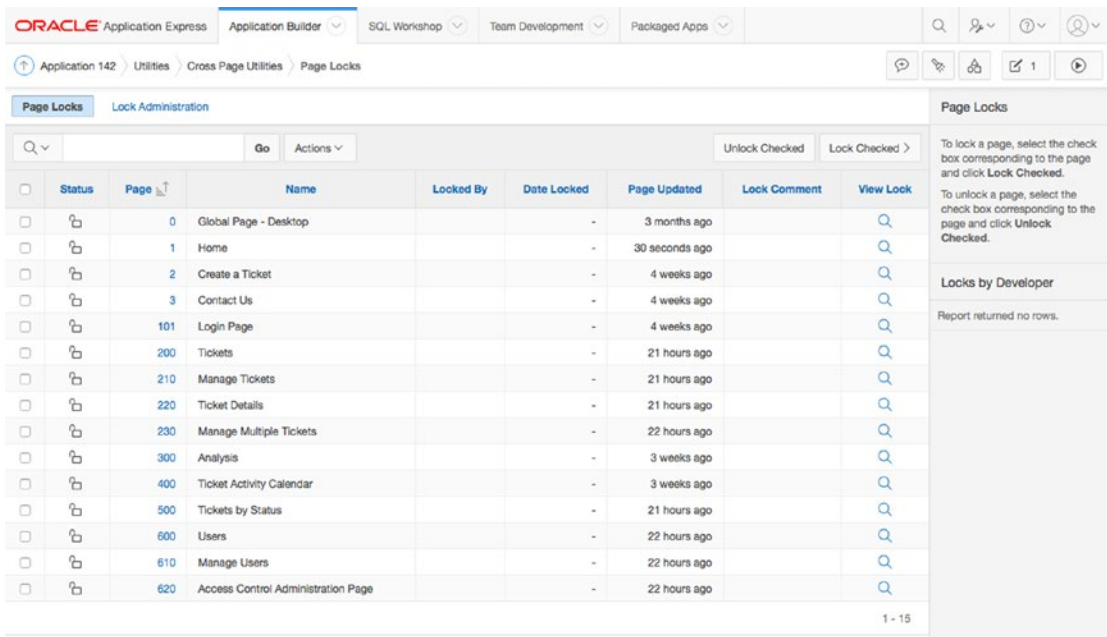
Figure 13-3. *Page Locks report*

From the Page Locks report, you can view all the pages that are locked and unlocked. You can lock and unlock multiple pages from this report. Workspace administrators can unlock any page, but developers can only unlock pages that they locked. Developers can also view the lock history for each page by clicking the View Lock icon, which looks like a magnifying glass.

■ **Note**   Page locks aren't maintained in the new version when an application is copied or exported. This means any page locks and comment histories are only relevant to the specific application in the workspace. If you copy an application into another workspace, nothing is locked in that new copy.

Application and Page Groups

Developing large APEX applications may require you to group applications and pages. APEX allows you to declaratively group applications and pages in applications. Grouping pages and applications can help you avoid the need to have strict application and page naming and numbering schemes.

Application Groups

When you have multiple applications in a workspace, you may want to group associated applications to help developers visualize which applications are related. For example, suppose you develop a large CRM system that consists of three modules: Marketing, Service, and Sales. For various reasons, you may want to create each of these modules in its own application and have a common Admin module that links the applications together.

If your workspace contains other suites of applications, developers may get confused about which suite of applications they're working on. To resolve this issue, you can create application groups. Here is the process to create an application group:

1. Navigate to the **Application Builder**.

2. Click the **Workspace Utilities** icon.

3. Click the **Application Groups** option from the **Workspace Utilities** menu.

4. Click the **Create** button.

5. Enter **Name** and **Description** values for the group. In this example, use CRM. Click **Create**.

The following steps demonstrate how to add individual applications to an application group:

1. On the **Application Groups** page, click the **Manage Unassigned** link at the right in the **Tasks** region.

2. Select the group in the **New Group** select list and then choose the applications by selecting the checkboxes. Click the **Assign Checked** button, as shown in Figure 13-4.

Figure 13-4. *Assigning application groups*

3. Go to the Application Builder main page and view the applications in a report format. You can use the Actions menu to add the Group column to the report. This allows you to view the list of applications, along with their application group, as shown in Figure 13-5.
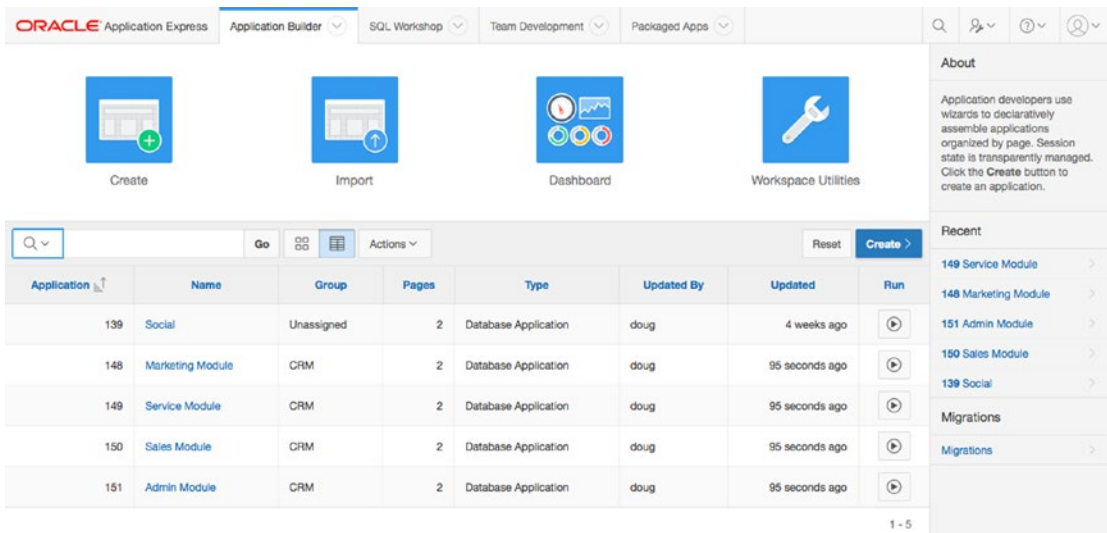
Figure 13-5. *Application report*

Page Groups

Page groups are similar to application groups except that they group pages together. They're application specific, which means a page group is only valid for a particular application. Page groups are very useful to help group common pages in an application.

An alternate approach to using page groups is to use a numbering scheme to group pages together. The page-number approach may not always work, however, because you may run out of numbers. For example, imagine that you group logical pages in sets of 10. What happens when you have 11 pages in a group? Of course, a workaround is to create large intervals for each logical page group, but even so you may run into a situation where a page doesn't conform to your numbering standards.

■ **Note**　You can also use page groups for purposes other than in the development environment. Suppose you grouped all the admin pages into an Admin page group. If you had a Global Page region that you wanted to appear only on admin pages, you could add a condition to check that the current page was associated with the Admin page group.

To create and manage page groups, from the Application page in APEX go to Utilities ➤ Page Specific Utilities ➤ Page Groups. On the Page Groups page, you can create and assign page groups in a way similar to how you created application groups in the previous section.

APEX Views and the APEX Dictionary

In traditional web development tools, if you need to search through all your code, you must comb through multiple text files. APEX is different because it stores code in the database. Thus, you can run queries to search through your code.

The APEX Schema

A common misconception about APEX is that it's an extra piece of software that you need to install. In fact, APEX is a framework that is stored in a schema in the database. At a very high level, each time you request a page, APEX queries the tables in its schema and executes many invocations of the HTP.P procedure to produce the HTML that is sent to the browser.

Each time you create an object in APEX, it's stored in a table in the APEX schema. For example, when you create a page item, it's stored in the APEX_050000.WWV_FLOW_STEP_ITEMS table.

■ **Note** As mentioned in Chapter 1, APEX was originally called *FLOWS*, and pages were originally called *STEPS*, which is why some of the table names contain these references.

Storing code in the database has advantages and disadvantages. An advantage is that you can query the database to quickly find what you're looking for in an organized fashion. For example, you can easily search through all your reports for a certain table or column reference. A disadvantage is that it's harder to search through all the objects in an APEX application at the same time. At last count, the APEX_050000 schema has over 450 tables. Searching through an entire application is easier in file-based web applications, because you can do a simple text search.

APEX Views

The APEX views give the developer visibility into the metadata that makes up the applications in the current workspace. There are several ways to access the data from these views. This section will discuss how to use these views to search in the APEX development environment.

To access the APEX views in the development environment, navigate to the Application Builder home page and then go to Workspace Utilities ➤ Application Express Views. If you're new to APEX, we recommend that you change the default view mode to View Report, which provides a detailed description of each of the reports (see Figure 13-6).

Figure 13-6. *APEX views*

365

The following example demonstrates how the APEX views can assist you. Suppose you need to compare the help text for items that reference *PRODUCT_ID*. Follow these steps to do this:

1. Go to **Workspace Utilities ➤ Application Express Views**.

2. Click the **APEX_APPLICATION_PAGE_ITEMS** view. You may need to search for this view in the interactive report or navigate to the next page.

3. On the **Select Columns** screen, add ITEM_NAME and ITEM_HELP_TEXT to the **Selected** column and click the **Filter** button, as shown in Figure 13-7.
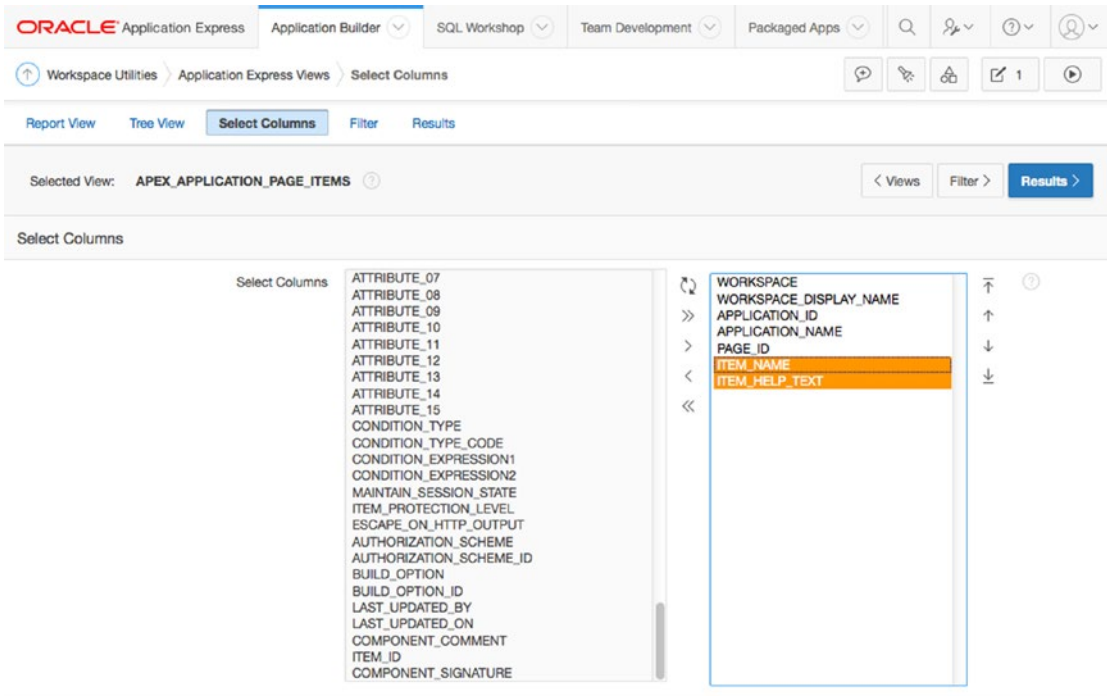
Figure 13-7. *Select Columns view*

4. On the **Filter** page, select **ITEM_NAME** for **Column**, select **LIKE** for **Condition**, and enter '%TICKET_ID%' for **Value**, and click the **Results** button, as shown in Figure 13-8. It's important to include the single-quote characters around your search value just as you would in a SQL query.
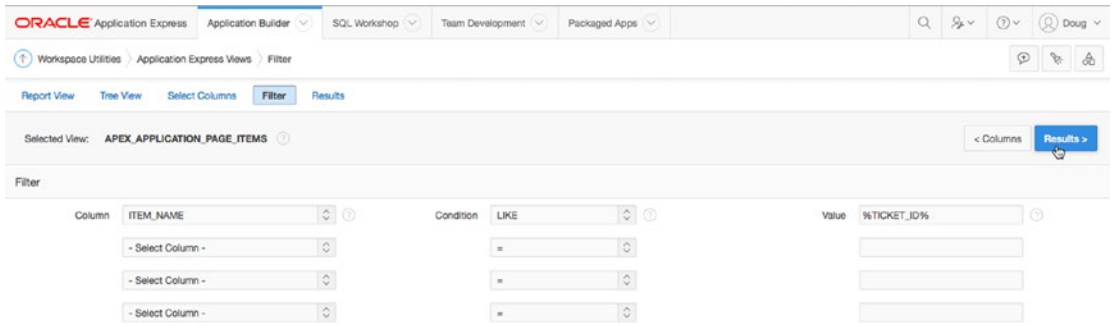
Figure 13-8. *Filter view*

5. You should now see the list of all page items in the workspace that have *TICKET_ ID* as part of the item name. Based on these results, you can modify the items that need to be changed. Because the report shows all applications in the workspace, you may want to apply an additional filter for a specific application.

Alternatively, you can view the list of APEX views as a tree by clicking the Tree View tab shown in Figure 13-9. The tree view provides an excellent method of understanding how one view relates to another. Once you click your desired view, you can continue with the process just described to display the results.
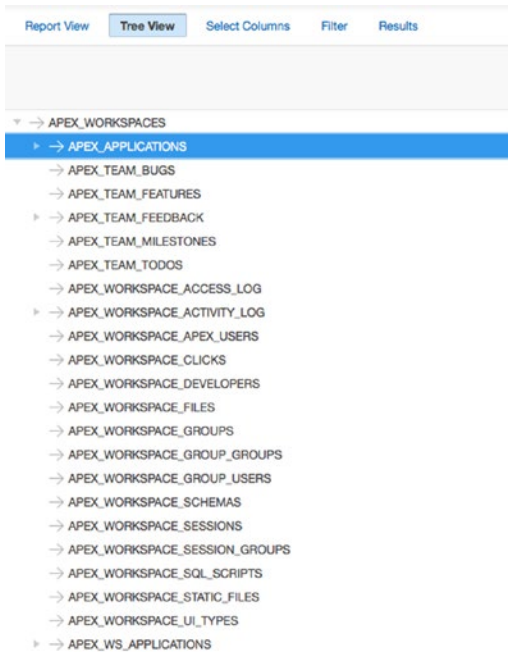
Figure 13-9. *Tree view*

APEX Dictionary

Because all the data resides in the database, you can reference the APEX views from SQL queries. As you become more familiar with the APEX views, you may prefer to query them using SQL because you can quickly apply predicates and you don't need to use a web browser to run reports.

APEX provides a view called APEX_DICTIONARY that lists all the APEX views. The APEX Dictionary contains all the information that is available in the developer GUI as well as descriptions for each of the columns. The following query lists all the APEX views:

```
SELECT *
  FROM apex_dictionary
 WHERE column_id = 0
```

There is a slight difference when querying the APEX views in SQL compared to the GUI. When you use the developer GUI, only applications that reside in the workspace appear in results. When you query through SQL, only applications whose parsing schema is the same as your connection appear in results. If you connect as SYS, SYSTEM, or a user who has been granted the APEX_ADMINISTRATOR_ROLE, you can view all the applications regardless of the application's parsing schema.

In the previous section, the example described how to look at all the help text for items with *TICKET_ID* in their name. You can view the same set of results by running the following query:

```
SELECT workspace,
       application_id,
       application_name,
       page_id,
       page_name,
       item_name,
       item_help_text
  FROM apex_application_page_items
 WHERE item_name LIKE '%TICKET_ID%'
```

Searching isn't the only use for the APEX views. You may also need to use the APEX views when you're creating plug-ins or adding advanced features to an application.

Searching in APEX

The previous section examined how to use the APEX views to search for items in your application. In this section, you will learn alternative ways to search through an application

APEX Finder

APEX provides a tool in the application that provides some reports that use the APEX views on common APEX objects. To access the APEX Finder, navigate to any application's main edit page (the one where APEX shows you the list of all pages in the application). On this page, click the Find icon (which looks like a flashlight and is located in the upper-right corner), as shown in Figure 13-10.
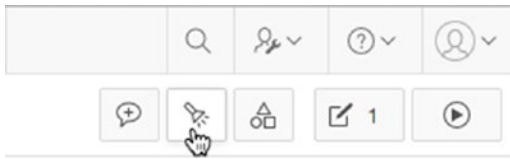
Figure 13-10. *Find icon*

Clicking the Find icon opens a pop-up window that contains interactive reports for the following object types:

- Application and page items

- Pages

- SQL queries from report regions

- Database tables in the parsing schema

- PL/SQL packages, functions, and procedures in the parsing schema

- Images, including standard images, workspace images, application images, and Font Awesome icons

- A list of debug log entries

- Application items, page items, and collections and their values in the current session

- A list of errors that have occurred in this application at both the region and page levels

The APEX finder is helpful because it allows you to quickly search for something while you're in the Application Builder and doesn't require you to leave the page. If you need more complex searches or filters, we recommend querying the APEX views.

Search Application

The APEX Views Utility, which queries against the APEX views, and the APEX finder are good tools if you know exactly what you're searching for in an object type. For example, if you want to see whether a particular table was referenced in a query, then you can search the APEX_APPLICATION_PAGE_REGIONS view for the table name in the REGION_SOURCE column.

What if you want to search an entire application to see whether it references a specific table? Suppose, for example, that you rename the TICKETS table to ISSUES. How can you easily search your entire application for any reference to TICKETS? The answer is to use a feature called Search Application, which searches through the entire application.

To search the entire application, enter your search criteria in the Search Application field by clicking the Magnifying Glass icon (located in the upper-right corner—see Figure 13-10). Figure 13-11 shows the detailed results of all occurrences of *TICKETS* in the application. For each result, a link is provided to the exact location of the result. Using the Search Application feature, you can easily find all references to the TICKETS table and replace them with ISSUES.
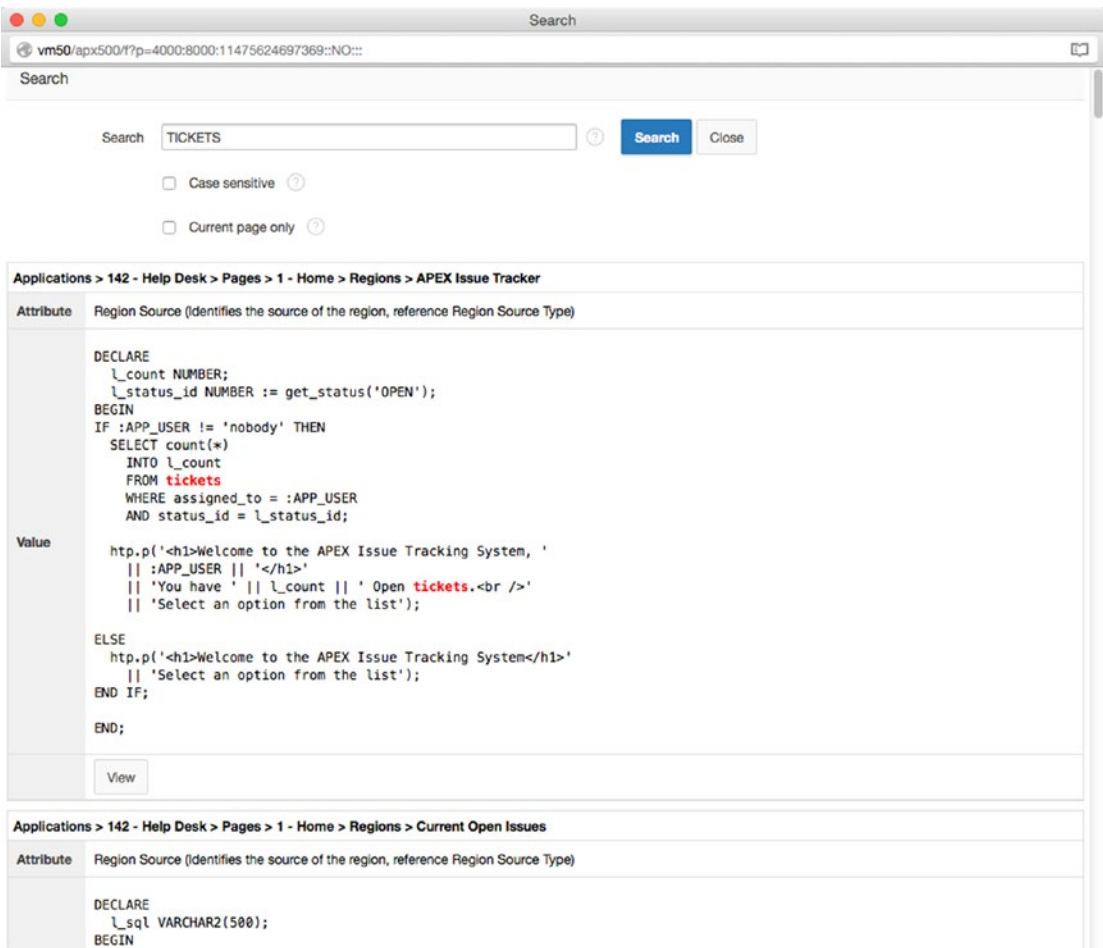
Figure 13-11. *APEX Search Application results*

The problem with the results in Figure 13-11 is that they contain results for all text that contains *TICKETS*. This includes labels, HTML, and so on. Although this may not seem like a problem, consider searching for the EMP table. Your results might contain things such as t*emp*late, *emp*loyee, *emp*no, and more. Because you only want references to the table with respect to SQL queries and PL/SQL blocks, you might want to exclude all occurrences of your search where its previous or next character is alphanumeric.

Regular expressions, which are supported by the APEX Search Application tool, can be used to accomplish this. The search criteria must be prefixed with regexp: in order to use regular expressions. Figure 13-12 shows Search Application when a regular expression is used to filter out occurrences of *emp* where its previous or next character is alphanumeric. To learn more about regular expressions and Oracle's implementation of them, refer to the Oracle database documentation. Look in particular at the *SQL Reference* manual.

Search

| Search | REGEXP:(^|[^[:alpha:]])emp([^[:alpha:])]$) | ? | Search | Close |

☐ Case sensitive ?

☐ Current page only ?

Figure 13-12. APEX search using a regular expression

Monitoring Your APEX Application

APEX can log each page access and login attempt. Logging is an excellent feature to enable, because it allows you to monitor your application and provides a way to help reduce errors and improve performance. This section will show you how to enable logging, some uses for the activity log, and how to view all login attempts.

Enabling Logging

By default, logging is enabled when you create an application. To verify that logging is enabled for your application, go to Shared Components and click the Application Definition Attributes link in the Application Logic region at the top left. In the Properties section is a Logging option, as shown in Figure 13-13. Ensure that it says Yes and click the Apply Changes button.

Figure 13-13. Enabling logging

371

For an application that has many page hits, you may want to disable logging, as it can slow down the application. Most applications don't have this issue, but it's important to know that the problem might occur.

■ **Note** Logs are stored in underlying APEX-owned tables and are purged at regular intervals. The default is to keep logs for 14 days before they're purged, but an instance administrator can increase this value to 180 days. It's recommended that if you wish to retain this data for longer periods, you set up a nightly job to copy it to your own schemas. An example of storing a local, permanent, log history is shown in the blog post at www.talkapex.com/2009/05/apex-logs-storing-log-data.html.

Using the Activity Logs

Each time a page is accessed, a log entry is stored. You can reference it from the APEX_WORKSPACE_ ACTIVITY_LOG view. A good example of how to mine the activity log is to search for errors in an application. No matter how hard you try, unhandled errors occur. Instead of waiting for users to report these errors (assuming that they even report errors), you can take a proactive approach. The following query identifies when an error occurs at the page or region levels:

```
SELECT *
  FROM apex_workspace_activity_log
 WHERE error_message IS NOT NULL
```

Once an application has been running for a while, you may notice that some pages are accessed more often than others and some pages aren't performing as desired. The following queries identify these two cases:

```
-- Find most accessed pages
  SELECT application_id,
         application_name,
         page_id,
         page_name,
         SUM (page_id) AS page_hit_count
    FROM apex_workspace_activity_log
GROUP BY application_id,
         application_name,
         page_id,
         page_name
ORDER BY SUM (page_id) DESC

-- Find slowest pages
-- Note: This depends on how you calculate slow
  SELECT application_id,
         application_name,
         page_id,
         page_name,
         ROUND (AVG (elapsed_time), 5) AS avg_elapsed_time,
         SUM (page_id) AS page_hit_count,
         MEDIAN (elapsed_time) AS median_elapsed_time
```

```
    FROM apex_workspace_activity_log
GROUP BY application_id,
         application_name,
         page_id,
         page_name
ORDER BY 5 DESC
```

By identifying the most-accessed pages, you can focus your attention on trying to speed them up. Slow pages may require tuning, but if they're accessed infrequently, you may not need to spend a lot of time on them.

Here are some other examples of uses for the activity log:

- *Top browsers*: If you build your application to support Firefox and IE and then find that half your users are using Chrome, you may want to invest some time ensuring that your application supports Chrome.

- *The time frame when people are using your application*: This gives you an idea of the best time for maintenance and upgrades. You can also derive the peak usage times.

- *Search criteria in interactive reports*: If there is a consistent search pattern, perhaps you need a better report or preset filters.

Login Attempts

The APEX_WORKSPACE_ACCESS_LOG stores all the login attempts to your APEX applications. The access log can be extremely useful when you're debugging user-authentication issues.

An example of utilizing the access log is to monitor invalid login attempts. When a user attempts to log in with invalid credentials, it's not recommended that you display the exact reason why their login attempt failed. You don't want to tell the user the exact reason, because it could reveal valuable information, such as whether the user exists. It may still be important for your operations team to know why a user wasn't able to log in, in case they need to resolve the issue. Because all login attempts are stored in the access log, for a failed login attempt you can see exactly why a user wasn't able to log in.

■ **Note**   If you create your own authentication process, you should use the APEX_UTIL.SET_AUTHENTICATION_ RESULT and APEX_UTIL.SET_CUSTOM_AUTH_STATUS procedures to ensure that you populate the access log with meaningful messages. For more information on these authentication procedures, please read the APEX API documentation.

APEX Advisor

The APEX Advisor is a tool that executes predefined checks against an application. These validations can help reduce errors in your application before it's tested or goes into production.

■ **Note**   Prior to APEX 4, the APEX Advisor was an open-source project developed by Patrick Wolf. During the development of APEX 4.0, Patrick joined the APEX team at Oracle and included the Advisor as a built-in tool. The open-source version of the APEX Advisor is available at http://essentials.oracleapex.info/.

To use the Advisor, edit an application and from the main application menu go to Utilities ä Advisor. Figure 13-14 shows all the checks you can perform. Hovering over each validation displays a brief description of the validation. You have the option to restrict the pages that the Advisor reviews by defining a comma-delimited list of pages to search for in the Check Pages region located at the bottom of the page. Once you select the checks to perform, click the Perform Check button. The results page provides a list of detailed issues that the Advisor finds plus links to each of the objects.
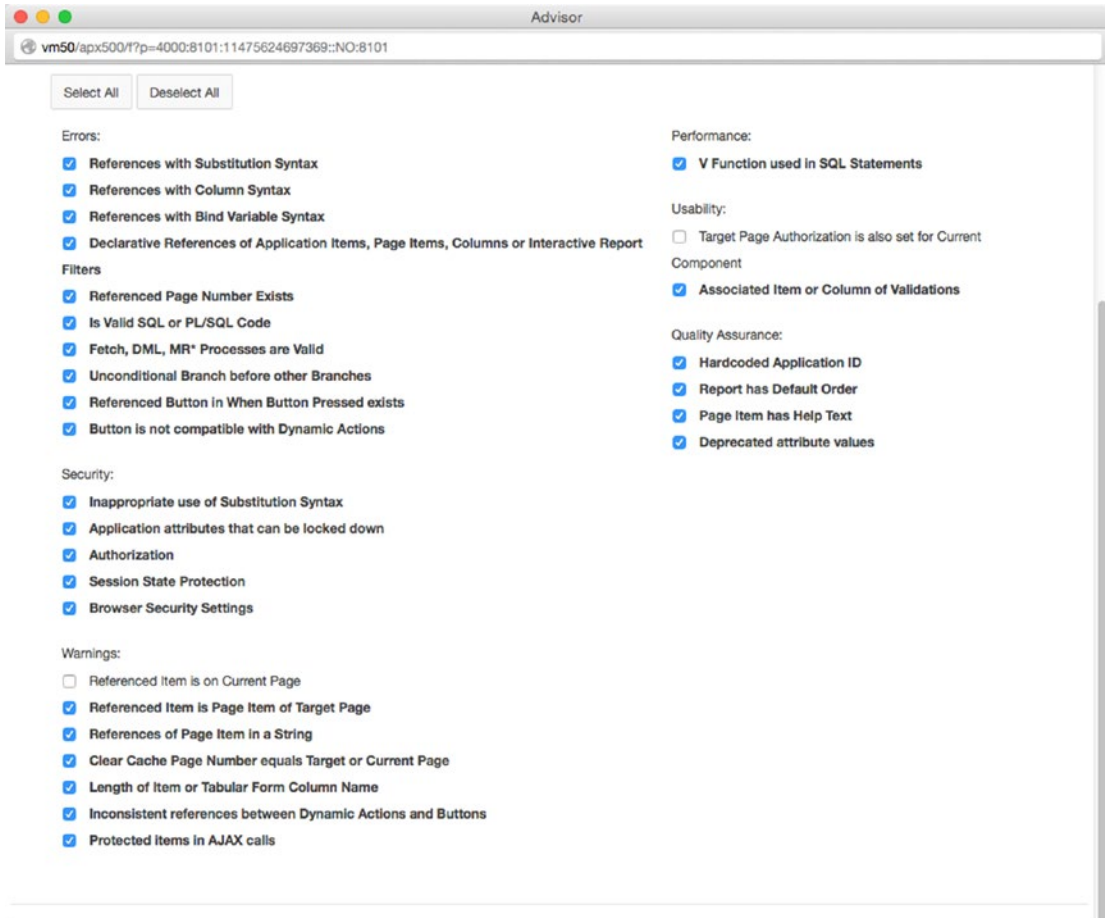
Figure 13-14. *APEX Advisor options*

The Advisor is an excellent tool to help you detect issues before your application is deployed to end users. It's still important to have development standards and a release process to help prevent issues. You should be aware that the Advisor might produce false positives in response to some of the business rules in your organization, so you should analyze each suggestion before fixing it.

Build Options

Build options let the developer conditionally include or exclude certain features of the application at runtime. Build options are either enabled or disabled for the entire application and can only be changed in the Application Builder. This means they aren't runtime configuration options.

Understanding the Need

Suppose you're working on a custom authentication scheme and you want to verify that the appropriate authentication results and custom status messages are populating in the activity log for invalid login attempts. Each time you attempt a login, you could switch programs and run a query against APEX_WORKSPACE_ACCESS_LOG. This process might get cumbersome, because you'd have to toggle between two applications. An alternate solution is to create a report on the Login page to display the most recent login attempts. After each bad login attempt, you can return to the Login page and see an updated Login Attempts report.

To build this report on page 101, the Login page, create a report with the following query:

```
SELECT user_name,
       authentication_method,
       access_date,
       authentication_result,
       custom_status_text
  FROM apex_workspace_access_log
 WHERE application_id = :app_id
ORDER BY access_date DESC
```

The Login page now looks like Figure 13-15. The report allows you to quickly see the authentication results when testing the login process.

Figure 13-15. *Login page with associated Login Attempts report*

Once you're content that your custom authentication scheme is working, you would normally delete the extra report region, because it's only there for debugging purposes. Deleting the region is counterproductive, however, because you may need it in the future when you modify the authentication scheme.

Creating a Build Option

Instead of removing the region, you should tag it as a Development Only object so it's available only while you're developing your application rather than when you are running in production. Follow these steps to create a build option to support this requirement:

1. In the **Application Builder** for the Help Desk application, go to **Shared Components ➤ Build Options (under Security)** and click the **Create** button.

2. Fill in each section as shown in Figure 13-16 and click the **Create Build Option** button.

Figure 13-16. *Creating a build option*

3. You should now see a **Development Only** build option, as shown in Figure 13-17.
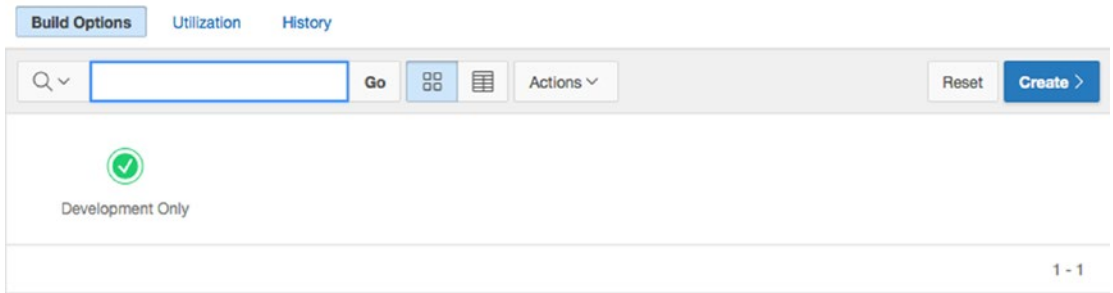
Figure 13-17. *Development Only build option*

Configuring Build Options

Before you continue with this example, it's important to review some of the options seen in Figure 13-16. The status values Include and Exclude can be misleading. Build options don't affect what is included in the application, just what is executed or displayed at runtime. A better description of the status options would be *enable/disable* or *on/off*.

The Default on Export option sets the default configuration of the build option when the application is exported and then imported. For this example, because you're using the build option to handle development-only features, it makes sense to always exclude the build option and require developers to explicitly include it.

In some cases there's no clear default option, so the person installing the application must choose the appropriate build-option status. You can configure a required choice as part of the application installation script.

Prompting for Build Option Status

To configure the application to prompt for a build option status during installation, go to Supporting Objects ➤ Build Options in the Application Builder. Select Development Only (Include) under Prompt for Build Options so as to prompt for the build option as part of the installation (see Figure 13-18), then click the Apply Changes button.
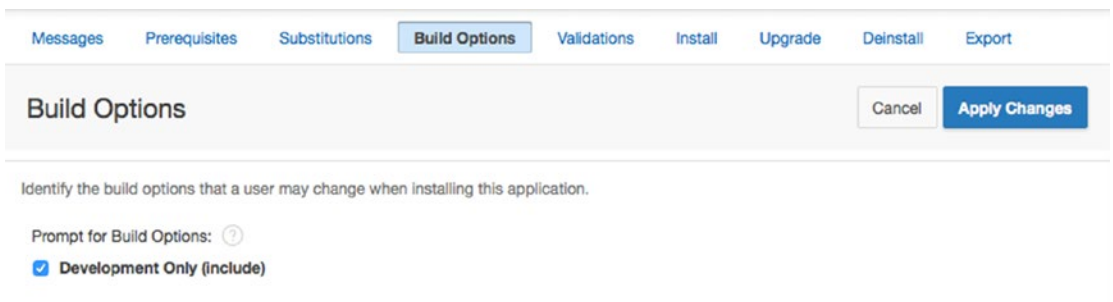
Figure 13-18. *Build option installation configuration*

377

When installing the application, the user will have the option to include the build option as shown in Figure 13-19. Again, *include* is a misleading term, because the build option will be included but will be disabled unless you specifically choose to include it during installation of the application.
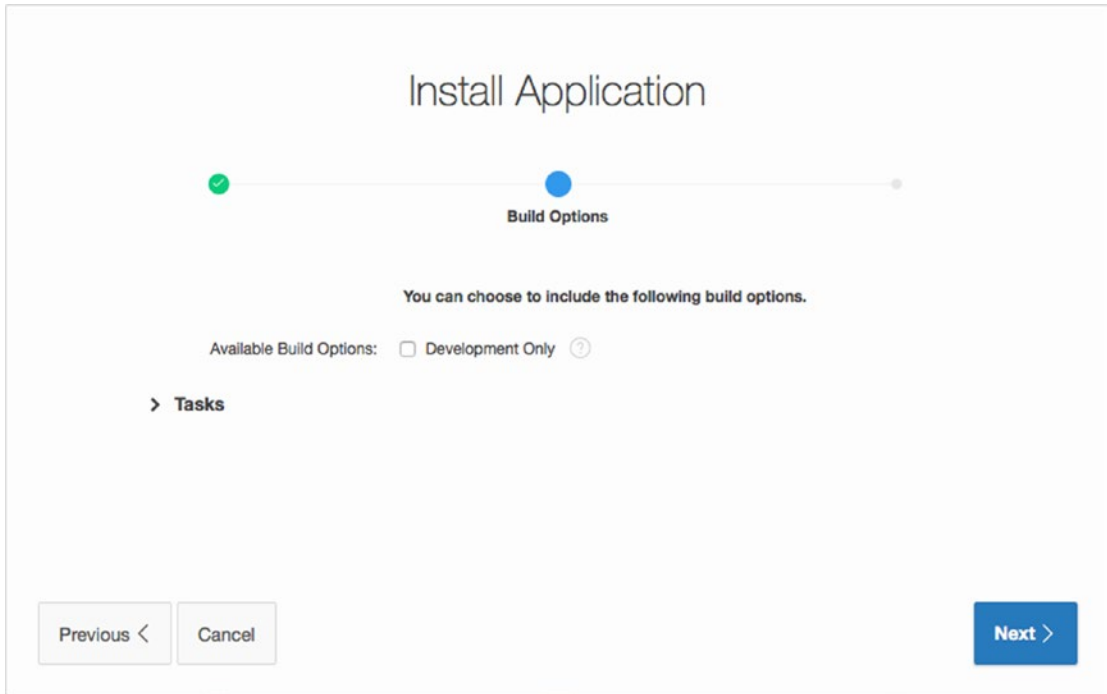
Figure 13-19. *Build option prompt*

Applying Build Options

Now that you've created and configured the Development Only build option, you need to apply it to the region in question—the Login Attempts region on page 101:

1. Edit the **Login Attempts** region.

2. In the Property Editor, scroll down to the **Configuration** section.

3. Select **Development Only** in the **Build Option** list (see Figure 13-20) and click the **Apply Changes** button.
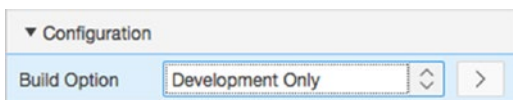
Figure 13-20. *Applying a build option*

> ■ **Note**   When you're applying build options, you can choose the opposite result by selecting the {Not …}
> option. This helps avoid having to create two build options that are always the reciprocal of each other.

The Access Log region is now run only when the Development Only build option status is set to
`Include`. You can apply build options to other APEX objects using the same process.

Reporting on Build Option Utilization

Build options can be applied to most APEX objects, including pages, regions, page items, tabs, and so on.
It can become difficult to keep track of which objects use, which build options in applications. The build
option Utilization report enables you to easily view which objects are using a particular build option. This
report can be very helpful when you're trying to get an overview of the impact that a build option has on the
application.

To view this Utilization report, go to Shared Components ➤ Build Options. Click the Utilization tab and
select the Development Only build option, as shown in Figure 13-21. The link in the description column
brings you to the specific object that is using the build option.

Figure 13-21. Build option Utilization report

Page-Specific Utilities

Page-specific utilities allow developers to perform bulk operations on APEX objects in an application. To
access page-specific utilities, go to Utilities from the Application Edit page. The Page Specific Utilities region,
located at the right, contains all the page-specific utilities available (see Figure 13-22).

Figure 13-22. *Page-specific utilities*

Each of the utilities provides tools associated with the type of object. This book doesn't cover each utility, but we encourage you to explore the available features.

APEX and Oracle SQL Developer

Oracle SQL Developer is a free database-development GUI. For more information about SQL Developer, go to http://www.oracle.com/technetwork/developer-tools/sql-developer.

Integration

APEX is integrated with SQL Developer. In SQL Developer, you can see all the APEX applications whose parsing schema is the same as your connecting schema (see Figure 13-23).
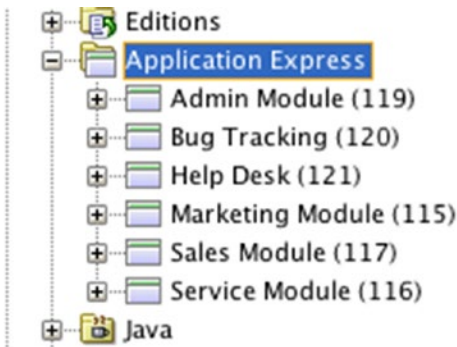
Figure 13-23. *APEX in SQL Developer*

In SQL Developer, you can view object information and perform basic application-level tasks such as importing and exporting an application, changing the application alias, and renaming an application.

Refactoring Support

APEX allows for anonymous blocks of PL/SQL. We strongly recommend that such blocks reference compiled code (packages, functions, and procedures). Storing code in packages helps separate the business logic from the display layer and may have some performance benefits.

In some cases, you may have written large blocks of code directly in the application. Eventually, you should move these large blocks of code into compiled PL/SQL code. SQL Developer provides a tool that automatically generates a PL/SQL package from the anonymous blocks of PL/SQL in APEX. You can then replace your large blocks of code with references to this package. To generate the package, right-click the application in SQL Developer and select Refactor (in Bulk), as shown in Figure 13-24.
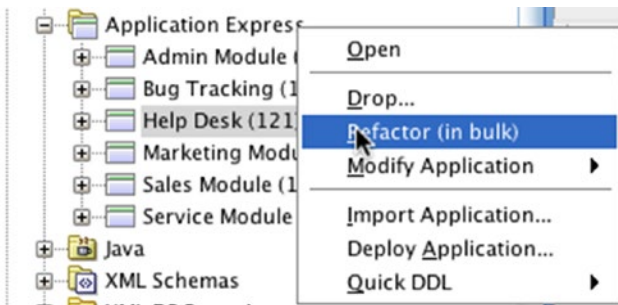
Figure 13-24. *Refactoring code*

SQL Developer opens a new worksheet with the necessary code to compile. The new worksheet also provides notes on what sections of your APEX applications to change and the code to replace them with. Similar to the APEX Advisor, these are recommendations; you should follow your organization's development standards, and so on.

Summary

Many tools in APEX make your life as a developer easier. Take some time to get to know the tools and utilities presented here, and you can undoubtedly speed up your ability to get things done. And although any PL/SQL GUI can help you edit and manage database objects, it should be clear by now that Oracle's SQL Developer has special hooks to make managing, developing, and debugging with APEX far more straightforward.

CHAPTER 14

■ ■ ■

Managing Workspaces

Once you start developing APEX applications and working in a team environment, you will need to spend some time managing workspaces. This chapter will cover the various tools and resources available with which to manage a workspace.

■ **Note** Administering a workspace covers many different areas. This chapter doesn't discuss all areas in detail; some have been mentioned in other chapters, and others aren't included because of space constraints. This chapter assumes that the user logging in is a workspace administrator.

Learning About Your Environment

When you log in to APEX as a workspace administrator and navigate to the Administration home page using the Administration drop-down menu in the upper right-hand corner of the page, you see icons for each of the major sections (see Figure 14-1). This chapter will cover each of these sections and their associated subsections.
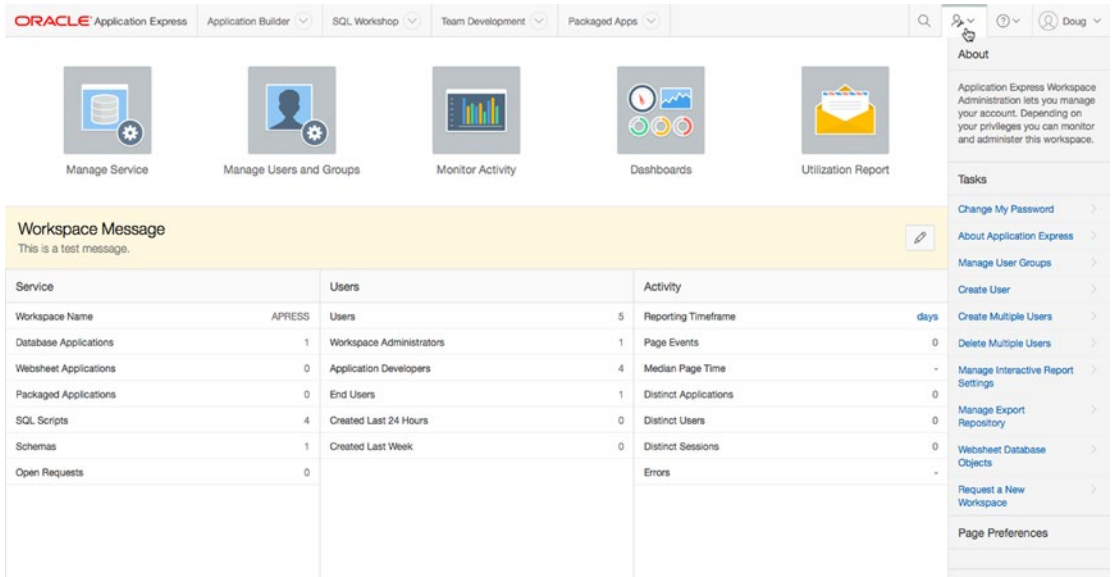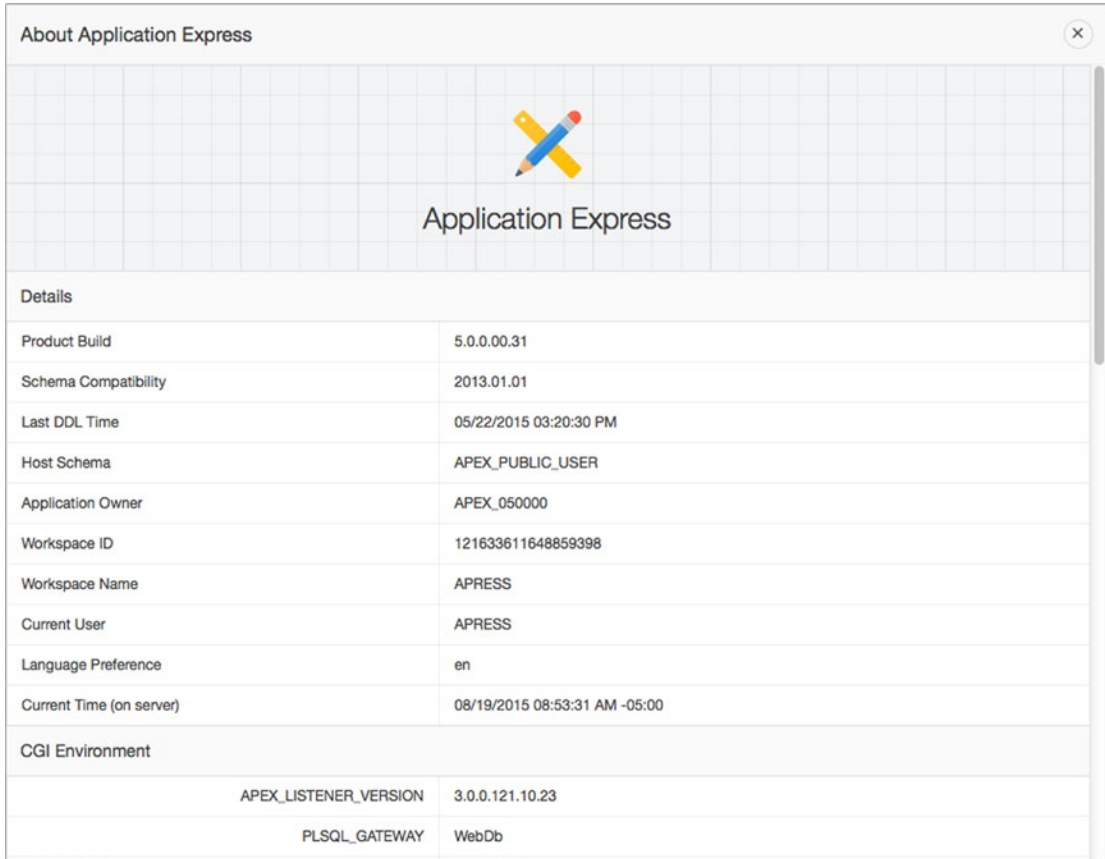
Figure 14-1. Administration home page

Viewing Instance Information

On the right-hand side of the Administration home page, in the Tasks region, the second link is About Application Express. Clicking this link brings you to a page that shows information about your current APEX instance (see Figure 14-2).

Figure 14-2. *About Application Express page*

The information on this page can be very useful when you're developing and debugging APEX applications. In fact, the About page is an excellent location from which to quickly get an overview of your particular APEX instance. Because APEX is stored in the database, all the values in the figure can also be obtained from a query. For example, you can find the database version from the following query:

```
SELECT *
FROM v$version
```

The list of CGI variables in Figure 14-2 is only available to workspace administrators. If developers need to see a complete list of CGI variables and their values, they can create a PL/SQL region and use OWA_UTIL.PRINT_CGI_ENV as the region source.

■ **Note**   You can only obtain CGI variables from a web interface because they're web-based variables. If developers try to query them in SQL*Plus or SQL Developer, they won't get any results. The following article contains more information on CGI variables: http://en.wikipedia.org/wiki/Common_Gateway_Interface.

Checking the APEX Version

The About Application Express page displays the current version of APEX. You can also obtain the version information using the following query:

```
SELECT *
FROM apex_release
```

Obtaining the APEX version from a query can be important for DBAs when upgrading an application. Also, developers sometimes write code that is dependent on the current version of APEX.

Managing the Service

Click the Manage Service icon to change preferences and other settings affecting the operation of your APEX service. You can find the Manage Service icon on the Administration home page shown in Figure 14-1.

Figure 14-3 shows the main Manage Service page.

Figure 14-3. *Manage Service page*

Workspace Preferences

You can enable and disable the different modules available to APEX developers. To configure each of the modules, click the Set Workspace Preferences menu item (see Figure 14-3). The Set Preferences page is shown in Figure 14-4.
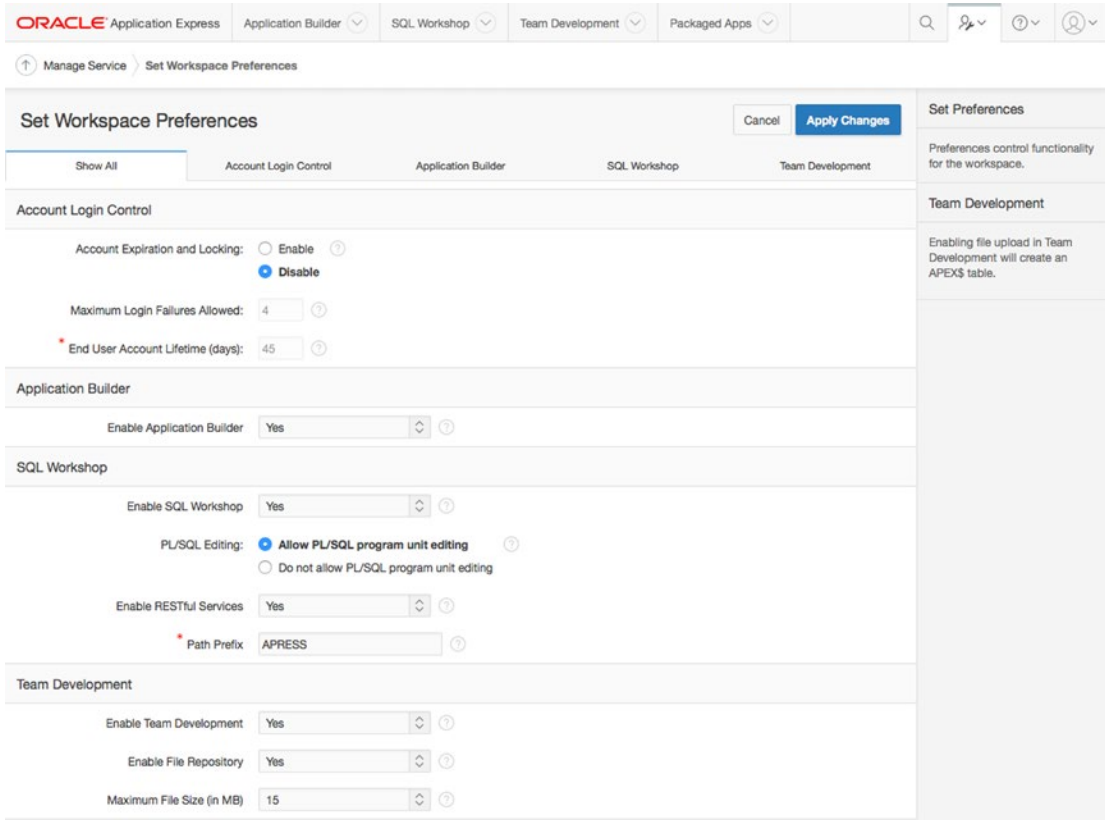
Figure 14-4. *Set Preferences page*

From the Set Preferences page, you can enable or disable the following APEX modules:

- Application Builder
- SQL Workshop
- Team Development

If you disable any of these modules, they're disabled for all users regardless of their privileges. In production instances, you may want to restrict access to the SQL Workshop as part of your corporate policy, because the SQL Workshop has access to all the objects in the schema.

The Account Login Control section manages APEX workspace users. Settings in that section don't affect your APEX applications unless you're using the default APEX authentication scheme that references workspace users.

Messages

Messages allow workspace administrators to create messages that will be visible to APEX developers. Messages are displayed in key areas throughout the APEX development environment. These messages aren't displayed in your APEX applications.

To create a message, click the Edit Message menu item (see Figure 14-3). Enter a message and click the Apply Changes button, as shown in Figure 14-5. The message appears in the announcement section, as shown in Figure 14-6.
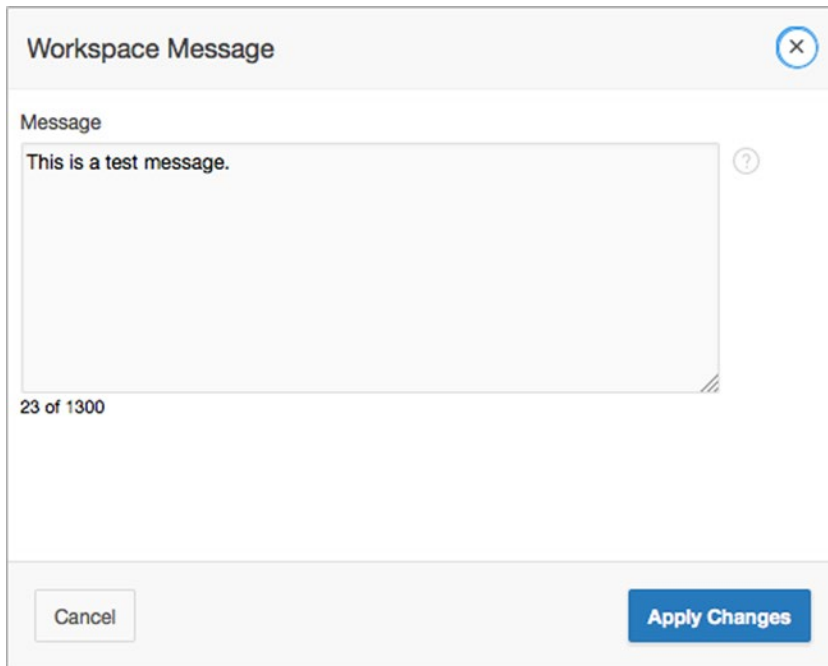
Figure 14-5. *Editing a workspace announcement*
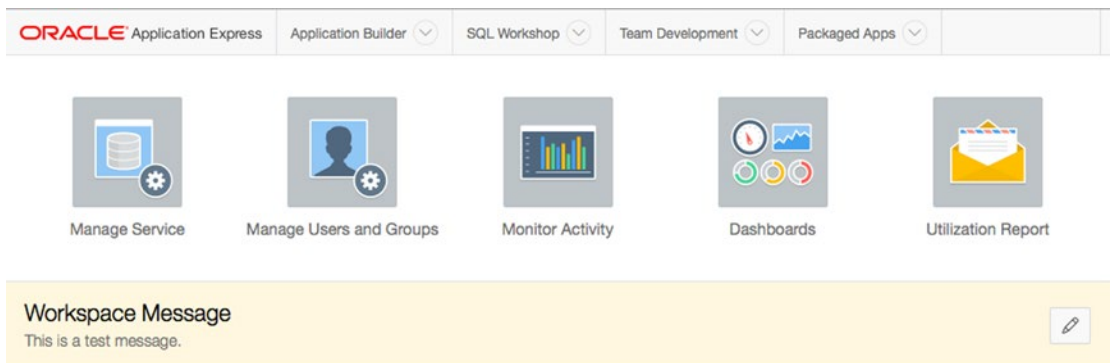
Figure 14-6. *A workspace announcement*

Managing Meta Data

On the right side of the Manage Service page is a region called Manage Meta Data (see Figure 14-7). The following subsections will cover what you can do using the menu choices in this region.
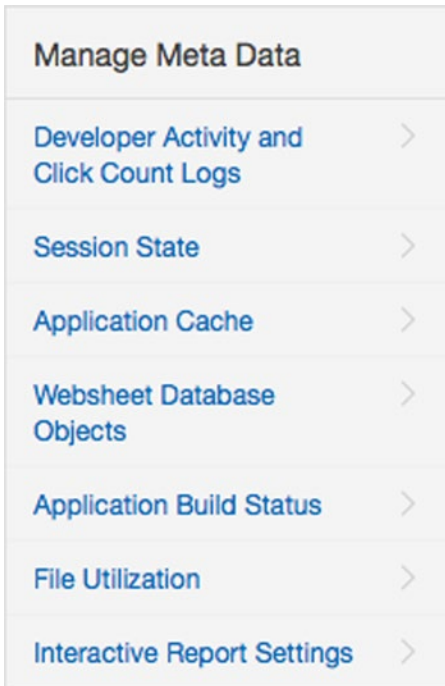
Figure 14-7. *Manage Meta Data menu*

Developer Activity and Click Count Logs

The Developer Activity and Click Count Logs menu option allows the purge of two types of logs. One logs changes made by developers in the workspace. The other tracks user clicks on links to pages outside your APEX application.

Developer Activity Logs

The Application Builder logs all changes made by developers. The logs are referenced in various locations throughout the Application Builder and in the Administration section. You can view the logs from the Monitor Activity section, which you can access by clicking the Monitor Activity icon shown in Figure 14-1. You can also obtain the developer activity logs from the following query:

```
SELECT *
FROM apex_developer_activity_log
```

Like the logs mentioned in the previous chapter, the developer activity logs retain up to one month of data. You can purge developer activity logs by clicking the Purge Developer Log button shown in Figure 14-8.
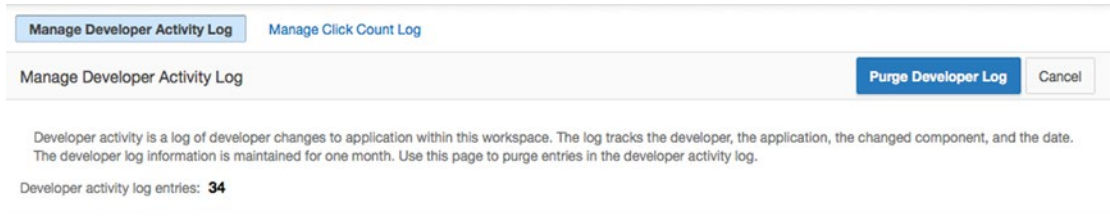
Figure 14-8. *Purge Developer Log button*

Click Count Logs

When creating a list (Application Builder ➤ Shared Components ➤ Lists), you can choose to track clicks on external links. These clicks are logged in the APEX_WORKSPACE_CLICKS log. You can also view the click count log in the Monitor Activity section (see the icon for Monitor Activity in Figure 14-1).

Similar to the developer activity logs, you can purge the click logs by selecting Manage Click Count Log and clicking the Purge Click Log button shown in Figure 14-9.

Figure 14-9. *Purge Click Log button*

Session State

Choosing Session State from the menu in Figure 14-5 brings you to a page that lets you manage session state and preferences. Preferences are slightly different than session state, as they're linked to the user and not the current session. This means that changes to a preference affect all current and future sessions for a user.

Manage Session State

You can view all the session values, clear these values, and end sessions from the various reports available in the Manage Session State region. A session is automatically created each time a user logs in to APEX. Because the APEX Builder is an APEX application, sessions are created for both developers and end users.

Clearing session state or terminating a session may be useful in development to simulate different situations. If you're doing this in a production environment with live users, you should be extremely cautious.

Manage Preferences

The Manage Preferences region contains links to view and manage user preferences. Preferences are used to permanently store values for each user. Because they're linked to a user, they're session independent. One of the most common uses for user preferences is to store the sort order for standard reports that APEX automatically manages.

Similar to session state, you shouldn't purge preferences in a production environment unless you're certain about what you're doing.

Application Cache

The Application Cache section provides the tools to purge page and region caches based on different criteria. In APEX you can cache pages and regions either for all users or by each individual user. Caching can help improve performance, because APEX doesn't need to regenerate the HTML code for a given region.

Websheet Database Objects

The Websheet Database Objects section allows you to create or delete the tables required for websheets and validate their status. Unlike an APEX application, websheet data is stored in tables that reside in each schema instead of in tables owned by the APEX_050000 schema. The tables start with the prefix APEX$_WS, and usually there are about ten of them.

When you first create a workspace, the websheet objects aren't created. To create them, go to the Websheet Database Objects section and click the Create Websheet Database Objects link shown in Figure 14-10. Follow the wizard to create the websheet objects. After you've created the websheet objects, you can create a websheet application.
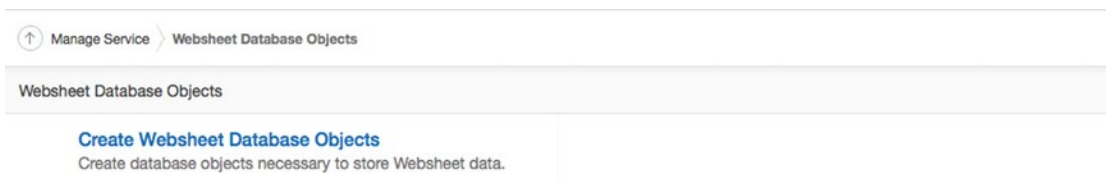
Figure 14-10. *Create Websheet Database Objects link*

Once the objects are created, the Websheet Database Objects page looks like that in Figure 14-11. From this screen, you can remove the database objects that relate to websheets or ensure that these objects are valid.
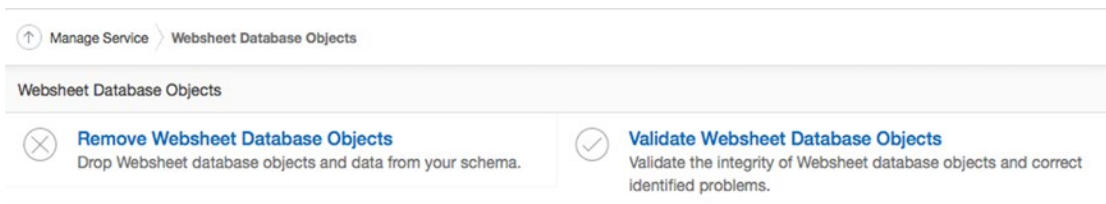
Figure 14-11. *Websheet Database Objects page*

Application Build Status
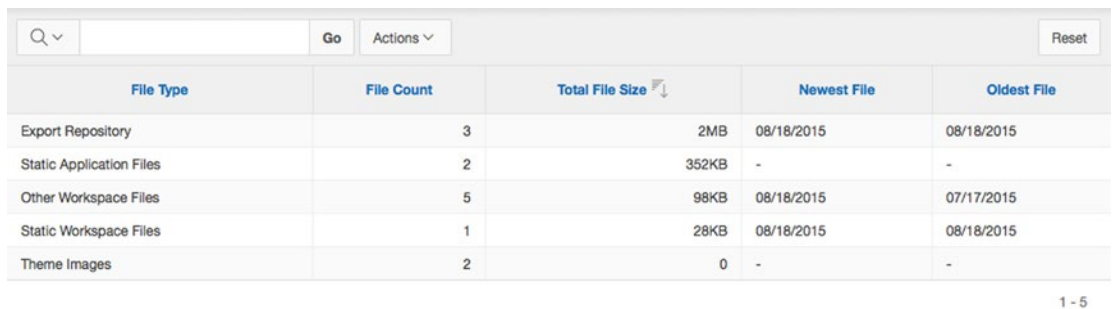
From the Application Build Status section, you can quickly manage both the Application Status and the Build Status for all the applications in the workspace. The Application Status controls the availability of the application. To get a full explanation of each application status, reference the APEX documentation.

The Build Status determines whether an application can be modified by developers or run only. In production environments, you may want to set the Build Status to Run Application Only to prevent any changes.

■ **Note** The Application and Build statuses can also be set from the Application Properties page for each application.

File Utilization

The File Utilization page provides an overview of all the types of files and their total size in the workspace, as shown in Figure 14-12. There are various locations where files can be stored as part of an APEX application. Over time, these files can take up unnecessary space.

File Type	File Count	Total File Size	Newest File	Oldest File
Export Repository	3	2MB	08/18/2015	08/18/2015
Static Application Files	2	352KB	-	-
Other Workspace Files	5	98KB	08/18/2015	07/17/2015
Static Workspace Files	1	28KB	08/18/2015	08/18/2015
Theme Images	2	0	-	-

1 - 5

Figure 14-12. *File-utilization information*

The export repository tends to consume the most space out of all the files listed on the File Utilization page. When you import an application into your workspace, the original file is stored in the export repository. The name *export repository* can be misleading, because the repository contains the original application files that are imported into the workspace. Once an application has been successfully imported and installed, you don't need to retain the file in the repository, so it can be removed.

The following steps explain how to clean up the export repository:

1. From the **Administration** home page, click the **Manage Export Repository** link located in the **Tasks** region at the right.

2. Select the files you no longer need and click the **Delete Checked** button, as shown in Figure 14-13.
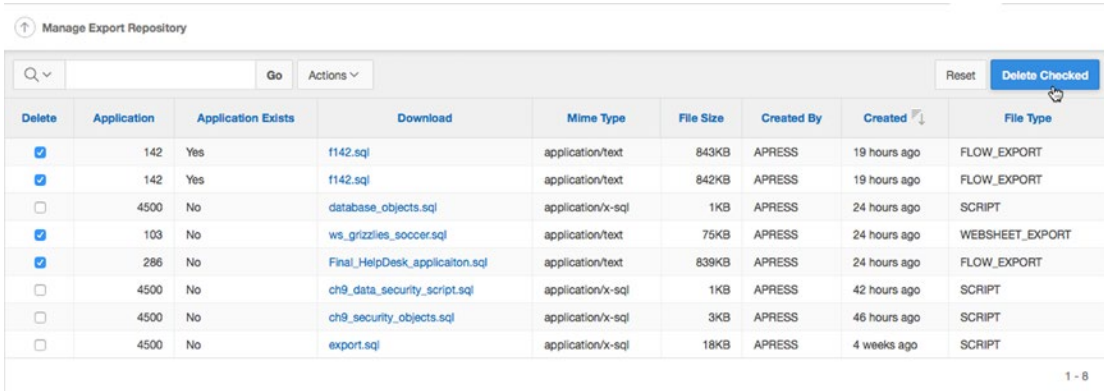
Figure 14-13. *Delete Checked button*

Interactive Report Settings

The Interactive Reports Settings page allows workspace administrators to manage saved reports and report subscriptions. Only workspace administrators can delete saved reports and report subscriptions. The only way to modify a report configuration is to log in as the user.

Saved Reports

The Saved Reports section allows you to delete certain saved interactive reports. Interactive reports have four different types of saved reports (see Figure 14-14). Primary Default is the default report setting as created by the developer and is accessible to all users. Alternative Default reports are also saved by the developer and are accessible to all users. This gives the developer the ability to pre-create several versions of the same report.
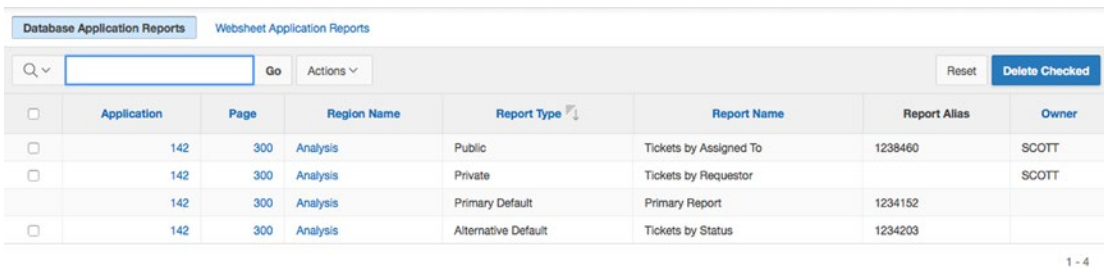
Figure 14-14. *Manage Saved Interactive Reports page*

Private interactive reports are custom interactive reports that are only visible to the users who created them. Public interactive reports are reports saved by a privileged end user that have been created as public so any other user can see them.

On the Saved Reports page, you can delete any report except the Primary Default report by selecting the report(s) and clicking the Delete Checked button. Primary default reports can't be deleted from this screen. They can only be altered by directly editing the report region.

Subscriptions

Similar to the Saved Reports section, you can delete interactive report subscriptions from the Subscriptions section. Subscriptions allow users to receive emailed reports on a schedule that they define. Subscriptions were a new feature in APEX 4.0 and must be explicitly enabled by developers.

On the Manage Subscriptions page, workspace administrators can either delete specific subscriptions or all subscriptions, as shown in Figure 14-15. Just like with saved reports, you can't modify any of the subscription attributes.

	Application	Page	Region	Subscribed By	Frequency	Email To	Created	Created By	Status
☐	142	300	Analysis	SCOTT	Daily	scott@sumnertech.com	70 seconds ago	SCOTT	Sent
☐	142	300	Analysis	TIM	Daily	tim@sumnertech.com	29 seconds ago	TIM	Submitted

1 - 2

Figure 14-15. *Manage Subscriptions page*

Managing Users and Groups

This section will cover most of the features available when you click the Manage Users and Groups icon located on the Administration home page shown in Figure 14-1. All references assume that you start on the Manage Users and Groups page shown in Figure 14-16.

User	Email	Account Type	Locked	Builder Last Login	Created
APRESS	admin@mycompany.com	Workspace Administrator	No	24 minutes ago	3 months ago
CANDY	candy@mycompany.com	Developer	No	-	6 weeks ago
DOUG	doug@mycompany.com	Developer	No	6 weeks ago	6 weeks ago
JOEY	joey@mycompany.com	Developer	No	-	6 weeks ago
MILO	milo@mycompany.com	End User	No	-	6 weeks ago

1 - 5

Figure 14-16. *Manage Users and Groups page*

The main page contains a list of all the end users, developers, and workspace administrators. You can modify a user by clicking their name. End users don't have access to the development environment and have access to applications only if the authentication scheme is the default Application Express.

Creating One User

To create a single user, click the Create User button shown in Figure 14-16. If you're only creating developers, you don't need to worry about entering some of the non-required fields, such as first and last name, because you usually know the user based on the username. You can also grant access to specific modules. For example, if you create an account for a project manager, you may only want to give them access to the Team Development module, as shown in Figure 14-17.

Figure 14-17. *Account privileges*

Creating Multiple Users

Creating multiple users one at a time using the Create User page can become slow and frustrating. APEX provides an interface to create multiple users. Start by clicking the Create Multiple Users button, as shown in Figure 14-16.

On the Create Multiple Users page, you need to provide an email address for each user. The email addresses can be delimited by commas, semicolons, or new lines. APEX generates the usernames from the email addresses that you enter. For example, the email address doug@mycompany.com yields as a username either doug@mycompany.com or just plain doug. You can specify whether you wish each email address to translate directly into a username, or whether you want the usernames to exclude the @domain suffixes. Figure 14-18 shows the radio button set for the latter option.

Create Multiple Users

To create multiple users at once, enter or copy and paste email addresses separated by commas, semicolons, or new lines.

Note that the password you specify will be assigned to each user and users will need to change their passwords upon login.

* List of Email Addresses

```
doug@mycompany.com
joey@mycompany.com
milo@mycompany.com
candy@mycompany.com
```

Usernames ○ Set username to full email address

● Exclude @ domain as part of the username

Account Privileges

Default Schema   APRESS

Accessible Schemas (null for all)

Users are workspace administrators   ○ Yes ● No

Users are developers   ● Yes ○ No

Application Builder Access   Yes

SQL Workshop Access   No

Team Development Access   No

Password

* Password   ••••   Passwords are case sensitive

* Confirm Password   ••••

Cancel

Next >

Figure 14-18. *Create Multiple Users page*

If your usernames don't correspond to your corporate email addresses, you can enter email addresses with invalid domains and select the option to omit the suffixes. For example, enter the email address jonathan@example.com when you just want a user named jonathan. Then select the option to exclude the @ domain portion of the email address.

The password that you enter is the same for all the users. When they log in to the workspace, they will be required to change it.

After you enter all the information on the Create Multiple Users page, click the Next button to go to the confirmation page shown in Figure 14-19. If everything is correct, click the Create Valid Users button.

Figure 14-19. *Create Multiple Users confirmation*

Organizing Users into Groups

You can associate an APEX user with multiple groups. These groups can be used in authorization schemes to grant or deny access to various parts of an application. Because groups are linked with APEX users, you can only use groups when your authentication scheme is set to the default Application Express scheme.

It's important to note that groups are associated with a single workspace. If you have a traditional setup with development, test, and production environments, you will need to create the same groups in each of the workspaces.

Creating a Group

To create a user group, click the Manage User Groups link in the Tasks region (shown in Figure 14-20), and then click the Create User Group button. Enter a unique group name and description, as shown in Figure 14-24. Groups can be nested so that they contain other groups. When a group contains other groups, they become a parent group and therefore "inherit" membership of the other groups.

Tasks

Change My Password　>

About Application Express　>

Monitor Activity　>

Manage Service　>

Manage User Groups　>

Create User　>

Create Multiple Users　>

Delete Multiple Users　>

Manage Interactive
Report Settings　>

Manage Export
Repository　>

Websheet Database
Objects　>

Request a New
Workspace　>
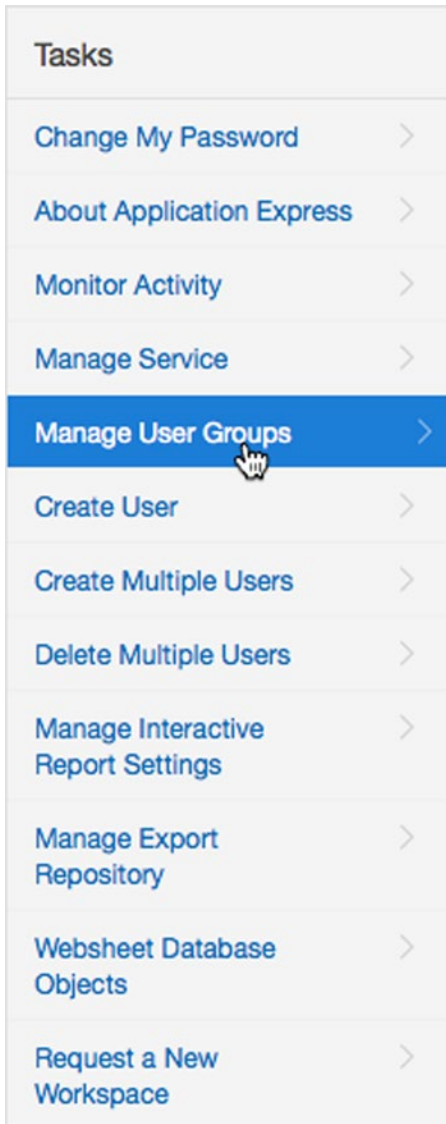
Figure 14-20. Manage User Groups option from the Tasks menu
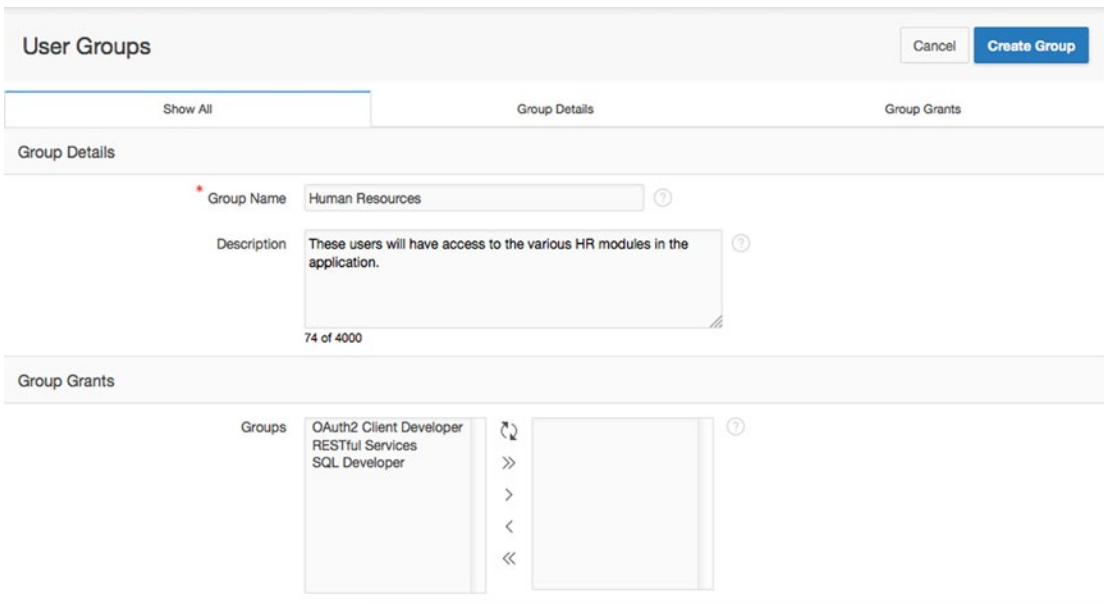
Figure 14-21. *Creating a user group*

Assigning Users to a Group

Assigning users to a group isn't very intuitive. Clicking Group Assignments in the Manage User Groups region only gives you a report of all the users and their groups. To assign a user to a group, you need to edit the user (Users ➤ Edit User) and scroll down to the User Groups region shown in Figure 14-22.
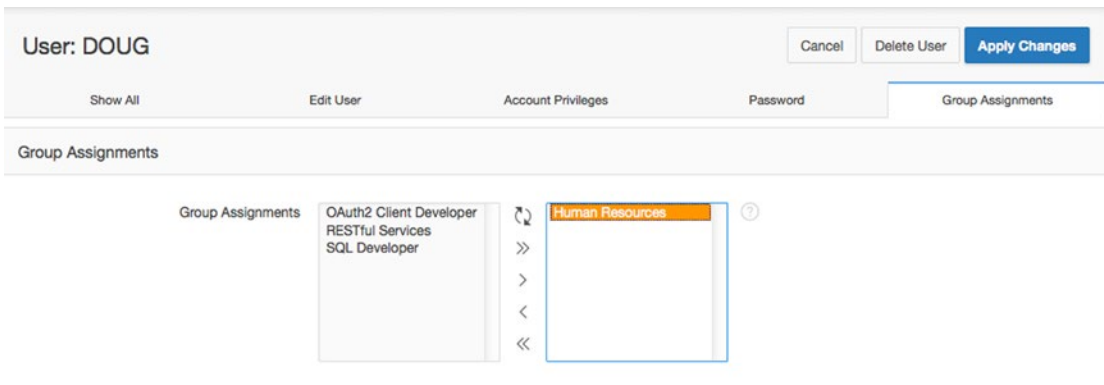
Figure 14-22. *Assigning users to a group*

Viewing Usage Reports and Dashboards

The Monitor Activity and Dashboard sections from Figure 14-1 provide many detailed and summary reports about your APEX workspace. The reports contain information about both developer activity and end-user activity. You're encouraged to explore each of these sections to see all the available reports.

Summary

APEX provides many tools to help you manage your workspace, plus reports that provide statistics about your workspace. Some tools can affect end users and should be used cautiously. However, don't let the need for caution deter you from using tools that can make your job easier. Learn the tools well. Be confident in your knowledge. Take a moment to think before you act. These are the keys to success.

■ ■ ■

Team Development

Team Development is an excellent tool for managing APEX software development. It is embedded within the APEX development environment, allowing you to manage your APEX projects with tools that you and your team use every day, like interactive reports. This eliminates the need for an external project-management tool. Team Development's simple, extensible, and flexible architecture is ideal for managing projects of any size. Its simplicity lends itself to small projects where you want to minimize your project-management overhead. Its extensibility scales well, allowing you to manage large projects by using its rich set of attributes to construct a sophisticated work breakdown structure (WBS). Its flexibility lets you adapt it to your organization's software-development culture.

APEX and Team Development are both well suited to working with agile software-development methodologies. APEX's Rapid Application Development (RAD) architecture allows the team to rapidly deliver regularly scheduled releases to business testers and end users. Team Development's feedback mechanism efficiently channels end-user issues back to the development team so that appropriate changes can be included in the following releases and easily documented. Team Development makes APEX, which is already an efficient development platform, even more efficient.

The primary purpose of this chapter is to highlight how Team Development works. A secondary purpose is to illustrate how agile software-development practices can be used with Team Development to deliver quality software while meeting your cost and schedule constraints. By the end of the chapter, you should have a good understanding of what is in Team Development and some ideas about how to integrate it into your team's development culture.

Team Development was added to APEX in version 4.0. The APEX team has used a version of Team Development while developing APEX, even before it was released as an APEX module. This design philosophy of using the tool to build the tool is a key reason why Team Development and APEX are practical and easy-to-use tools.

Team Development Overview

Team Development consists of milestones, features, to-dos, bugs, and feedback. These modules are complemented by a small number of utilities called Team Actions that are used to manage the Team Development environment. The entity-relationship diagram (ERD) illustrates the relationships between the main entities (see Figure 15-1).
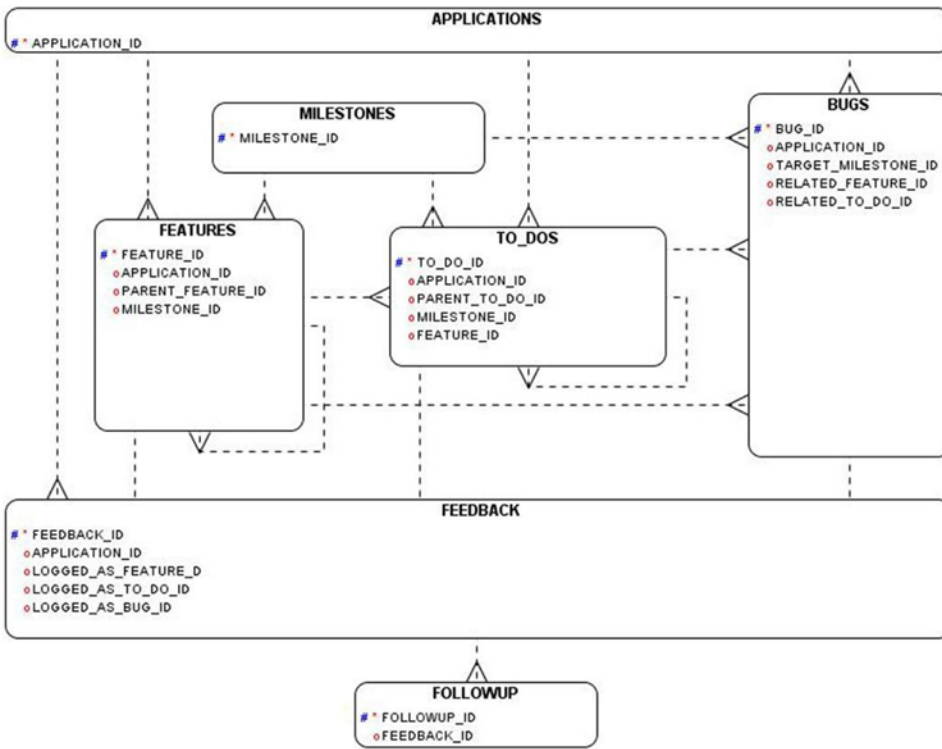
Figure 15-1. *A simplified entity relationship diagram of Team Development*

The Team Development entities are contained within a single workspace. When you're managing several applications simultaneously within a workspace, you and your teams can easily filter Team Development's data using the tool's interactive reports. That way, each stakeholder sees only the work that is of immediate interest for the tasks at hand.

Milestones are the high-level scheduling components of Team Development. They can be used to track major and minor releases of an application. Don't confuse milestones with the due dates that are associated with features and to-dos. For example, a milestone could be the scheduled date for a product release, but a feature that is included in the release might be due well in advance of the milestone date. If you use an agile software-development strategy, milestones are well suited to define the time boxes or sprints.

Features describe the big picture. At a high level, they describe and track progress for the major pieces of functionality that your client has requested. Attributes like owner, start date, due date, summary description, priority, status, and so on are easily tracked. In addition, you can optionally track the status of individual components, such as the user interface, testing, documentation, globalization, security, and accessibility. The self-join that is associated with features allows you to further subdivide a feature into child features as development progresses and the design becomes more refined. The high-level nature of features makes them an ideal source for management status reports.

To-dos are used to assign detailed work to individuals or teams. Due dates, time estimates, and so on are tracked. Like features, to-dos have a self-join capability that allows you to subdivide them as more information becomes known during the development process. This is handy when a single to-do requires effort from more than one team member.

When a defect is found, it's reported as a bug. *Bugs* have their own lifecycle that includes a rich set of attributes, such as description, resolution, release, assignee, context, and date information. When you find a bug that turns out to be a design flaw, Team Development allows you to link the bug to multiple features,

milestones, and to-dos, because the bug fix might require an extended timeframe that spans a number of product releases.

Feedback is one of the primary sweet spots of Team Development. The feedback mechanism can be installed in an application in a matter of minutes and gives the entire team an efficient and elegant communication channel for comments, suggestions, issues, defect reports, and responses. Once installed in an application, the feedback mechanism is promoted with the application from the development environment to the test environment and on to the production environment. Feedback data and the related responses are copied back and forth between the various environments via the APEX import/export facility. This keeps the developers, testers, and end users in close touch with each other. Another key feature of the feedback mechanism is that it automatically records all the critical "under the hood" data that end users know nothing about. The application context, environment variables, and, most important, session state are all captured. This data is invaluable when you're diagnosing an issue.

At first glance, the Team Development entities seem to contain a large number of attributes. So, do you have to enter all these attributes in order to use Team Development? No. To get the most out of Team Development, you and your team must plan how you're going to use it, with a view to effectively managing the software-development process with minimum effort. You can pick a small subset of Team Development's attributes that fit your software-development culture and key into only that subset of the data. Interactive reports can easily be customized to display only the data that is important to you.

Team Development Interface

The Team Development interface is consistent with the overall APEX development interface. It makes extensive use of dashboards so as to quickly give you an idea of how your teams are progressing. Drilling into the details to view and update the data is fairly straightforward and intuitive. APEX developers will have no trouble navigating within the tool or customizing interactive reports. Managers, testers, and business analysts will also be able to use Team Development after a short training session.

APEX Home Page

The APEX home page (see Figure 15-2) clearly indicates how important Team Development is to the APEX team. The APEX team could have easily put Team Development under a minor link somewhere on the home page. Instead, Team Development has been promoted into a marquee module that is on an equal footing with the Application Builder and the SQL Workshop.
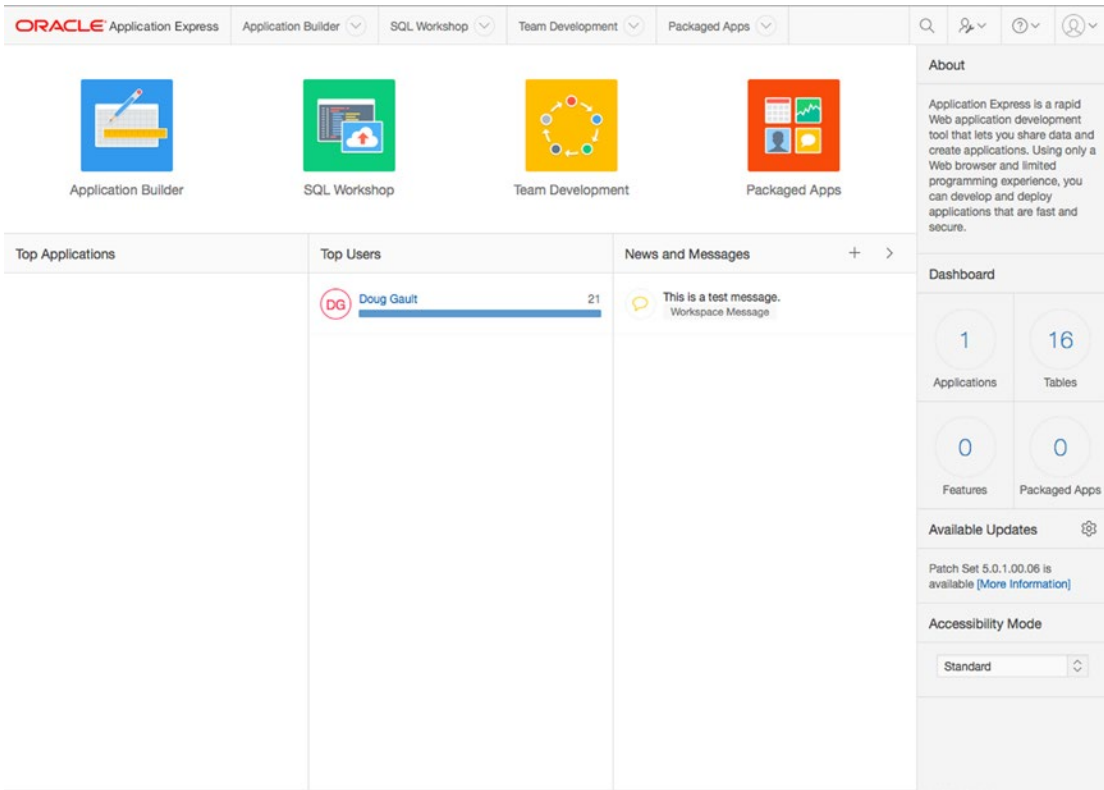
Figure 15-2. *APEX home page*

The APEX home page highlights the easy and convenient navigation. The News and Messages region, which will house workspace-level messages as well as news directly from Team Development, contains two links (Create News Items and Edit News Items) that enable quick changes to the region without having to drill down into subpages. In the Dashboard region to the right, you'll see that one of the metrics is Features. The region to the right-hand side often contains either further information or a list of tasks available to the developer. Look for similar links throughout the Team Development interface; they're very handy.

Team Development Home Page

The Team Development home page (see Figure 15-3) serves two purposes. First, it contains links to all the detailed entity regions; and second, it shows you a dashboard for each entity. You also see links to Utilities, which will be discussed near the end of this chapter in the section "Team Development Utilities."
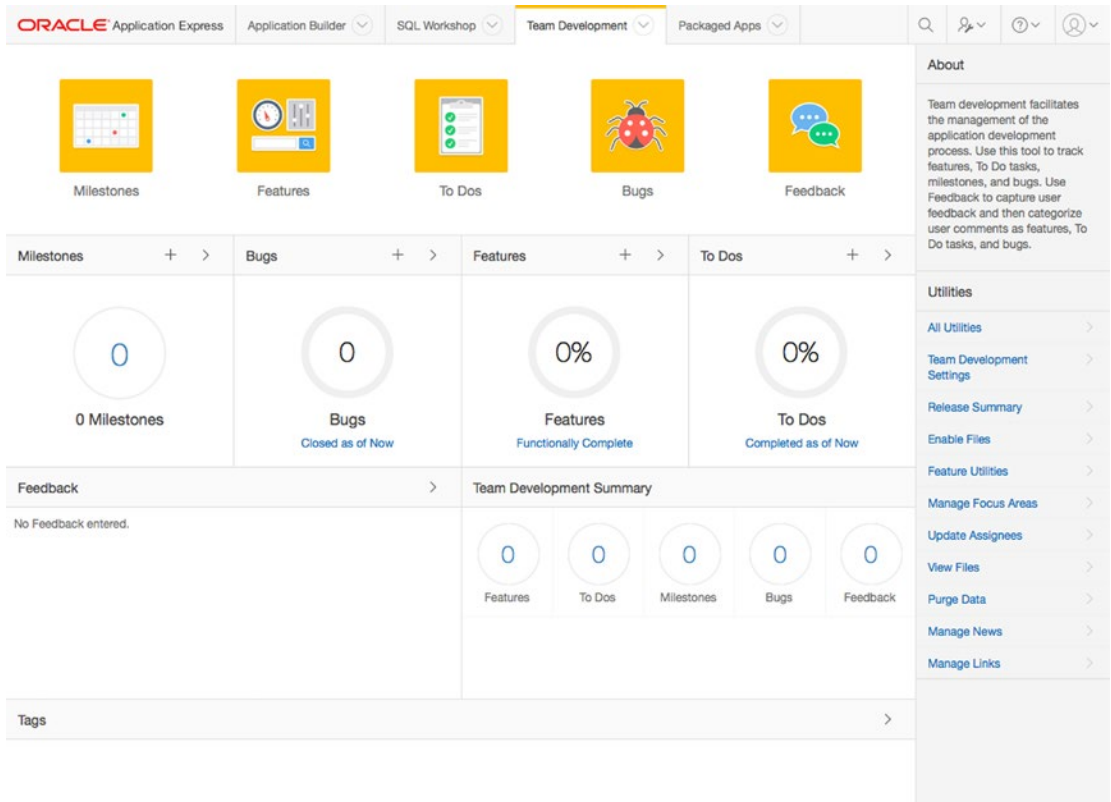
Figure 15-3. *Team Development home page*

The main Team Development dashboard page provides a high-level overview of activity in the module. From here, users can either add new items or navigate directly to the individual detailed dashboard for each area of Team Development.

The detailed dashboards can be filtered by a combination of assignee, release, and application, depending on the area. These filters make it easy for users to customize the page to their needs. For example, managers can ask, "How are we doing?" while a developer can ask, "What do I need to do for this release?" This high-level filter region is found at the top of dashboard pages associated with each Team Development entity. The filter fields are tailored to each entity. The dashboard regions link to their underlying entities. There are Add and Edit icons in each header, as well as a circle graph for each item type; when you click any of these items, you're taken the appropriate report, Create page, or Edit page for the related item.

Common Design Elements

When you drill into the individual Team Development entities, you will find several common design elements that help you navigate quickly and intuitively between entities. Figure 15-4 highlights these design elements. At the upper left is a set of tabs tailored to each entity. Some of the tabs—Dashboard, Report, and Calendar—are common to all entities. Others are unique to a specific entity.
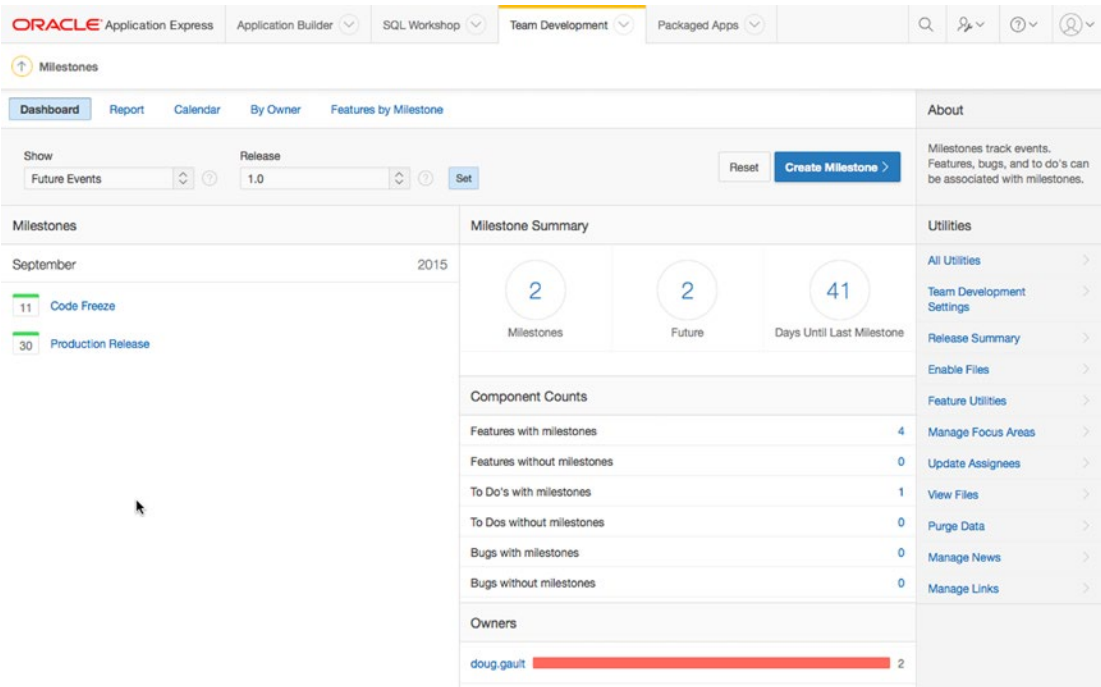
Figure 15-4. *GUI design elements that are common to all entities*

All of the dashboard pages contain a Filter region that is tailored to the entity. The filter's select list contains entries for only the entities that are tracked by Team Development. For example, if there is an application that doesn't have any features associated with it, that application doesn't appear in the select list.

Many of the individual dashboard regions contain links that take you to the entity's Report page and automatically set the filters on the Details page's interactive report so you see only the entity records that are related to the dashboard item you selected. This navigation strategy, once you get used to it, is extremely convenient.

Drilldown Functionality

The Calendar tabs (see Figure 15-5) display links to individual entity records based on the entity record's due date. Clicking the entry's link takes you directly to the Edit page for the selected entity record. Clicking the Cancel button on the Edit page returns you to the calendar.
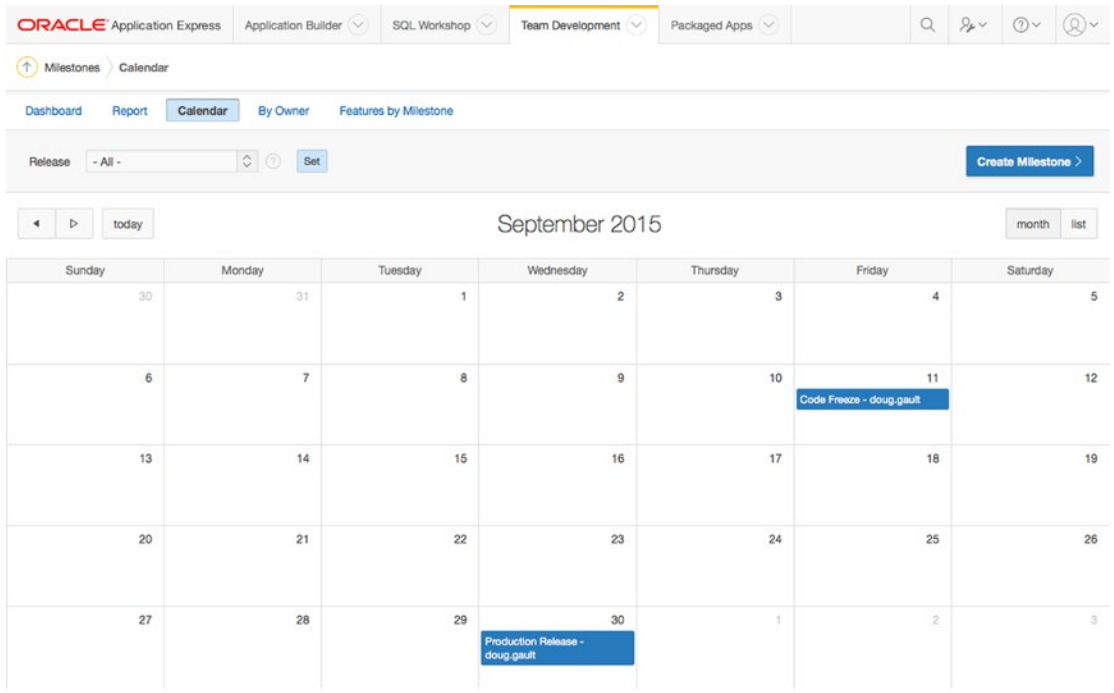
Figure 15-5. *Calendar with links to an Edit page*

Some of the entity tabs display their data graphically (see Figure 15-6). Clicking a graph automatically takes you to the entity's Details page and sets the interactive report to show only the data selected on the graph.
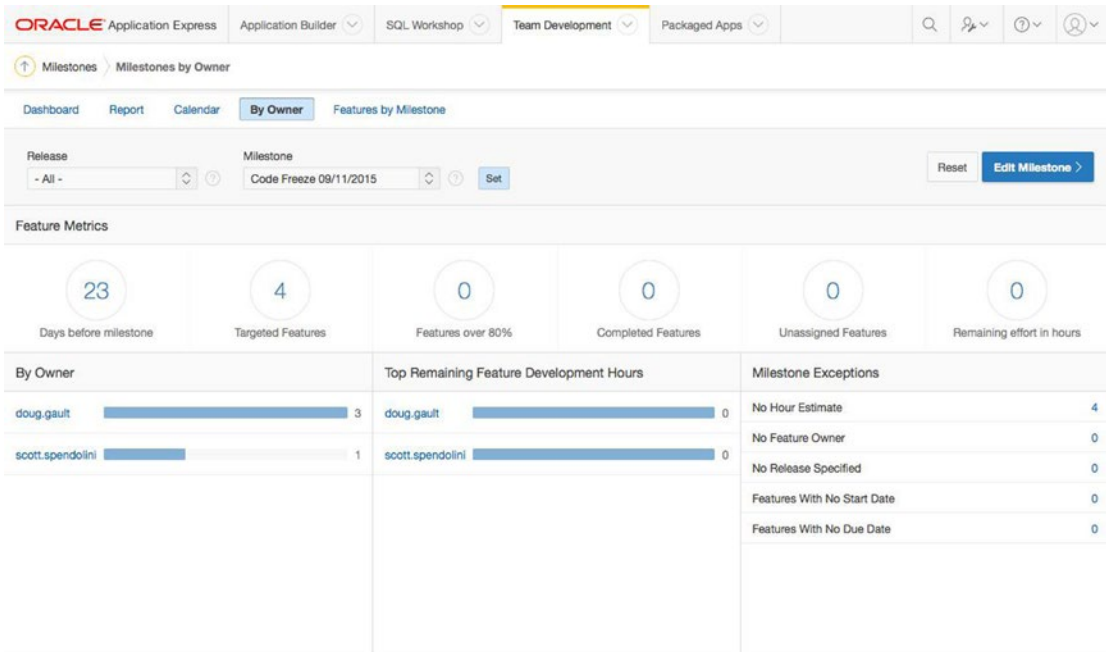
Figure 15-6. *Graphic data summary*

Tagging

A powerful organizational attribute called *tags* is associated with all entities. A tag is a free-form text input that enables you to group records by a keyword. This is handy, because some record groupings don't fit into the neat and tidy relational data model. For example, you might want to be able to find all features that contain a shuttle item across all applications. By adding the tag "shuttle" to these features, you can use the interactive report Search field to find all records that contain the tag of "shuttle." All dashboard pages contain a region that displays the tag strings defined in their respective entity. This lets you quickly find and correct typos so the free-form tags can be kept accurate.

GETTING PAST THE INITIAL DISCOMFORT

At first, Team Development's organization, layout, and navigation might appear a bit strange and non-intuitive. I certainly struggled with it at first. My initial perception was probably the result of my previous experience with project-management tools that use pages with a work-breakdown structure on the left and a Gantt chart on the right.

The initial discomfort with Team Development was similar to my first experiences with APEX. Like many experienced developers, I came from development environments in which screen widgets are listed on the left and dragged onto a screen, and in which the X-Y coordinates are set together with a widget's height and width, all at pixel-level precision.

When I first used APEX, it took a while to get used to how APEX built pages. After a few tries, I learned to love the product. I believe the same will be true for you with Team Development. After a bit of experience, you'll find that the interactive reports are, in fact, a practical alternative to Gantt charts, especially in an agile software-development environment in which the time boxes are short and the lists in the interactive reports are correspondingly small.

Milestones

Milestones are used to define and track event dates for both scheduled and one-time happenings. On the surface, this appears to be a simple concept; however, it can be a confusing area, because milestone functionality overlaps with releases and can be confused with the due dates associated with features, to-dos, and bugs. Fortunately, the confusion can be mitigated with a bit of planning and organization.

One strategy you can use to organize milestones and releases is to create one of each for the same release event. The milestone contains the metadata associated with the release: the date, a type, the owner, a description, and tags. The release is defined as a configurable LOV that is shared by all Team Development entities, but it contains no descriptive metadata. Also, releases are one of the handy high-level filters associated with features, to-dos, and bugs. The need for both metadata and the high-level filters is the reason a milestone and a release must be used to describe the same event. Happily, the overhead in doing this is low.

Another suggestion for workspaces that contain multiple applications is to prefix features, milestones, releases, to-dos, and feedback with the name or code of their associated application. In many cases, this isn't strictly necessary, but it can make some of the dashboards more readable, because they often contain only an entity's name without any supporting data.

Milestones, of course, are used to define and track other events in the software-development lifecycle. Important meetings, tool software upgrades, and requirement deliveries are just a few examples.

The Milestones interface contains a number of tabs. The Dashboard and Calendar tabs are common to all entities and were discussed earlier.

Milestones Report Tab

The Report tab in Figure 15-7 displays an interactive report that contains a list of milestones. The high-level filters let you select all or only future events as well as individual releases. The Edit link in the interactive report takes you to the milestone's Edit page, which contains intuitive items that are documented under their labels.

Figure 15-7. *Milestones tab*

By Owner Tab

The By Owner tab (see Figure 15-8) is a dashboard that summarizes the relationships between a single milestone and the other Team Development entities, broken down by owner. Knowing the number and status of the related features, to-dos, and bugs is a good management tool that is useful for controlling a sprint or time box that leads up to a release.
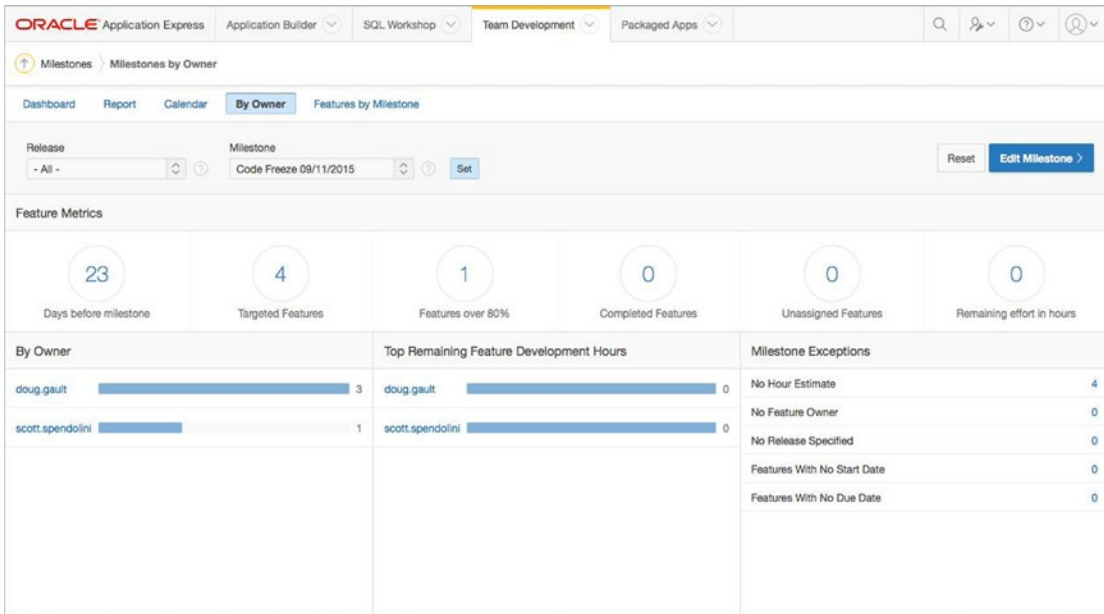
Figure 15-8. *Milestones By Owner tab*

Features by Milestone Tab

The Features by Milestone tab (See Figure 15-9) contains an interactive report that, in detail, illustrates the relationship between milestones and the features associated with the milestone. This is mainly a management report.

Figure 15-9. *Features by Milestone tab*

Features

Features describe an application from a high-level perspective. They're used at the beginning of a development project to define the scope well enough that budgetary estimates can be made for cost, schedule, and resource requirements. Once the project is approved and work begins, child features are added to describe, control, and track progress in more detail. Features are the heart of management status reports.

The Features Details page contains a number of tabs. The Dashboard, Calendar, Focus Areas, and Owners tabs aren't discussed here because they or their equivalents are common to all entities and their characteristics were mentioned earlier. The Report, History, and Progress Log tabs are highlighted next.

Features Report Tab

The Report tab (see Figure 15-10) contains an interactive report that, in turn, links to the Edit page for individual feature records. Because the data grid is an interactive report, each user can tailor it to their needs. You can choose your columns from approximately 50 attributes, sort to your taste, and even group related records together. Interactive reports are discussed in detail in Chapter 6.

Figure 15-10. *Features interactive report*

The Copy link is useful when you're entering a number of related features. For example, if you're entering 10 or 20 features that belong to a single application, you might expect many of the data attributes to be repeated on every record. The Copy link creates a new feature as an exact copy of the existing one in the interactive report. Only the name is changed. Once the new feature is created, you can then quickly edit it and change the attributes that are unique to that record.

The Edit link takes you to the Features Edit page (see Figure 15-11). You can easily explore the individual attributes on your own; their definitions are generally self-evident and are documented by clicking an attribute's label. However, there are areas on the Features Edit page that deserve more explanation:

- *User Interface, Testing, Documentation, Globalization, Security, and Accessibility regions*: These regions are used to track sub-lifecycles that are associated with a feature. For example, if a separate team is responsible for documentation, their progress can be tracked separately from the main feature (see Figure 15-12). If the documentation effort isn't being tracked, then the Documentation region can be excluded from the Features Edit page by turning it off in the Team Development settings area. See the subsection later in this chapter on "Team Development Settings."

- *Configurable lists of values (LOVs)*: Some LOVs, such as Status, are controlled by the APEX team. Others, such as Owner, are controlled by the development team. Instead of maintaining the developer-configurable LOVs in a separate maintenance area, these lists are maintained in place. For example, to add a new owner, you enter the appropriate name in the New Owner item and click the Apply Changes button, and that name is added to the LOV. Unfortunately, currently there is no easy and safe way to edit these configurable LOVs; we hope this ability will be added in a future APEX release.

Figure 15-11. *Features Edit page*
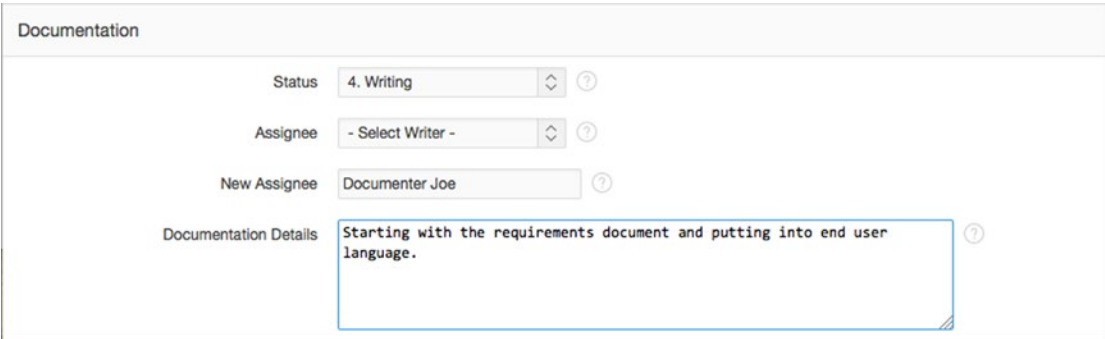
Figure 15-12. *Features Documentation region*

History Tab

The History tab (see Figure 15-13) presents a detailed audit trail of all changes made to all features. The tab presents an interactive report that contains information on the feature, column name, old value, new value, user who made the change, and when the change was made. You can view the report on the History tab for quality-assurance purposes or as a source of evidence if contractual disputes arise.

Figure 15-13. *Features History tab*

Progress Log Tab

The Progress Log tab in Figure 15-14 is, in effect, a diary. Stakeholders can, on the Features Edit page, enter free-form text into the progress log at any time. The progress log acts as a communication channel for the development team and can be used to record new ideas, reminders, a flash of brilliance, questions, and anything else that needs to be remembered.

Figure 15-14. *Features Progress Log tab*

To-Do Items

To-do items are the workhorses of Team Development. Team Development uses the term *to-do* as a noun. A to-do is a specific task that is assigned to an individual or team along with a due date. In simple terms, a to-do describes the *what*, *who*, and *when* of the task at hand.

To-dos can be subdivided into many child and grandchild tasks. This process of decomposition can be done to the point of diminishing returns. A rule of thumb is that the number of subtask levels should be appropriate for effectively controlling the tasks at hand. What is appropriate is, of course, a function of your team's culture. Also, in an agile software-development shop, the subtask levels tend to remain shallow due to the daily face-to-face communication between team members.

There are four tabs in the To-Dos module: Dashboard, To-Dos, Calendar, and Progress Log. You'll find working with these tabs to be easy and intuitive.

A handy navigation feature links the development environment directly to the To-Dos module. When a to-do has its context set to both an application and a specific page number within that application, a count of to-dos appears in the Team Development drop-down menu (see Figure 15-15). In the drop-down you can see counts of to-dos, bugs, feedback, and features. When you click the to-do count, the link takes you directly to the To-Dos Report page that has the interactive report filters set to list only the open to-dos associated with the Development page (see Figure 15-16). The developer can then quickly select one of the to-dos, see what needs to be done, and return to the Development page to complete the to-do task. This efficient navigation strategy also applies to bugs and feedback.
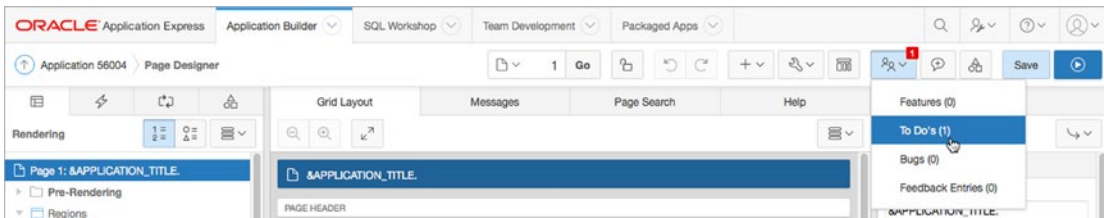
Figure 15-15. *Development page linked directly to the To-Dos module*

Figure 15-16. *To-Dos page filtered by an application's development count link*

Bugs

In the software-development world, bugs are a fact of life. We all hope that bugs are found and fixed in the unit- and system-testing cycles before a product is released to end users. Happily, this is generally true, with the caveat that a small number of bugs slip through into the production environment. Team Development's Bugs module is a practical environment that is used to track easy and hard bugs in the development, test, and production environments.

Easy bugs are usually caused by a coding error or a programmer's misunderstanding of a requirement. The symptoms are obvious, the root cause is simple to find, and the fix is almost trivial; for example, changing a plus sign to a minus sign or reorganizing an IF statement in the code. Easy bugs are usually found early in the product's development cycle and are, in general, caught by the programmers or testers. However, easy bugs must be recorded and reported so that improvements can be made to the coding process; a large number of easy bugs can turn out to be surprisingly expensive.

Hard bugs are, well, hard. They can be caused by design flaws anywhere in the system, subtle interactions between software systems, subtle interactions between software and hardware, and awkward interactions between users and the GUI. Hard bugs can have their own lifecycle, and the fix might be spread over several product releases. Team Development's Bugs and To-Dos modules can be used together to track a hard bug's resolution, even when it spans several product releases.

The Bugs module highlights how you can take advantage of Team Development's simplicity, extensibility, and flexibility. Simplicity makes tracking easy bugs almost trivial. A bug is found, and everything about the bug is recorded in Team Development's Bugs module, including the *what*, *who*, and *when* data. Easy bugs can stand on their own, or you can associate them with a to-do depending on how you organize your Team Development environment.

Extensibility and flexibility come into play when you deal with hard bugs that may take an extended amount of time to fix and may possibly require effort from several developers or teams. Team Development can handle this situation by linking a hard bug to a to-do (see Figure 15-17). The linked to-do is then set up as the parent of several child to-dos that are used to track the tasks required for the fix.
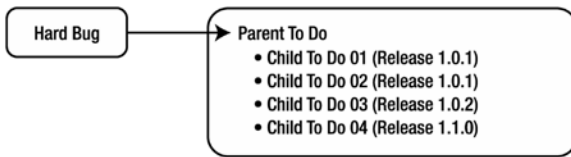
Figure 15-17. *Linking a hard bug to a parent to-do*

Feedback

Team Development's Feedback module is an APEX "sweet spot." Even if your team chooses not to use Team Development to manage its software-development efforts, you should at least consider using the Feedback module. The Feedback module provides a cost-effective channel through which end users, test team members, and even the developers themselves can send suggestions, comments, and bug reports directly to business analysts and developers. Responses to the feedback can optionally be communicated back to the persons who triggered the feedback. And feedback can be taken from all three typical environments: production, test, and development.

■ **Note**   The Feedback module is so cost effective because the cost-to-benefit ratio almost approaches zero. It takes only a few minutes to set up feedback in an APEX application, and it gives you a huge benefit.

Configuring Feedback

You begin configuring the feedback mechanism by creating a new page in your application. The Create Page wizard contains a Feedback Page option (see Figure 15-18). Select this option, and click the Next button.
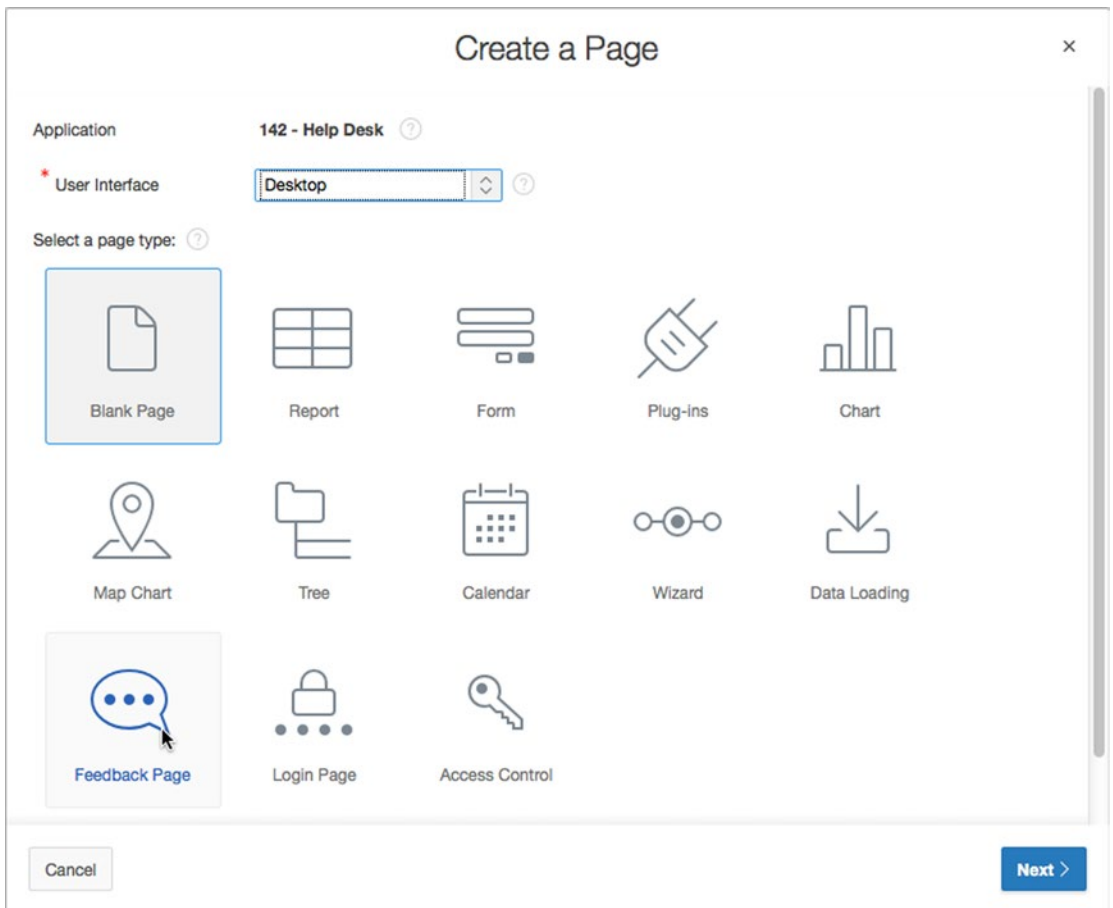
Figure 15-18. Feedback Page option in the Create Page wizard

The next page in the wizard sets up the details for the Feedback page (see Figure 15-19). You enter the page number and page name and then select a page template, form region template, and label template. You can also declaratively create a Feedback link on the application's global navigation bar. The defaults, in most cases, work well.

Figure 15-19. *Feedback page set up in the Create Page wizard*

However, one attribute requires a bit more explanation because its benefit isn't immediately obvious. The Extra Attributes field allows you to add up to eight custom fields to your Feedback page. These are sometimes called *flex fields*. These fields can be used to prompt end users for additional information when they submit feedback. For example, a public APEX website requires no login; therefore, the user's identity can't be captured automatically. The extra attributes can be used to capture the user's name and email so the feedback response can be sent to them.

Click the Create button and then run the application. You now see the Feedback link in the navigation bar (see Figure 15-20). If you can't use the default Feedback link in your design, you can move it anywhere you like in your application. The link uses the standard APEX ƒ?ρ syntax.
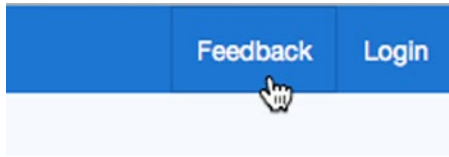
Figure 15-20. *Application with the Feedback link*

Polishing the Feedback Page

When you click the Feedback link, the Feedback popup page is displayed (see Figure 15-21). As you can see, the page needs a bit more work to polish it. Because this is a standard APEX page, you can complete the polishing in one or two minutes by clicking the Edit Page button to navigate to the Page Designer.
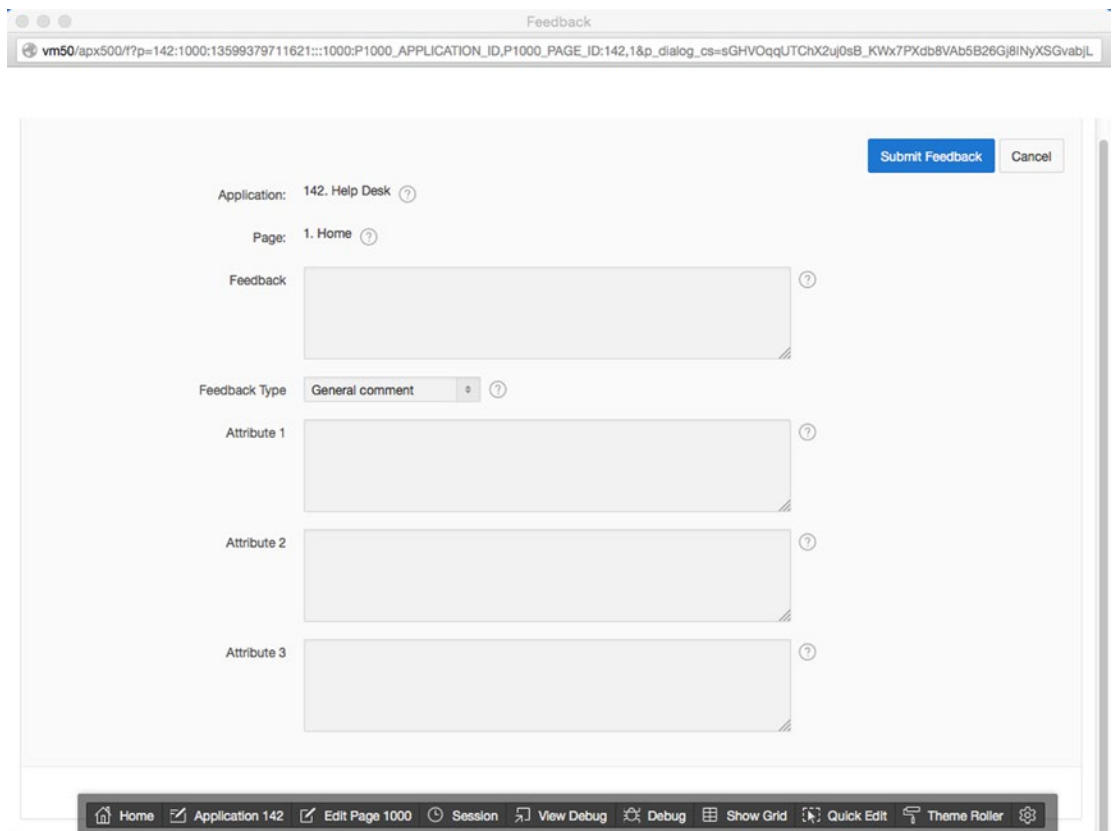
Figure 15-21. *Feedback page prior to some polishing*

In this example, let's change Attribute1, Attribute2, and Attribute3 to Name, Department, and Email. APEX automatically added these items to the Feedback page (see Figure 15-22) when you selected 3 in the Extra Attributes field in the Create Page wizard.
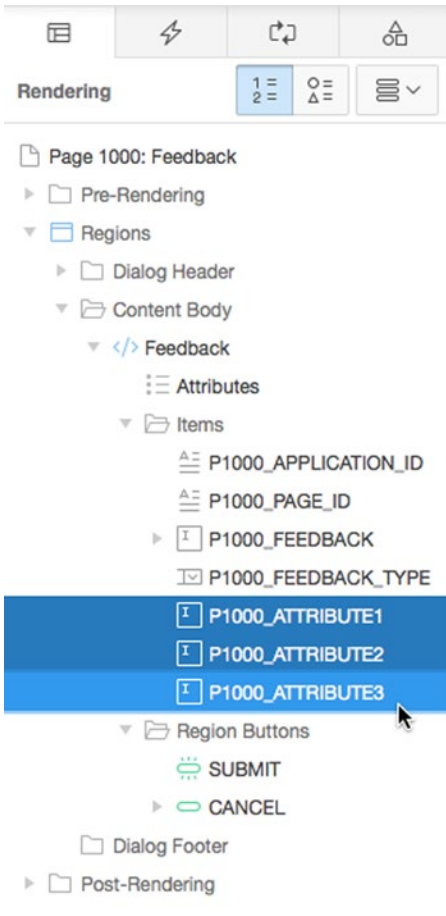
Figure 15-22. *Extra attributes on the Feedback page*

Configuring the extra attributes is a two-step process. First, rename the items so they describe the data, change the item types from Textarea to an item type that is appropriate for the data, and rename the label to something meaningful to the end users (see Figure 15-23).
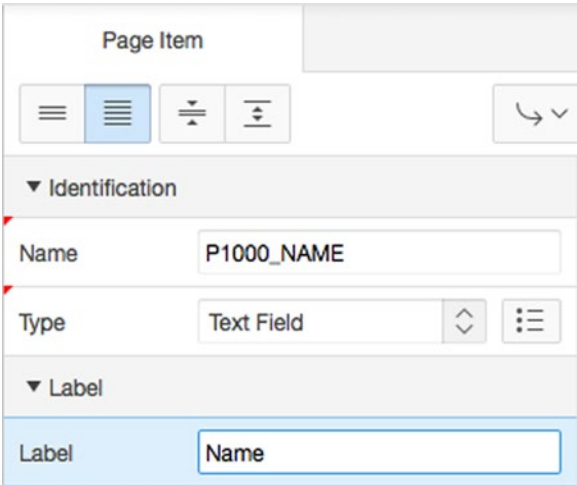
Figure 15-23. *Reconfiguring extra attributes*

Second, in the Page tab of the Tree Pane (see Figure 15-24), you must edit the Submit Feedback process so that the API calling parameters match the renamed and relabeled extra attributes (see Figure 15-25).
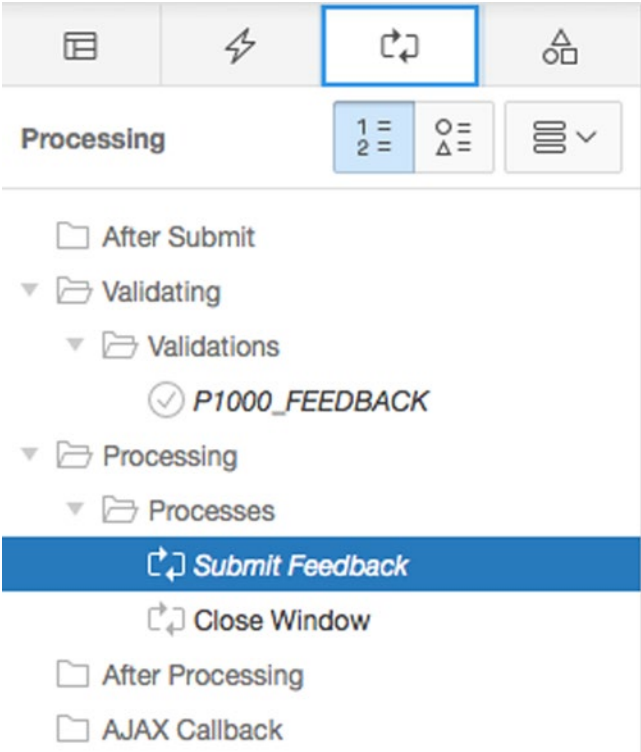
Figure 15-24. *Feedback Page's Processing region*

Figure 15-25. *Feedback Page Processing changes in the Submit Feedback process source*

After you complete the changes and rerun the application, you will see the finished Feedback page (see Figure 15-26). Enter some feedback into the page and click the Submit Feedback button. Then, go on to the next section and learn how to review the feedback that you and others have entered.

Figure 15-26. *Finished Feedback page with three extra attributes*

Viewing Feedback

You can review all feedback for an application from the Team Development page. Navigate to that page and then click the Feedback tab. You should see results similar to those in Figure 15-27.

Figure 15-27. *Feedback entry*

When you drill into an individual feedback record, you will find a wealth of data and information that can make your life as a developer much easier. The Feedback region contains a read-only description of each feedback record. The Disposition region is where you track feedback status, tags, developer comments, and public response. In addition, there are three buttons: Log as Bug, Log as To Do, and Log as Feature. These buttons create a new Team Development entity and copy data from the feedback record to the new entity, which saves time and ensures accuracy. The Follow Up region is like a diary; it's a list of remarks that are added over time as the feedback is processed. This is handy when several people must review the feedback before action is taken. The extra attributes that you added to the Feedback page are displayed in the Additional Attributes region. The remaining regions display read-only data that describes the application context, the runtime environment (browser type and version), and the entire session state. If a bug is reported, the developer has all the information required to reproduce the bug, which is a valuable benefit.

Responses to Feedback

The feedback mechanism contains a response table. In principle, responses should be sent to the users who initiated the feedback. However, there is no easy and declarative mechanism that enables you to send responses to users. If you want to send responses to users, you first need to address a number of design issues. For example, do you want to broadcast responses to all users or send individual responses to individual users? Do you want to use email or create reports? Do you want to send responses to a team? Do you want to route the responses based on a feedback classification scheme?

After you design your response strategy, you can then build a tool that fulfills the requirement by writing some PL/SQL code and accessing the APEX views that expose all of the Team Development data. The details of building this tool are beyond the scope of this book, however.

Communication Between Workspaces

Team Development is a property of a workspace. Many professional shops maintain multiple workspaces for production, testing, and development environments. This means that if an end user enters a feedback record, that record resides in the production workspace, and the developers can't see it. This situation is easily remedied by using APEX's existing export/import functionality. APEX version 4.0 and above has added feedback to the list of entities that can be imported and exported to and from a workspace. This makes it possible to export the production feedback and import it to the testing environment, where the business analysts can evaluate it. If required, the feedback can be exported from the testing environment and imported to the development environment, where the developers can evaluate and then log it as a bug, to-do, or feature.

423

Team Development Utilities

Team Development Utilities (see Figure 15-28) are a miscellaneous set of utilities that help you manage the Team Development environment. The links to the individual utilities are found on the right side of the Team Development home page. The All Utilities link takes you to a page that provides a menu of all utilities with a short description of each. Clicking on the individual utility links takes you directly to that utility.

Figure 15-28. *Team Development Utilities*

Team Development Settings

A small number of defaults that are global to the workspace are configured on the Team Development Settings page (see Figure 15-29). The Enable Tracking Attributes region is used to turn on/off the feature regions that track detailed work related to the user interface, testing, documentation, globalization, security, and accessibility. When you set these attributes to No, the corresponding region on the feature Details page isn't displayed.
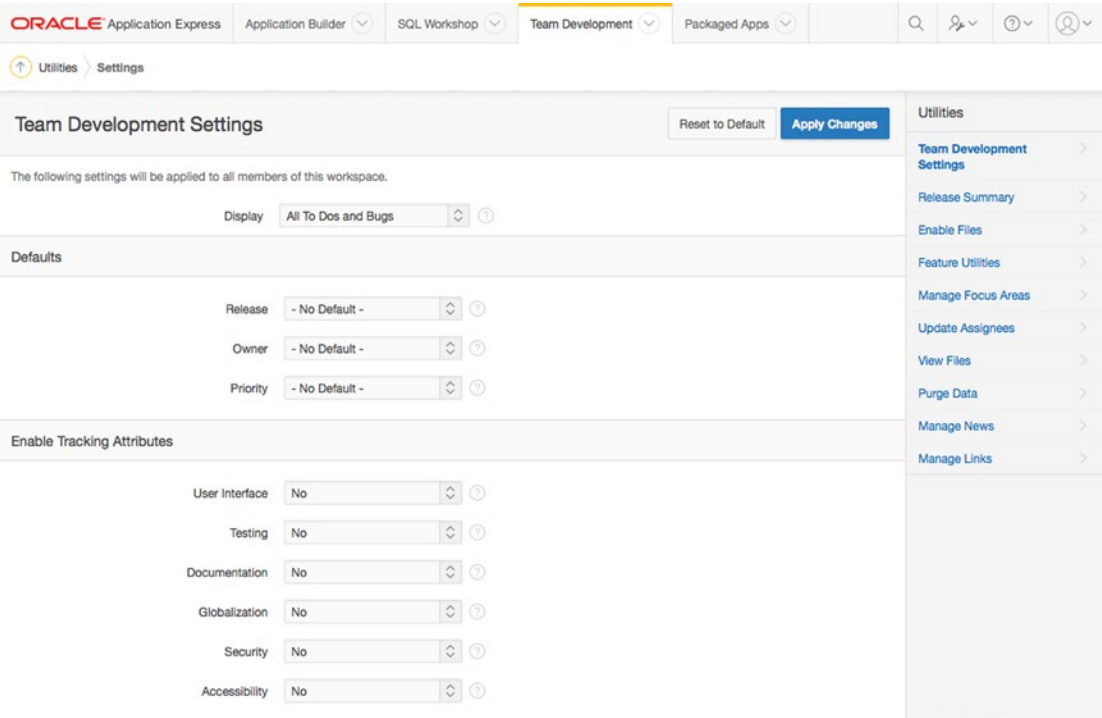
Figure 15-29. *Team Development Settings page*

Release Summary

The Release Summary page is a management report. It's organized by release name and can be filtered by developer and release name. Figure 15-30 shows a small part of this comprehensive report.
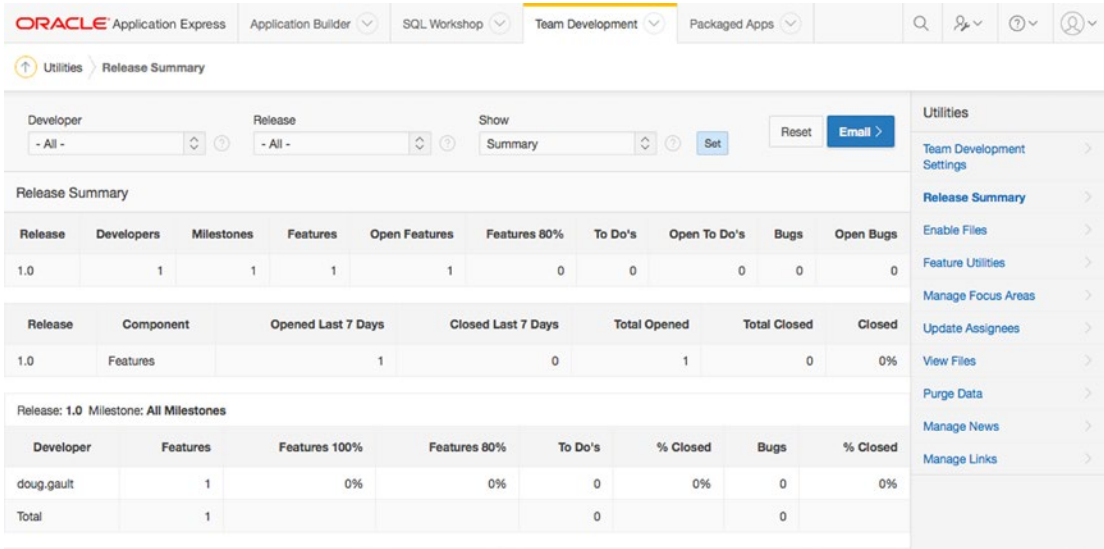
Figure 15-30. *Release Summary page*

Enable Files

The Enable Files link is a shortcut to the Workspace Preferences screen. Here, you can set the value of Enable File Repository, which specifies whether or not files may be uploaded to Team Development. Selecting Yes will create a local APEX$ table to store the files. The feature is only available if the Instance Administrator also enables it at the instance level.

Feature Utilities

The Feature Utilities (see Figure 15-31) perform bulk updates to the Team Development data. You can assign milestones to unassigned features, set feature due dates, change milestones, and push due dates for open features. Before you do this, having a clean backup is well advised.
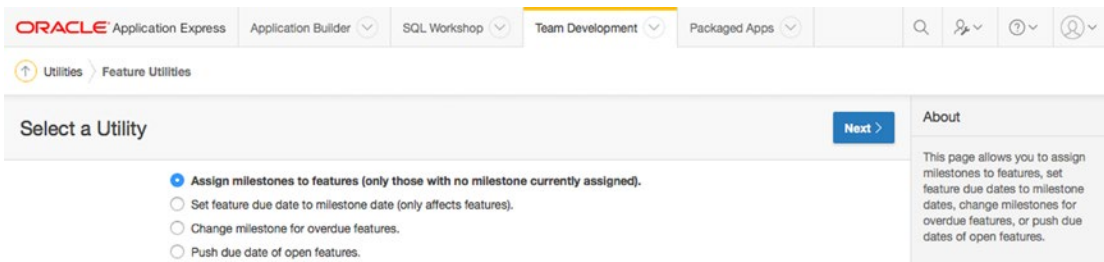
Figure 15-31. *Feature Utilities*

Manage Focus Areas

The Manage Focus Areas utility allows you to view the Focus Areas that have been created, the feature count per Focus Area, the Distinct Owners, and the last edit of a feature in a given Focus Area. By editing, you are also able to rename a Focus area.

Update Assignees

The Update Assignees utility allows you to reassign various components from one assignee to another. You can do this for either all releases or a specific one, and you can also choose the components to reassign. See Figure 15-32.
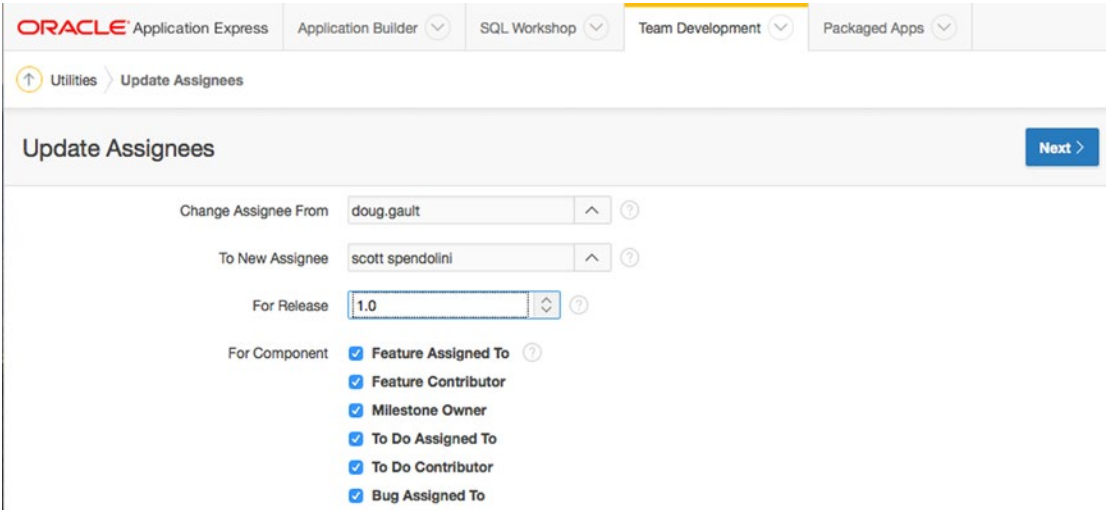
Figure 15-32. *Updating the assignee for various components*

View Files

If the feature has been enabled at the workspace level, developers can attach files to a Feature, To-Do, or Bug. The View Files report will list all of the files that have been uploaded, and some metadata about the files. You are also able to download the files from this report. If the feature to do so has not been turned on, the link will read "Enable Files."

Purge Data

This utility will allow you to purge data for selected component types. This will delete all entries for the selected types with no opportunity for recovery. This is useful if you're starting a brand new development cycle or if you need to clear test data from Team Development.

Manage News

On both the APEX home page and the Team Development home page is a region that contains news messages. You can add news items (see Figure 15-33) by clicking the Manage News link or by clicking the plus-sign icon in the News region.
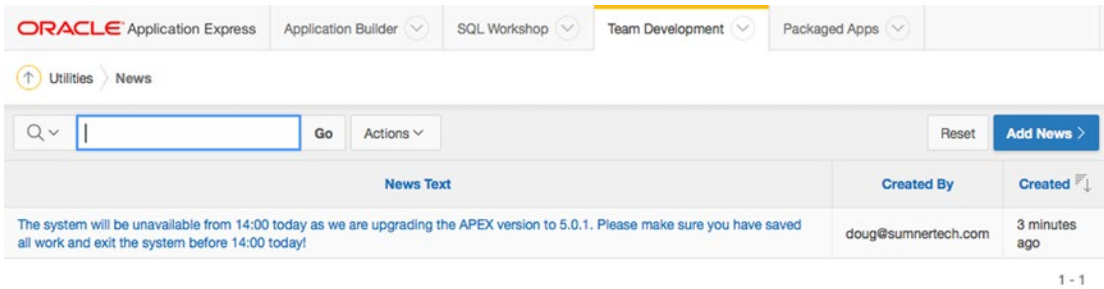
Figure 15-33. *Managing the News section*

The News region is handy for broadcasting development news when the team isn't co-located; the successful or failed promotion of a release to the test environment is an example. Teams that are co-located and have a daily status meeting probably won't use this region.

Manage Links

The Manage Links feature is a simple list of links to documentation, other systems, and web pages that you can easily define as any valid URL (see Figure 15-34). The links let developers quickly navigate to URLs that have been approved by the development team. The entire team then shares common documentation, such as SQL or PL/SQL references that can help the team adhere to common development styles and standards. This, in turn, helps to encourage consistency, which greatly improves the software-development process.
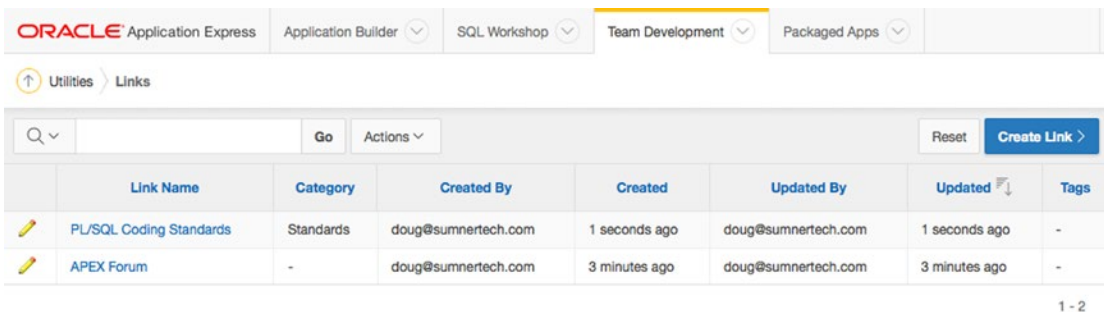
Figure 15-34. *Manage Links feature*

User Roles for Team Development

Access to Team Development is useful for stakeholders both inside and outside the development team. The team lead and senior developers should have access to Team Development. Access by junior developers depends on the team's culture and trust level. Interested stakeholders who are outside the development team could include the project manager, test team, and business analysts.

Access to the APEX development environment is controlled in the APEX Administration area under the Manage Users and Groups menu. This area maintains the list of APEX users, dictating what level of access each user has (see Figure 15-35). An outside stakeholder is set up without administrator and developer privileges; however, the Team Development Access field is set to Yes. When they log in to the APEX development environment, they see only the Team Development area and nothing else. There is only one problem at this time with this scenario: outside stakeholders can't see the applications in the Application drop-down lists.

Figure 15-35. *Account Privileges for a project manager, tester, or business analyst*

Summary

Team Development is a software-development tool that has been tailored to work in the APEX development environment. The five main entities (milestones, features, to-dos, bugs, and feedback) work together in a framework that is simple, extensible, and flexible. Teams that embrace agile software-development methodologies will find APEX's Team Development tool to be a comfortable fit with their culture.

■ ■ ■

Dynamic Actions

One of the most exciting features introduced in recent versions of APEX was dynamic actions, which provide the ability to declaratively define complex client-side behavior such as validations, highlighting, alerts, setting page values, and so on, without the need to hand code large amounts of JavaScript.

Dynamic actions have been significantly extended since their introduction, providing more flexibility and functionality declaratively. This helps the developer break away from the traditional server-side scripting model by executing the dynamic-action logic on the browser instead of incurring a round trip to the server.

Dynamic actions are event-driven just as manually written JavaScript would be. But APEX uses the declarative information provided to generate the required JavaScript code, which is then implemented at runtime. This chapter will examine and implement a number of different dynamic actions so you can get a feel for what they can achieve.

Dynamic Action Benefits

One of the major advantages of using declarative dynamic actions as opposed to hand-coded JavaScript is that dynamic actions understand and can take advantage of APEX core objects such as regions and items, allowing easy reference and manipulation. Another benefit of using declarative logic is that, when you choose to upgrade to the next release of APEX, the framework around dynamic actions will ensure that any code generated will be compatible with the new version of APEX.

But beyond the base benefits of the declarative nature of APEX, dynamic actions let you code very complex client-side actions without having to learn a whole new technology to do so. In fact, it's likely that you could code upward of 80% of everything you need to do with nothing more than the Dynamic Action wizard, SQL, and PL/SQL.

However, because JavaScript is the de facto standard for coding browser interactivity, it's also likely that at some point you'll be forced to learn a bit about JavaScript. Learning JavaScript is beyond the scope of this book. After all, you bought this book to learn APEX. But if you do want to learn more about JavaScript, Apress has a number of excellent books on the topic.

Breaking Down Dynamic Actions

In their original incarnation, dynamic actions were split into two categories: standard and advanced. The only real difference between these two categories was what the related wizard let you achieve. Under the covers, both dynamic action types were identical, and once you left the wizard, all options were available to you. The more recent versions of APEX (including 5.0) have done away with this artificial separation. The definition of a dynamic action can be broken down into the following components:

Identification: Defines the name of the dynamic action and its execution sequence

When: Defines when the action will be fired. You can choose the event, the object or objects that will participate in causing the action to fire, and any condition that applies to the event.

Actions: Dynamic actions can contain both True and False action sets. The True action set is executed if the defined event occurs for the selected objects and any condition applied evaluates to TRUE. The False action set executes if the defined event occurs for the selected objects and any condition applied evaluates to FALSE.

Affected elements: Identifies which objects on the page are affected by the dynamic action

As with other parts of APEX, dynamic actions support conditions, authorizations, and build-option features.

Dynamic Actions in the Help Desk Application

Dynamic actions are all about making your application's user interface easier for the user to utilize. In the following exercises, you will implement increasingly complex dynamic actions to make the interface of your application more robust.

Starting Simple

In the first exercise you will edit the Contact Us form on page 3 of the Help Desk application. Although there is nothing wrong with the form as it stands, you've been asked to deny input into the Body text area until the user has entered something into the From email address field.

To create a dynamic action to do this, follow these steps:

1. Edit **Page 3** of your application.

2. Right-click the **P3_FROM** item and choose **Create Dynamic Action** from the context menu. Using the menu shown in Figure 16-1 is the most direct way to create a dynamic action.
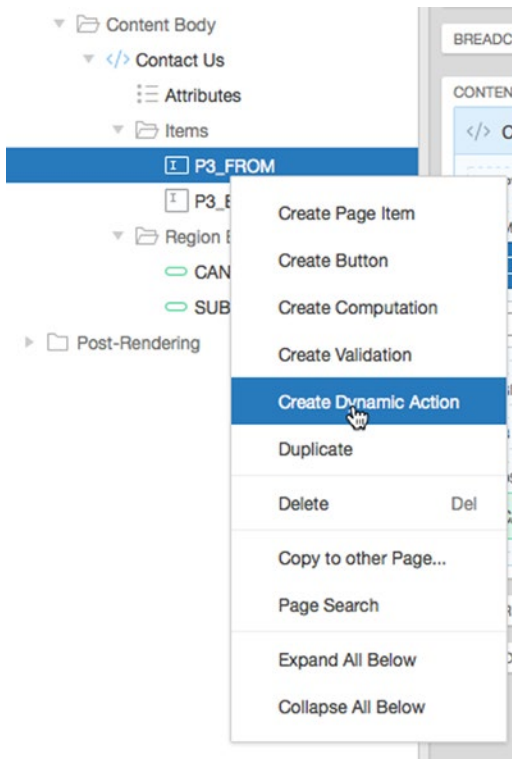
Figure 16-1. *Using the right mouse shortcut to create a dynamic action*

3. As shown in Figure 16-2, enter `Disable Email Body` for the **<u>Name</u>** of the dynamic action. Just as with other components, the more descriptive the name, the easier it is to identify.
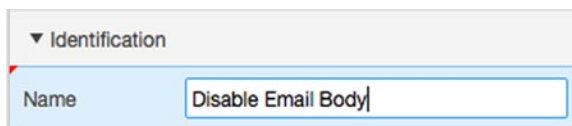
Figure 16-2. *Specifying a name for the dynamic action*

4. Leave **<u>Event</u>** set to **Change** and set **<u>Condition</u>** to **is null**, as shown in Figure 16-3.
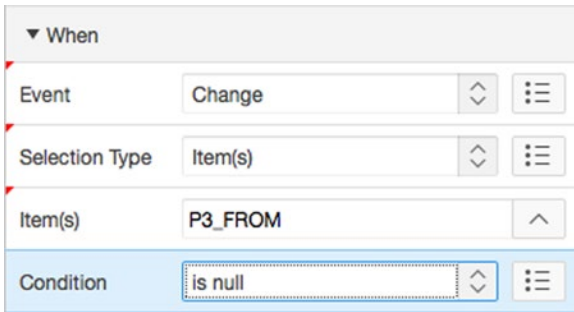
Figure 16-3. *Setting a dynamic action's Event and Condition for execution*

5. In the Rendering Tree on the left under the **True** node, click the **Show** action (highlighted in red), as shown in Figure 16-4.
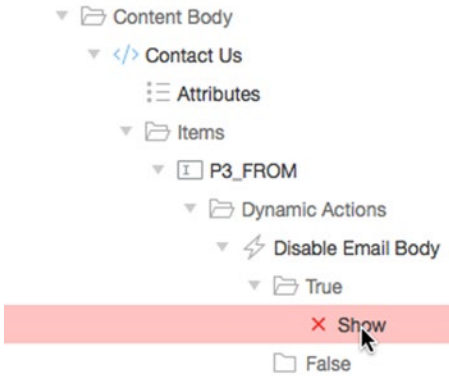
Figure 16-4. *Selecting the default True action so that it can be edited*

6. In the Properties Editor, select **Disable** for **Action**, set **Item(s)** to **P3_BODY**, and set **Fire On Page Load** to **Yes**, as shown in Figure 16-5.

Figure 16-5. *Setting the True action details*

7. In the Rendering tree, right-click the **False** node for the Dynamic Action and select **Create False Action** from the context menu.

8. In the Properties Editor, select **Enable** for **Action**, set Selection Type to Item(s), set **Item(s)** to **P3_BODY**, and set **Fire On Page Load** to **Yes**, as shown in Figure 16-6.

Figure 16-6. *Setting the False action details*

9. **Save** and **Run** the page.

Recapping the steps in the exercise, you created a dynamic action that fires any time the Change event for the item P3_FROM is triggered. You set the condition so the action fires only when P3_FROM is null. The action is set to Disable, which disables an item so the user can't navigate to it, and you chose to run the dynamic action whenever the page is loaded. This ensures that the affected item is disabled to start with. You also created the opposite False action. This enables the item whenever the Change event is fired and P3_FROM is not null. In both the True and False actions, you chose P3_BODY as the affected element. This indicates that it's P3_BODY that is enabled and disabled depending on the state of the P3_FROM item.

When running page 3, note that the Body item is disabled until you enter something in the From item and navigate away. Conversely, if you delete all content from the From item, the Body item again becomes disabled, but only after you navigate away from P3_FROM. This is acceptable, but it would be nicer if the Body item became enabled as soon as you typed anything in the From item. Let's set the triggering event to be Key Release instead of the default Change event. Here's what to do:

1. Edit **Page 3** of the application.

Dynamic actions that are triggered by a form element can be edited from one of two places. First, you can see the dynamic action defined in the tree under the triggering element, as shown in Figure 16-7. However, you can also navigate to the Dynamic Actions tab in the Tree Pane (shown in Figure 16-8) to see all dynamic actions for the current page.

▼ </> Contact Us
　　:≡ Attributes
　　▼ 🗁 Items
　　　▼ 🗉 P3_FROM
　　　　▼ 🗁 Dynamic Actions
　　　　　▼ ⚡ Disable Email Body
　　　　　　▼ 🗁 True
　　　　　　　　Disable
　　　　　　▼ 🗁 False
　　　　　　　　Enable
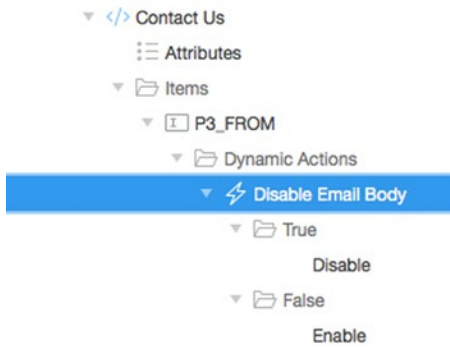
Figure 16-7. *Dynamic actions as they appear in the Rendering Tree*

Dynamic Actions

▸ 🗀 Actions Fired on Page Load
▼ 🗁 Events
　　🗀 Page Load
　▼ 🗁 Change
　　▼ ⚡ Disable Email Body
　　　▼ 🗁 True
　　　　　Disable
　　　▼ 🗁 False
　　　　　Enable
　　🗀 Click
　　🗀 Dialog Closed
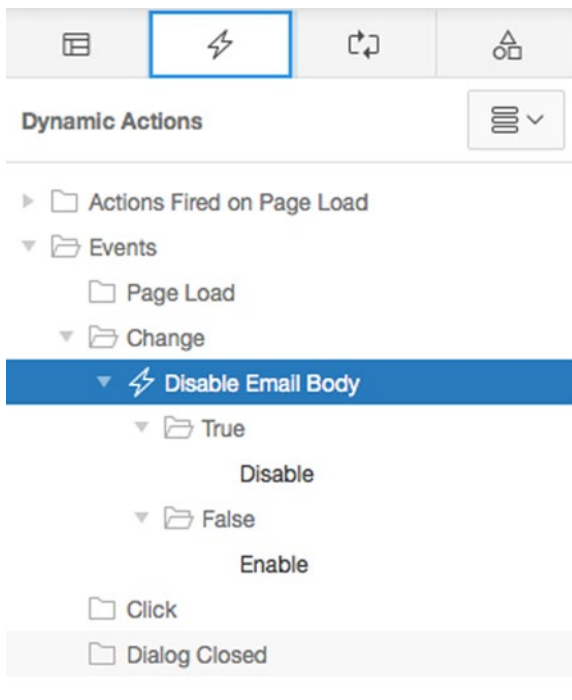
Figure 16-8. *Viewing the Dynamic Actions tab*

2.　From either the Rendering Tree or the Dynamic Actions Tab, click the **Disable Email Body** dynamic action to edit it.

3.   In the **When** section of the Properties Editor shown in Figure 16-9, change **Event** to **Key Release** and click **Save**.
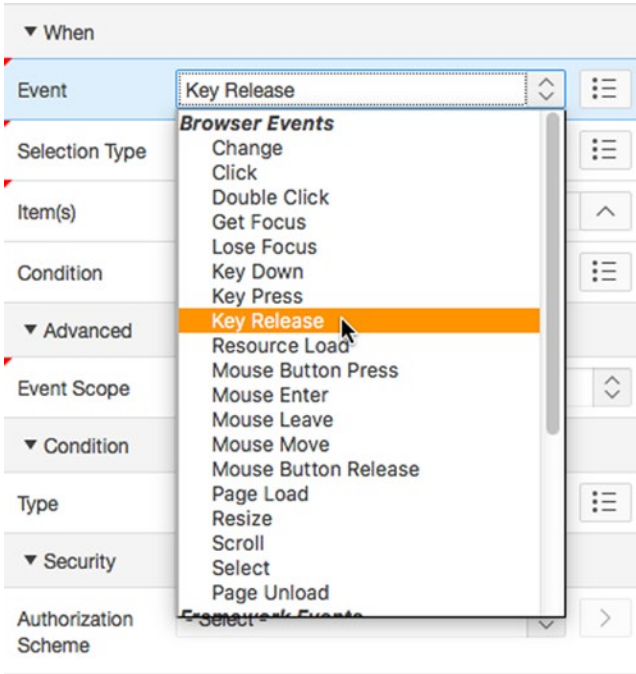
Figure 16-9. *Specifying Key Release as the event*

To test this change, run page 3 again. The page opens and looks like Figure 16-10. Start typing an address into the From item. As soon as any value is entered, the Body item becomes enabled, as in Figure 16-11. Conversely, the moment you delete all content from the From item, the Body item becomes disabled again.
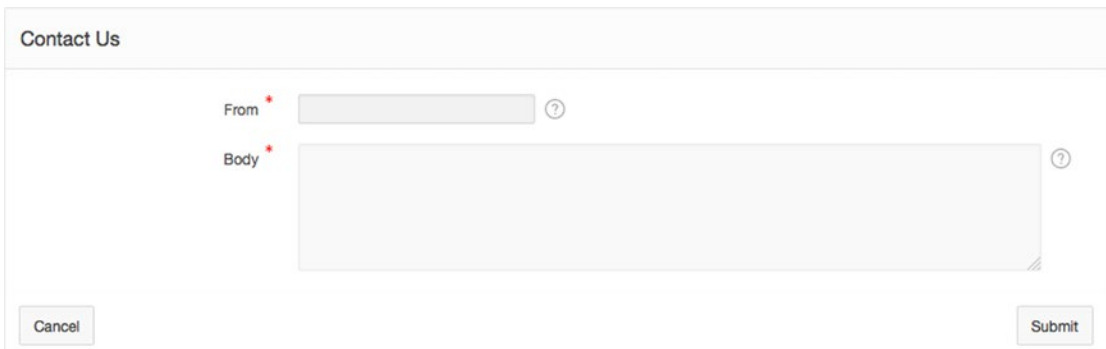
Figure 16-10. *Before entering a value in the From field*

Figure 16-11. *After entering a value in the From field*

Using Page-Level Events

Dynamic actions give you full control over the triggering events and actions performed. Events can be triggered at various levels, including when the page loads, unloads, is resized, and so on.

In the next exercise you will use the Page Load event to pop up a dialog reminding the user to be as verbose as possible when they enter their ticket. Follow these steps:

1. Edit **Page 2** of the application.

Because this event isn't tied to an individual item, but instead to a page-level event, you need to create the dynamic action accordingly.

2. Navigate to the **Dynamic Actions** tab of the <u>Tree Pane</u>.

3. Right-click the <u>Page Load</u> node in the tree and select **Create Dynamic Action** from the context menu, as shown in Figure 16-12.
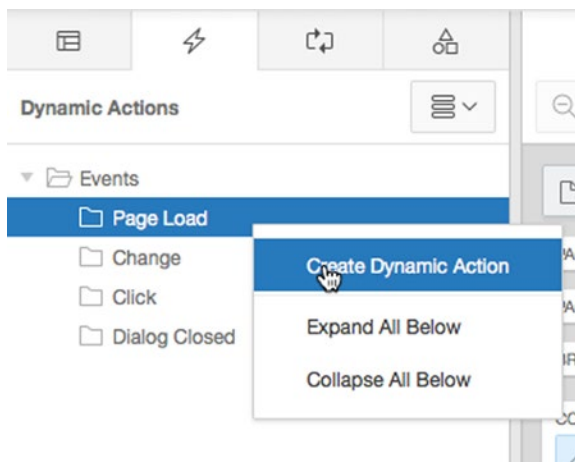
Figure 16-12. *Creating a dynamic action at the page level*

4. In the Properties Editor, enter `Alert User` for **<u>Name</u>**.

5. In the Dynamic Actions Tree under the **<u>True</u>** node, click the **Show** action (highlighted in red) to edit its properties.

6. Select **Alert** for **<u>Action</u>** and enter "`Please be sure to be as complete as possible when entering the details of your issue.`" for **<u>Text</u>**, as shown in Figure 16-13.
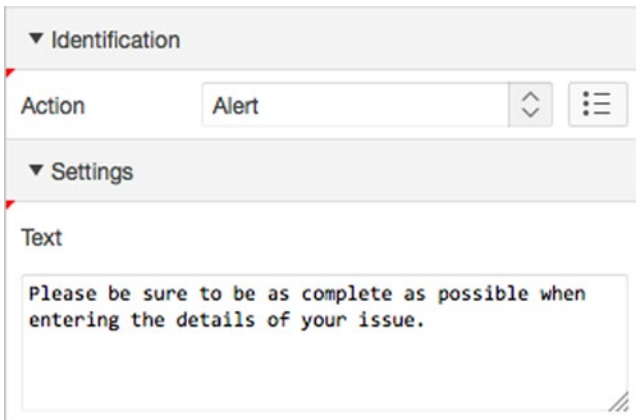
Figure 16-13. *Setting Action and Text for the dynamic action*

7. **Save** and **Run** the page.

Running page 2 now generates a pop-up every time you load the page. Figure 16-14 shows the pop-up as seen when using the Chrome browser for Mac OS X.
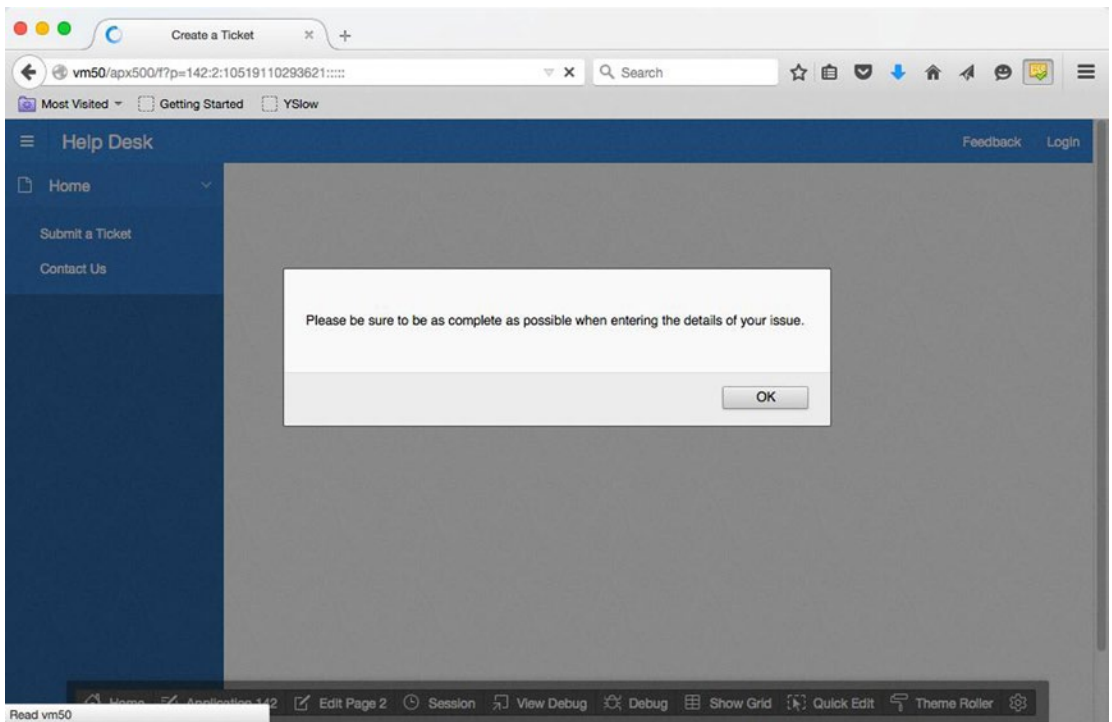
Figure 16-14. *Alerting the user*

■ **Note** Both the Alert and Confirmation actions available to you in APEX take advantage of the native dialogs provided by the browser being used by the end user. You have no control over the look and feel of these dialogs, and each browser may render them differently. If you need control over the look and feel of dialogs, you likely need to consider using a built-in APEX Modal dialog or coding your own dialogs based on jQuery.

Dynamic Actions with Multiple Triggering Elements

Dynamic actions also give you the opportunity to define multiple triggering elements. Using this method, you only need to create a single dynamic action to catch the events of several page items.

Your public ticket-entry page contains several page items that shouldn't be left blank. However, your APEX validations won't fire until the user submits the page. In this exercise, you will create a dynamic action that checks each of these page items as you navigate through the form to see if you left the values null. If a value is null, the background color of the item will be set to pink using the background style element. If it is not null, the background of the item will be set back to white. Follow these steps:

1. Edit **Page 2** of the application.

2. Navigate to the **Dynamic Actions** tab of the **Tree Pane**.

3. Right-click the **Events** node in the tree and select **Create Dynamic Action** from the context menu.

4. In the Properties editor, enter Highlight Null Values for **Name**.

5. Set **Event** to **Lose Focus**, set the Selection Type to Item(s), and enter the following for the **Items** field, as shown in Figure 16-15:

   P2_SUBJECT,P2_DESCR,P2_CREATED_BY
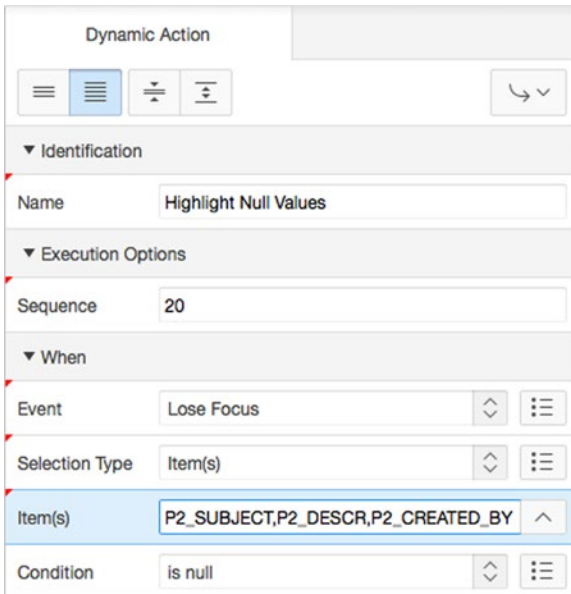
Figure 16-15. *Creating a dynamic action with multiple triggering elements*

6. Set **Condition** to **is null**.

7. In the Dynamic Actions Tree under the **True** node, click the **Show** action (highlighted in red) to edit its properties.

8. Set **Action** to **Set Style**.

9. In the **Settings** section, enter background for **Style Name** and pink for **Value**.

10. In the **Affected Elements** section, set **Selection Type** to **Triggering Element**.

11. Set **Fire On Page Load** to **No**.

12. In the Dynamic Actions tab of the Tree Pane, right-click on the **False** node of the Highlight Null values dynamic action and select **Create FALSE action**.

13. Set **Action** to **Set Style**.

14. In the **Settings** section, enter background for **Style Name** and white for **Value**.

15. In the **Affected Elements** section, set **Selection Type** to **Triggering Element**.

16. Set **Fire On Page Load** to **No**.

17. **Save** and **Run** the page.

By entering a comma-separated list of page items, you indicate that the Lose Focus event should fire when the user navigates away from any of these items. When the dynamic action fires, it checks to see if that item is null and sets it to the appropriate color. The dynamic action knows which item's background color to set by referencing the triggering element for the affected element.

Run page 2 of the Help Desk application and tab through each field, leaving them all blank. You should notice that as you leave a blank field it immediately turns pink. If you go back and enter text into a pink field and then navigate away, the background is set to white.

■ **Note**   Depending on the browser you're using, you may see that after the pop-up message is dismissed, the Subject field turns pink. This has to do with the order of precedence some browsers give to JavaScript events. Certain browsers place the cursor in the initial page item prior to raising the PageLoad event. Once the PageLoad event fires, the Subject field loses focus, and the LoseFocus event fires. When you have multiple dynamic actions on a page, which you often will, you need to make sure they don't adversely affect one another.

Dynamic Actions Using PL/SQL

Dynamic actions are architected to be an extensible framework, giving the developer full control over coding complex actions that might not be available in a purely declarative environment. In the spirit of UI usability, you should help the user adhere to the business rules of the application without introducing undue work for them. In this exercise, you will take the requirement for P2_CREATED_BY to be entered in uppercase and use SQL and PL/SQL to create a dynamic action that alters the user's input to uppercase, no matter what they enter. Here are the steps:

1. Edit **Page 2** of the application.

2. Navigate to the **Page Rendering** tab in the Tree Pane.

3. Right-click the **P2_CREATED_BY** item and choose **Create Dynamic Action** from the context menu.

4. Enter Change Case to Upper for **Name**.

5. Set **Event** to **Lose Focus**, set **Condition** to **is not null**.

6. In the Dynamic Actions Tree under the **True** node, click the **Show** action (highlighted in red) to edit its properties.

7. Set **Action** to **Set Value,** and in the **Settings** section, select **PL/SQL Expression** for **Set Type**.

8. Enter UPPER(:P2_CREATED_BY) for **PL/SQL Expression** and P2_CREATED_BY in **Page Items to Submit**.

9. In the **Affected Elements** section, set **Selection Type** to **Triggering Element**.

10. Set **Fire On Page Load** to **No**. See Figure 16-16.

Figure 16-16. *Using a PL/SQL expression as the body of a dynamic action*

11. **Save** and **Run** the page.

444

Here you use the PL/SQL UPPER expression to take the user's input and convert it to uppercase. You reference the value the user entered by using the bind variable :P2_CREATED_BY. However, because the value the user entered into the web browser has not been submitted to APEX, that value isn't currently in session state. That's why you need to include it in the list of page items to submit. If you needed to reference several values of user input, you must enter them all as a comma-separated list.

Now, when page 2 is run, any text entered in the Created By field is made uppercase when the user exits the field.

■ **Note**   Dynamic actions that use SQL or PL/SQL for their conditions or body actually make a call back to the database server to run the code in question. Depending on the weight and complexity of the code, this could potentially introduce performance issues. Save the use of SQL and PL/SQL for actions that require interaction with the database to retrieve data that isn't available from directly within the page.

Dynamic Actions Using JavaScript

In addition to PL/SQL, you can use JavaScript in dynamic actions. In this exercise, you will use JavaScript to determine the onscreen status of a ticket that you're editing on page 210. If the user has set the status to CLOSED, the dynamic action will automatically set the Closed On date to today's date:

1. Edit **Page 210** of the application.

2. Navigate to the **Dynamic Actions** tab of the Tree Pane.

3. Create a new dynamic action by right-clicking the **Events** node in the tree and selecting **Create Dynamic Action** from the context menu.

4. Enter AutoFill Closed_On Date for **Name**.

5. Make sure **Event** is set to **Change**, set **Selection Type** to **Item(s)**, and enter P210_STATUS_ID for **Item(s).**

6. Set **Condition** to **JavaScript expression**.

7. Locate and open the file ch16_javascript.txt. This file can be found where you unzipped the files associated with this book.

Examine the JavaScript string that is being used as the body of the condition. This may seem very cryptic at first, but when it's broken down, it's quite straightforward. Let's look at it in pieces:

```
this.triggeringElement.options[this.triggeringElement.selectedIndex].text == 'CLOSED'
```

The keyword **this** references the JavaScript event that kicked off the chain of events to start with, and **triggeringElement** references the item on the page that was at the root of the event. So, in this case, **this.triggeringElement** is talking about P210_STATUS_ID.

Here's where a little developer knowledge has to be introduced. Being the developer, you know that P210_STATUS_ID is a select list, and that a select list has from one to many values the user can select. In HTML, these values are called *options*.

Because of the way you've declaratively defined the P210_STATUS_ID select list, only one option can be selected at a time. You can access the option that is currently selected on the page by using the JavaScript **this.triggeringElement.selectedIndex**. The square brackets use that index to reference the selected option from the P210_STATUS_ID select list.

Although you could reference the **value** of the selected option, that would only give you the ID of the selected status. You'd then have to make a round trip to the database to find out the text status. Instead, you can use the **.text** JavaScript method to get the text that the select list is displaying to the end user and see what they selected.

Once you have that, you can then compare it to the value you're looking for, which is CLOSED.

1. Copy the contents of the file into **Value**, as shown in Figure 16-17, and click **Next**.

Figure 16-17. *Using JavaScript for the condition text of a dynamic action*

2. In the Dynamic Actions Tree under the **True** node, click the **Show** action (highlighted in red) to edit its properties.

3. Set **Action** to **Set Value**, set **Set Type** to **PL/SQL Expression**, and enter SYSDATE for **PL/SQL Expression**.

4. In the Affected Elements section, set **Selection Type** to **Item(s)** and set **Item(s)** to **P210_CLOSED_ON**.

5. Make sure **Fire on Page Load** is set to **No**.

   Although you want the CLOSED_ON date to be set when you choose a status of CLOSED, you want anything that is currently entered in the CLOSED_ON date to be removed if you choose any status but CLOSED. So, use the False action of the dynamic action to do this.

6. In the Dynamic Actions tab of the Tree Pane, right-click on the **False** node of the AutoFill Closed_On_Date Dynamic Action and select **Create FALSE action**.

7. Set **False Action** to **Set Value**.

8. Set **Set Type** to **PL/SQL Expression**, and enter NULL for **PL/SQL Expression**.

   Although P210_STATUS_ID triggers the dynamic action to fire, P210_CLOSED_ON is the affected element.

9. Set **Selection Type** to **Item(s)** and set **Item(s)** to **P210_CLOSED_ON**.

10. Make sure **Fire on Page Load** is set to **No**.

11. **Save** and **Run** your application.

Run your application and edit any ticket using page 210. Change the status to CLOSED and then back again to any other status. You will see the value for the Closed On date being set and cleared according to the value you choose for the select list.

Summary

Dynamic actions have many uses and are extremely flexible. However, you must make sure that multiple dynamic actions on the same page don't interfere with each other. Also, because dynamic actions run as JavaScript in the browser, try to do as much as you can declaratively, or with JavaScript, without resorting to SQL or PL/SQL. This reduces the number of calls to the database server and avoids potential performance bottlenecks.

It's probably inevitable that you'll be required to learn at least a little JavaScript to achieve more-complex results. JavaScript syntax isn't hard to learn, and it can be a useful addition to your skillset as a web application developer.

■ ■ ■

Page Designer Walkthrough and Reference

One of the challenges to productivity in versions of APEX prior to 5.0 was the need to drill down into the specific item you wanted to edit before you could change any of its properties. This meant learning where these items existed in the Tree View, the context menus, and the actual edit screens. It also meant that editing multiple items on a page could be quite tedious.

APEX 5.0 has completely reengineered the process of building and editing pages. The familiar Tree View that was introduced as part of APEX 4.0 has been retired and replaced by a new Page Designer. No longer do you drill down to separate pages to edit individual items—now everything is done on a single page.

While this can mean a huge leap forward in productivity, at first it can be quite daunting, especially to those of us who have spent significant amounts of time becoming familiar with the methods of APEX 4.0.

This section will walk you through the new Page Designer section by section and introduce the nomenclature used throughout this book to refer to its sections and functionality. This will also be a good place to refer back to if you're trying to remember where to find something, or are wondering what a particular button or item does.

Page Designer Overview

While the new Page Designer might seem like a radical departure from the way things were done in previous versions of APEX, it's actually a tried and tested design. Just think of Visual Studio, Eclipse, NetBeans, and the myriad other IDEs in the market. They all conform to a very similar design pattern, which was used to model the new APEX 5.0 Page Designer.

Figure A-1 depicts the Page Designer, shown editing Page 1 of the Sample Database Application.
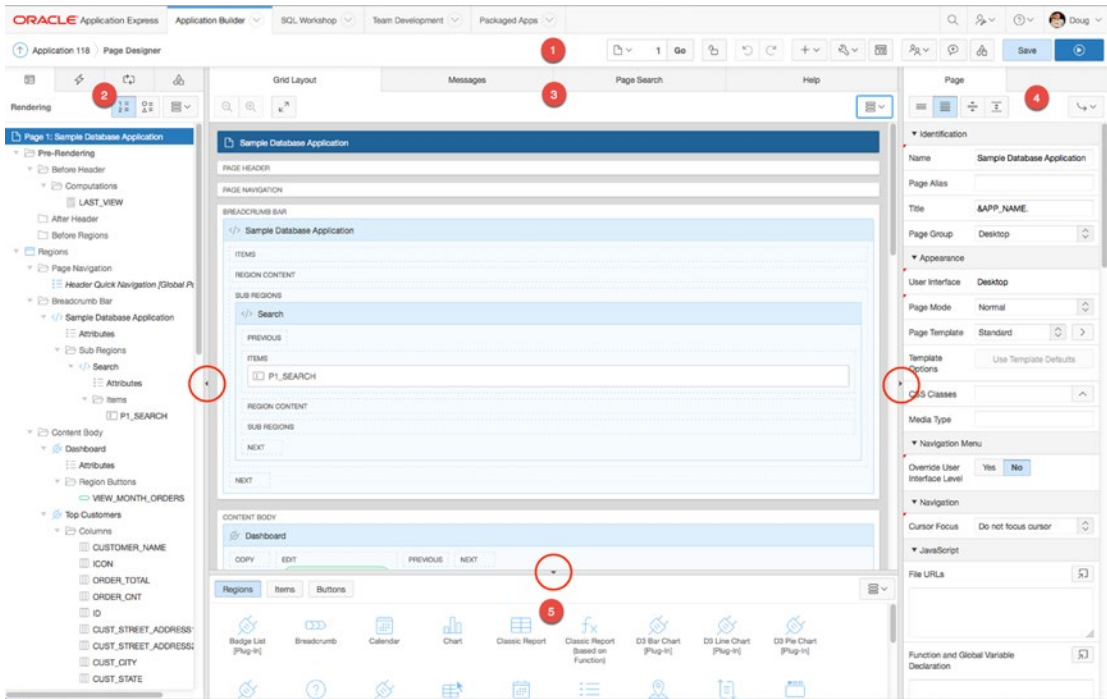
Figure A-1. *The APEX 5.0 Page Designer*

The Page Designer is broken down into five major areas, as indicated by the numbers in Figure A-1:

1. **Page Designer Toolbar** – Appears at the top of the Page Designer, providing access to page-level activities

2. **Tree Pane** – Displays a set of tabs that allow access to Rendering, Dynamic Actions, Processing, and Shared Components

3. **Central Pane** – Provides access to layout, messages, page-level search, and context-sensitive help

4. **Property Editor** – Allows editing of properties for the selected item(s) on the page

5. **Gallery** – Provides access to the different region, item, and button types available within APEX

One thing that you will probably notice is that, on smaller screens such as laptops, the interface can look quite cramped. You can adjust the size of the sections by clicking and dragging on the splitters between the sections, and you can toggle a section's visibility by clicking on the small triangle icon, indicated in Figure A-1. However, if you're like me, you may want to do most of your work on a monitor with a bit more room so you can stretch out the panes to a more comfortable and readable size.

Now, let's explore each of the five main sections in more depth.

Page Designer Toolbar

Figure A-2 shows the Page Designer toolbar, which is designed to give you access to a number of page-level activities.
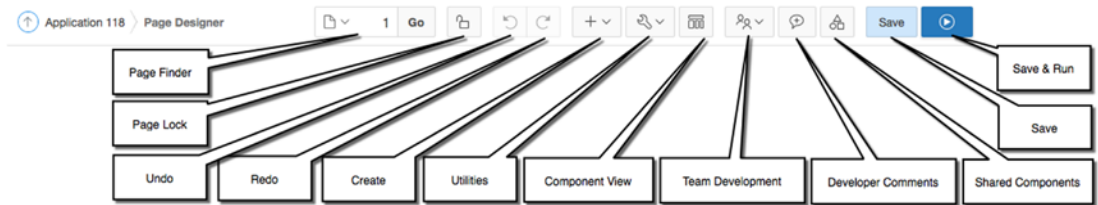
Figure A-2. *The Page Designer toolbar*

At the left, you see the APEX breadcrumbs telling you which application you're currently editing and that you're currently on the Page Designer. Working from left to right from there are the controls. Hovering over each control will present a tooltip with a description and, in some cases, more information about the control.

> ***Page Finder***: Clicking on the pull-down menu opens the Page Finder dialogue, allowing you to select a page to edit. You may also enter the page number directly and click the Go button to navigate to that page. If you attempt to navigate away from the current page, and there are any unsaved changes, APEX will ask if you would like to discard those changes and continue.

> ***Page Locking***: The padlock icon displayed here indicates the locking state for the current page. If the page is unlocked, the padlock icon appears to be opened; if locked, the padlock icon will appear closed. Clicking on the icon will toggle the locking state and assign the current user as the owner of that lock. When locking a page, you will be asked to provide a lock comment that will display to other users who access the page.

> ***Undo***: This rolls back the last change(s) to an item or set of items. Hovering over the icon will indicate the change(s) that will be rolled back.

> ***Redo***: This reapplies the last change(s) to an item or set of items. Hovering over the icon will indicate the change(s) that will be reapplied.

> ***Create***: This pull-down menu provides access so that you can create one of a number of APEX objects. The following options are provided as part of the Create menu:

> > ***Page***: Initiates the **Create a Page** wizard

> > ***Page as Copy***: Initiates the **Copy Page** wizard

> > ***Form Region***: Initiates the **Create Form** wizard

> > ***Report Region***: Initiates the **Create Report Region** wizard

> > ***Page Component***: Opens a help dialogue that outlines the ways page components can be created in APEX 5.0

451

Shared Component: Initiates the **Create Application Component** wizard

Page Group: Navigates to the **Create Page Group** page in APEX

Developer Comment: Opens the Developer Comments dialogue. Here, you can enter and view comments against the current page or a set of pages.

Team Development: Provides a sub-menu allowing the creation of Features, Bugs, and To Dos in the team-development tool.

Utilities: This pull-down menu provides access to a number of page-related utilities. The following options are provided as part of the Utilities menu:

Delete page: Allows you to delete the current page. The dialogue that is displayed upon selecting this option allows you to confirm the deletion and to decide whether or not you wish to cascade that delete to any related list entries, as well as shows you the contents of the page you will be deleting.

Advisor: Opens a new browser window displaying the APEX Advisor and selects the current page for processing. From here you may perform checks on either your application or a specific page, including looking for errors, security issues, QA issues, and other best practices.

Caching: Navigates to the Caching Dashboard for the current page. From here you can manage the page- and region-caching that is active for the current page.

Attribute Dictionary: Navigates to the Attribute Dictionary Dashboard for the current page. From here you can update the current page from the Attribute Dictionary or use the current page to update the Attribute Dictionary.

History: Navigates to a report of historical changes that have been made to the current page and application. The report shows the changes made and the developer who made the changes.

Export: Initiates the Export Page wizard, allowing for the export of a single page of the application as a backup prior to making changes, or to transfer from one instance to another.

Cross-Page Utilities: Navigates to a set of pages that can be used across multiple pages.

Application Utilities: Navigates to a set of application-level utilities.

Page Groups: Navigates to the Page Groups management and assignment page.

Upgrade Application: Navigates to the Application Upgrade report and wizard. Here you can see which elements of an application could potentially benefit from upgrading functionality to the latest provided by APEX 5.0

Component View: Switches from the Page Designer View to the Component View, which has been available since early versions of APEX.

Team Development: This pull-down menu provides access to the features of the Team Development module and will only appear if Team Development is enabled within the workspace. The following options are provided as part of the Team Development menu:

Features: Navigates to the Team Development module and displays a report that is filtered to show incomplete Features for the current page.

To Dos: Navigates to the Team Development module and displays a report filtered to show incomplete To Do items for the current page.

Bugs: Navigates to the Team Development module and displays a report filtered to show incomplete bugs for the current page.

Feedback: Navigates to the Team Development module and displays a report filtered to show unhandled feedback for the current page.

Developer Comments: Initiates a dialogue where developer comments, bugs, or to do items can be created against the current page.

Shared Components: Navigates to the Shared Components page of APEX, providing access to all application-level shared components.

Save: Saves any outstanding changes on the current page

Save & Run: Saves any outstanding changes on the current page and then runs the page.

Tree Pane

The Tree Pane on the left-hand side of the Page Designer gives you access to all of the components of the current page by providing a set of four tabs: Rendering, Dynamic Actions, Processing, and Page Shared Components. Figure A-3 shows the top region of the Tree Pane and indicates the common features.
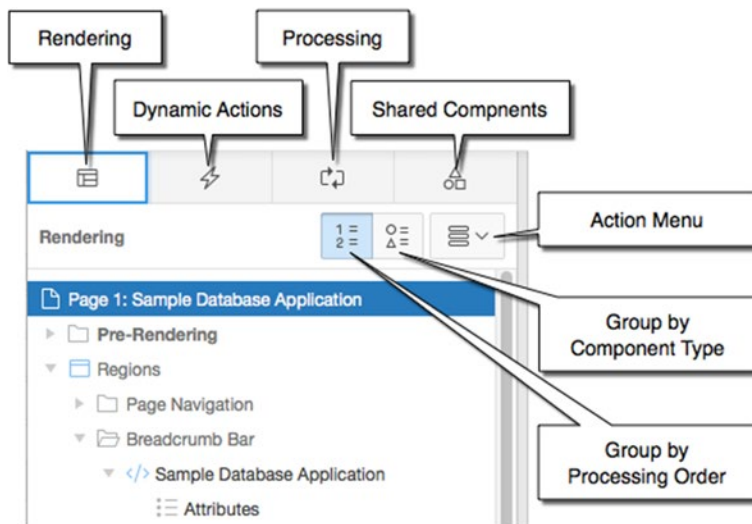
Figure A-3. *The Page Designer's Tree Pane*

Clicking on one of the four tabs across the top will contextually change the tree data displayed in the body of the pane:

> The ***Rendering*** tab changes the tree to display the components related to page rendering, including Regions, Items, Buttons, Logic, and so on.

> The ***Dynamic Actions*** tab changes the tree to display all dynamic actions defined on the current page, regardless of how they are triggered.

> The ***Processing*** tab changes the tree to display all application logic associated with page processing, including Computations, Validations, Processes, and Branches.

> The ***Page Shared Components*** tab changes the tree to display application-level shared components that are associated with this page.

■ **Note**   There is some overlap between the Rendering and the Dynamic Actions tabs in that the Rendering tab will display any Dynamic Action whose triggering element is a rendered component. Don't let this trick you into believing that they are separate Dynamic Actions. They are the same and are displayed in both places for convenience.

When either the Rendering or Processing tabs are active, two extra buttons (as shown in Figure A-3) are displayed that allow you to either *Group by Processing Order* (the default) or *Group by Component Type*.

The Dynamic Action and Page Shared Component tabs are grouped by Event and Component type, respectively, and then ordered by their user-assigned sequence.

The final control, the Action Menu, is a context-sensitive menu that mirrors the context menus you see when right-clicking the components in the tree. Both menu types provide a multitude of functionality depending upon which tab is current as well as which node you have selected in the tree.

For instance, if you're currently on the Rendering tab and have a Region selected in the tree, clicking either the Action Menu or right-clicking on the region name in the tree will produce a context menu of available options.

Hovering over any node in the tree, regardless of which tab you're on, will display a tooltip containing basic information about the component.

You may also use the tree to reorder components using Drag & Drop. Reordering the items within a form is as easy as clicking and dragging the items into the order you want them. While dragging an item, a yellow helper node appears in the tree indicating a legal drop position. Note, however, that this only changes their order and doesn't affect any of the items' layout properties.

Central Pane

The Central Pane is broken down into four separate tabs, as indicated in Figure A-4: Grid Layout, Messages, Page Search, and Help. Each of these tabs merits its own detailed explanation.
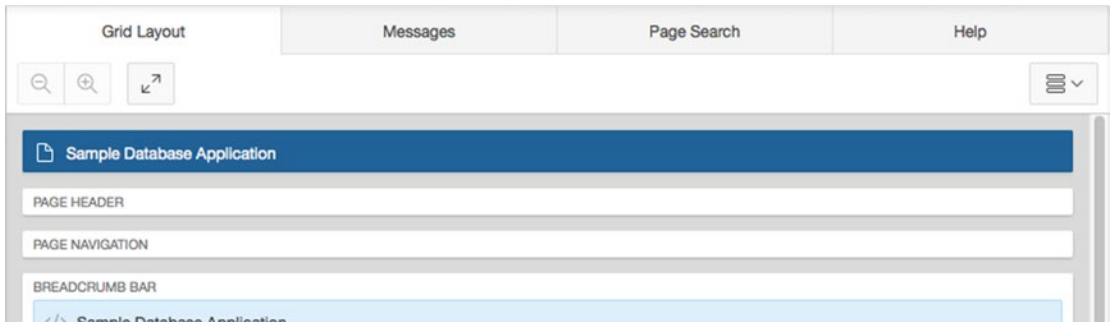
Figure A-4. *Tabs of the Central Pane*

Grid Layout

The Grid Layout tab, as shown in Figure A-5, provides a visual representation of the components that will be displayed on the page. While this is definitely not a full WYSIWYG representation of the final rendering, it does a very good job of allowing the developer to understand how components will be rendered on the screen.
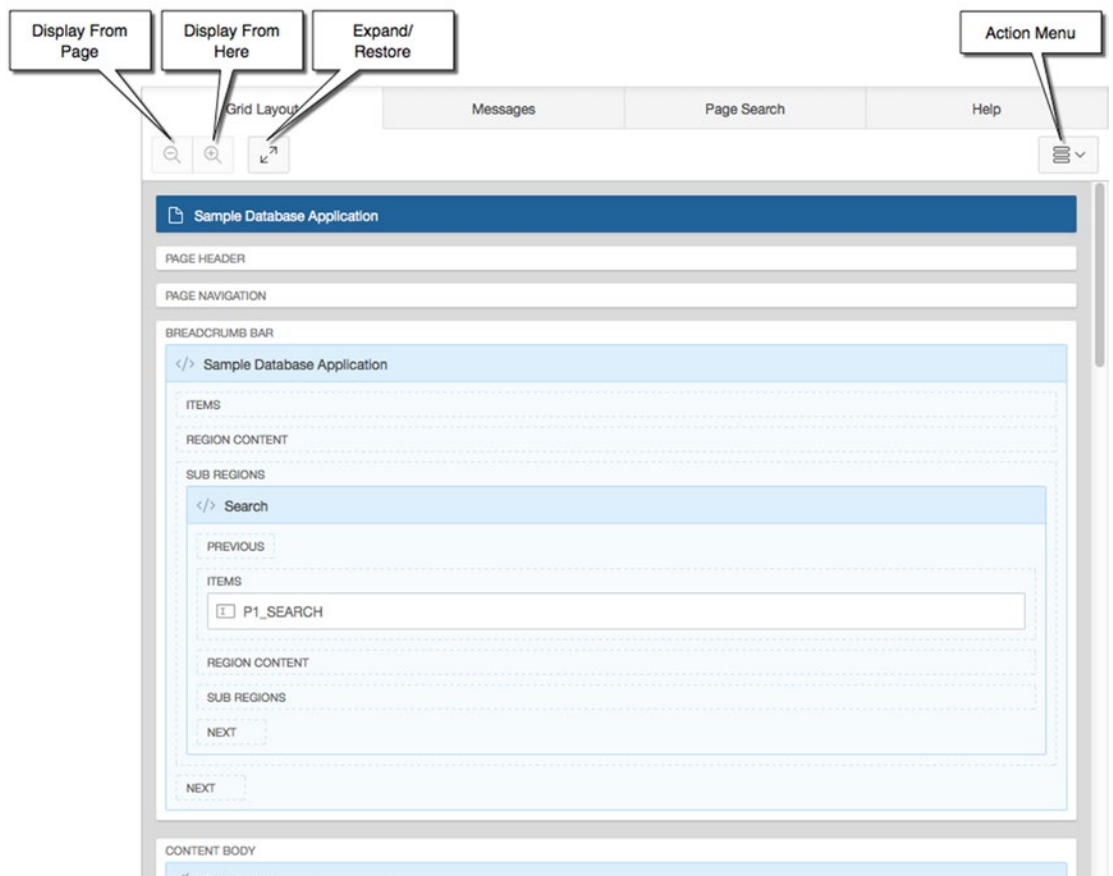
Figure A-5. *Grid Layout and associated controls*

As mentioned earlier, you can use the region splitters to resize the amount of space each panel takes up on the screen, or you can, by clicking the arrow on the splitter, hide a region completely. There are times, however, that you may want to expand the Grid Layout to take up the entire canvas. Clicking the *Expand/Restore* button will initially expand and then reinstate the Grid Layout.

Alternatively, on particularly large or busy pages, you may want to focus on the layout of a specific region. Clicking on the region in the Grid Layout and then clicking the *Display From Here* button will hide all other regions on the page, outside of the selected region, from view. Clicking on the *Display From Page* button will restore the Grid Layout to its default view, displaying the entire page's visible components.

Visible components on the page can be moved around using *Drag & Drop*. New components can also be added to the page by dragging and dropping them onto the Grid Layout canvas from the *Gallery* (see Figure A-6). When dragging items into position, the areas where the object may be dropped will have a yellow background. As you hover over a placement area, a grey box will appear within the yellow background (as shown in Figure A-6) indicating where the object will be dropped.
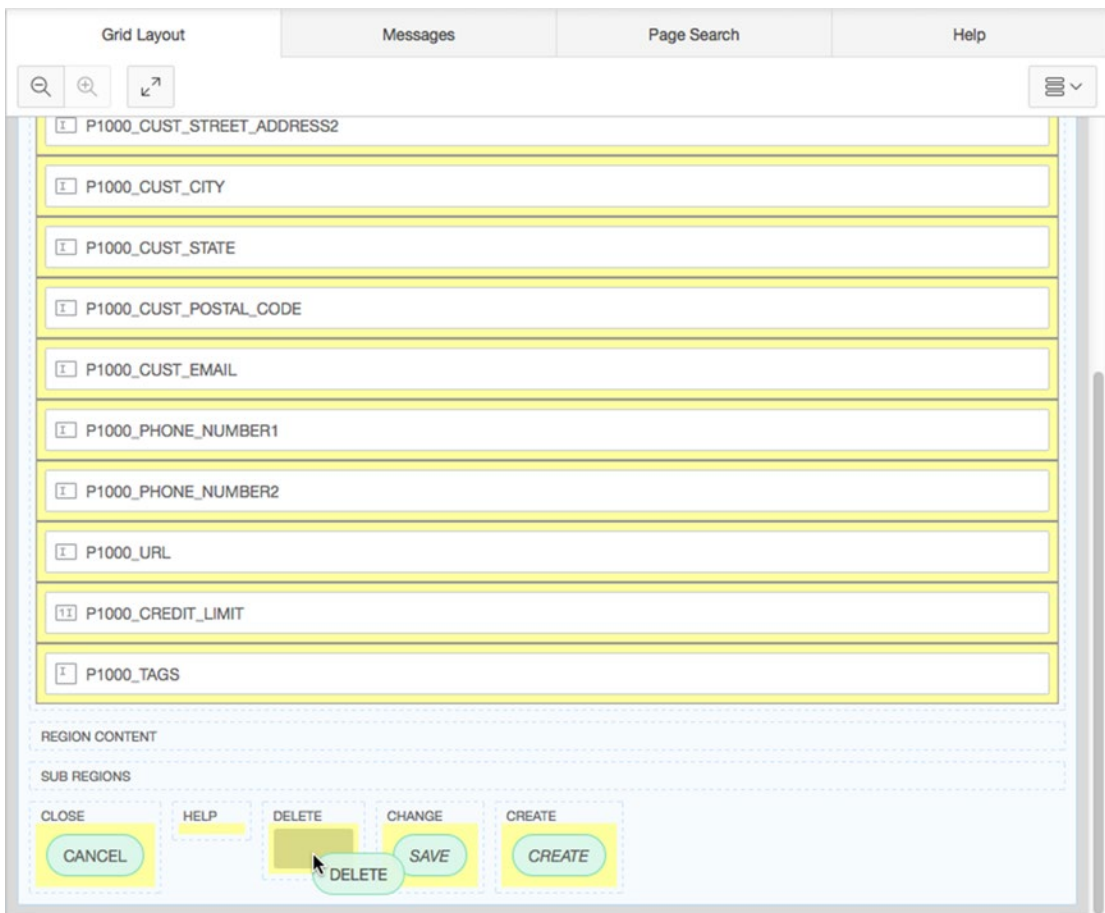
Figure A-6. *Using Drag & Drop to place a button*

As with the Tree Pane, the Grid Layout contains an Action Menu and context menus available for all components. By either selecting the component in the Grid Layout and right-clicking or clicking the Actions Menu you will be presented with the actions available for that component.

There are a set of options in the Grid Layout's Action Menu and context menus that control what is shown in the Grid Layout:

Hide Empty Legacy Positions: When this option is selected (which is the default), empty template positions, which are deemed to be "legacy," are hidden from the Grid Layout. Legacy positions are considered to be those that relate to older themes and, while not deprecated, are discouraged from being used moving forward. Even if this option is selected, if the position contains a component, it will be shown in the Grid Layout.

Hide Empty Positions: When this option is selected, any position that does not contain a component will be hidden from view. This allows the developer to focus only on those positions that currently contain items.

Hide Global Page Components: When this option is selected (default), the Grid Layout hides any components that are placed on the current interface's global page. This allows the developer to focus only on the components that are defined on the current page.

Hide Buttons: When this option is selected, all region areas that are considered button containers and any assigned buttons are hidden from view. This allows the developer to focus on the placement of items within a region.

Hide Items: When this option is selected, all region areas that are considered item containers and any assigned items are hidden from view. This allows the developer to focus on placement of buttons within a region.

Again, similar to the Tree Pane, tooltips are available in the Grid Layout, providing basic information about the component over which you're hovering.

■ **Note**   Hidden items will not display in the Grid Layout but will appear in the Rendering section of the Tree Pane.

Messages

As you develop your page by placing components and editing their attributes, you will inevitably see a component highlighted in either red or yellow, and the Messages tab will be badged to indicate the number of messages available, as shown in Figure A-7.
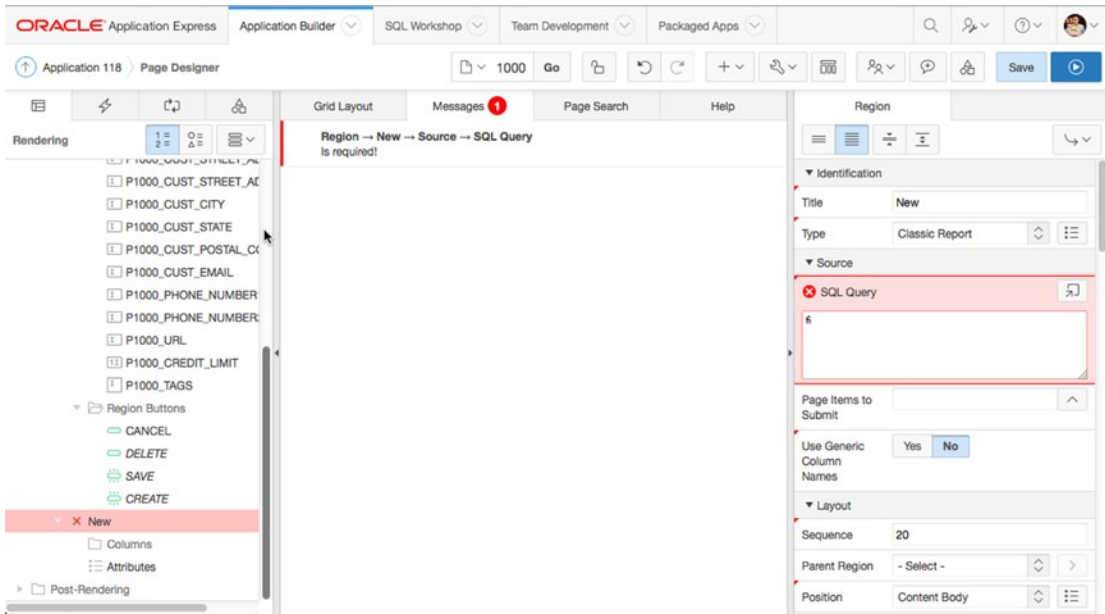
Figure A-7. *The Messages tab showing a message and the associated component and attribute*

Messages come in two types:

> ***Errors***: Indicated in red, these messages must be addressed before the page can be saved. Clicking on the text of the error in the Messages tab will display the attribute in the Properties Pane associated with the error.

> ***Warnings***: Indicated in yellow, these messages are there to warn you of potential problems. You are still able to save the page without addressing the warning, but the page may not act properly until the warning is fully addressed. Clicking on the text of the warning in the messages tab will display the attribute in the Properties Pane associated with the warning.

Page Search

Figure A-8 shows the Page Search tab, which allows you to search through the metadata of all page components.

| Grid Layout | Messages | Page Search | Help |

Search P1_SEARCH

☐ Match Case

☐ Regular Expression

Clear

🔍 Page Item → P1_SEARCH → Identification → Name
P1_SEARCH

🔍 Action → Set Focus → Affected Elements → Item(s)
P1_SEARCH

🔍 Branch → Go To Page 30 → Behavior → Target
f?p=&APP_ID.:30:&SESSION.::&DEBUG.:1,30:P30_SEARCH:&P1_SEARCH.
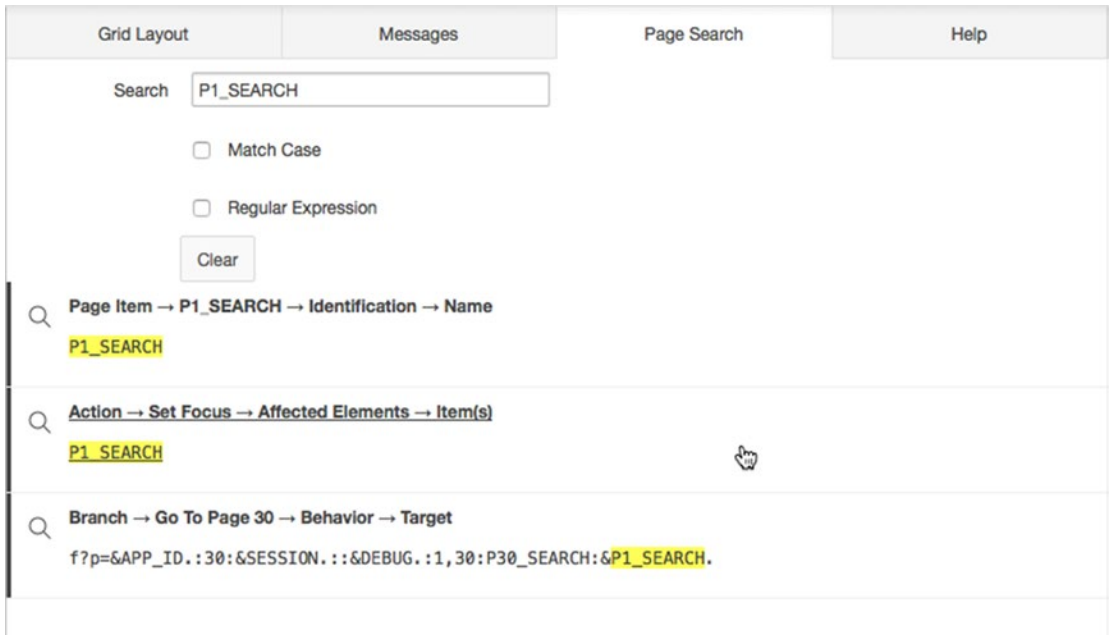
Figure A-8. *The Page Search tab showing search results*

Entering any text into the search field and pressing RETURN will search the metadata of your application for the search term entered and display any references found, highlighting your search term. Clicking on any of the items found with the search will navigate to that component within the page designer.

Using the ***Match Case*** option will require the search to match the case of the entered search term exactly. Using the ***Regular Expression*** option will treat the search term as a regular expression string.

The ***Clear*** button will clear the search from the entire Page Search form as well as any results listed below it.

Help

Every attribute within the Property Editor has associated help available to the developer. To view the help, select a component in either the Tree Pane or the Grid Layout and then select an attribute. At that point, switching to the Help tab will display the help for the selected attribute. Figure A-9 shows the help text available for the Subtype attribute of the P1_SEARCH text field.
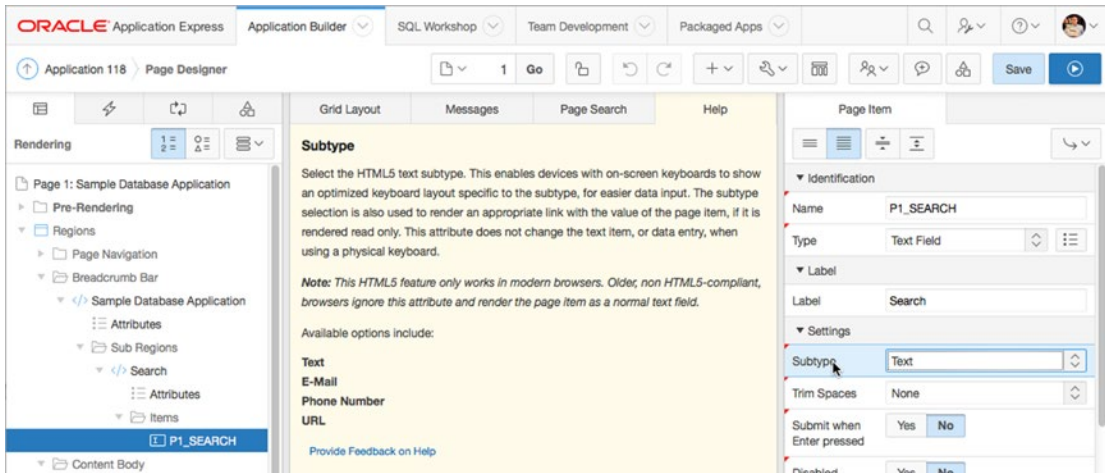
Figure A-9. *Viewing help for the P1_SEARCH Subtype attribute*

This context-sensitive help will assist developers in quickly coming up to speed on the many new attributes that have been introduced in APEX 5.0

At the bottom of each help section is a ***Provide Feedback on Help*** link. Clicking on the link will open a new browser window and navigate to an APEX application that allows you to provide feedback on the help text for the attribute currently in context. The feedback will be used by the Oracle APEX development team to help refine the help text in future releases.

Property Editor

The Property Editor on the right-hand side of the Page Designer displays all of the attributes for the component(s) currently selected in either the Tree Pane or the Central Pane. Figure A-10 shows the Property Editor pane and outlines the related controls.
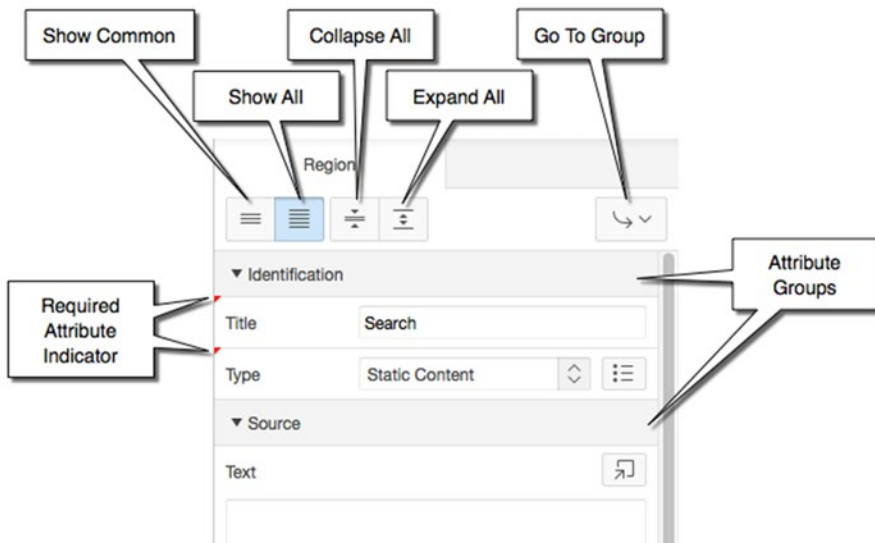
Figure A-10. *The Page Designer's Property Editor pane*

As you select components in the Tree Pane or the Grid Layout, the Property Editor will update to show the attributes for the currently selected component(s). Attributes are organized into ***Attribute Groups*** and can be expanded and collapsed by clicking on the triangle icon beside the group name. The ***Required Attribute Indicator*** takes the form of a red triangle in the upper-left corner near the attributes label.

The amount and type of information that is displayed in the Property Editor can be controlled by the controls at the top of the pane:

> ***Show Common***: In cases where only one component is selected, choosing this option will show only the attributes whose values differ from those of the default. If more than one component is selected, this option will also show the attributes that have been commonly edited across the selected components.

> ***Show All***: In cases where only one component is selected, choosing this option will show all attributes for the selected component. If more than one component is selected, this option will also show all common attributes for the selected components.

> ***Collapse All***: Collapses all of the Attribute Groups so that only the headings are shown.

> ***Expand All***: Expands all of the Attribute Groups so that all appropriate attributes (as controlled by Show Common and Show All) are displayed.

> ***Go To Group***: Places focus on the first attribute of the group selected from the pull-down menu.

As you edit the various attributes of APEX components, it is likely that you'll be familiar and comfortable with most of the controls you are presented with. In the next sections, we'll go over a few control types whose function might not be immediately apparent.

461

Quick Picks

The Quick Pick control appears to the left of attribute select lists, as shown in Figure A-11. Clicking on the Quick Pick icon will produce a short list of what are considered the "most common" options of the full select list.
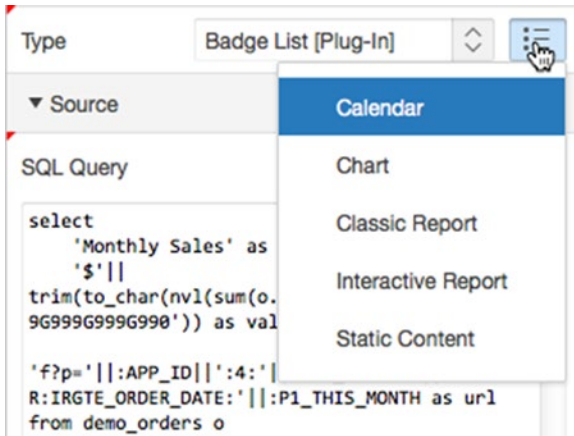
Figure A-11. *The Quick Pick icon being clicked for a select list*

Go To

The Go To control appears when an attribute references another component on the page. By clicking the icon shown in Figure A-12, the Page Designer's focus is set to the component indicated by the attribute. This provides a quick way to navigate directly to the referenced component.
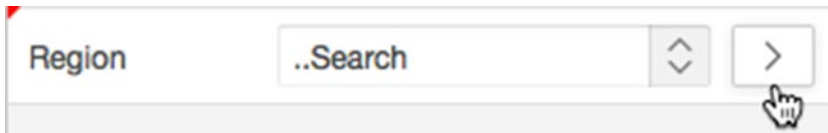
Figure A-12. *The Go To icon associated with the Region attribute for a text field*

Options Dialogue Button

There are a number of attributes that require a more complex pop-up dialogue to set their values. In these cases, APEX provides an Options Dialogue Button, as shown for both the Template Options and the Target Page in Figure A-13. Clicking on the grey box containing the text produces a dialogue that provides options for the attribute in question. Figure A-14 is an example of the Template Options Dialogue for a button.

Figure A-13. Template Options and Target attributes with Options Dialogue Buttons
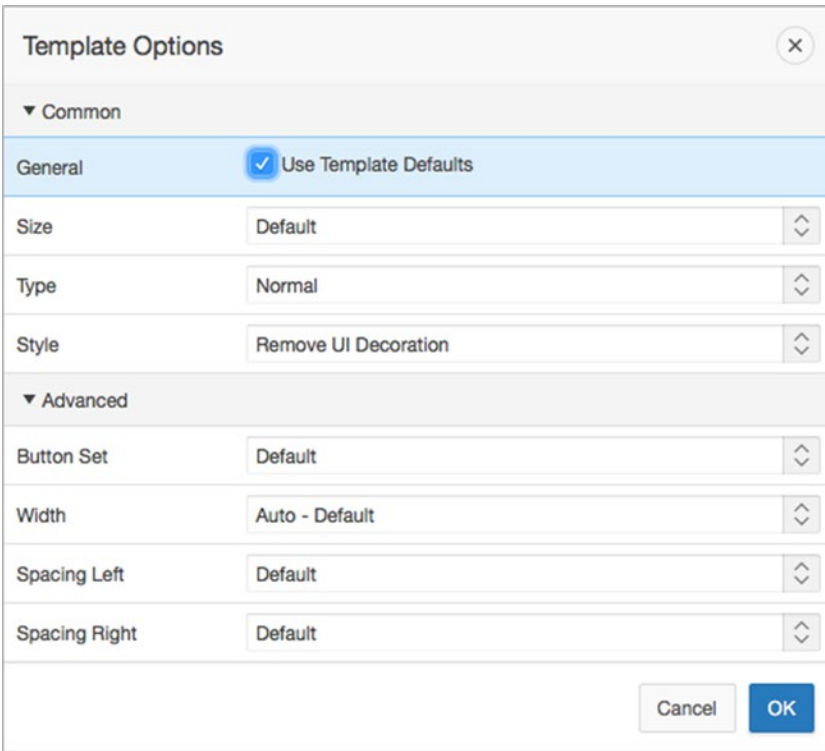
Figure A-14. *Example dialogue for a button's Template Options*

Code Editor

For attributes requiring SQL, PL/SQL, or large amounts of static text, APEX provides a Code Editor control. As shown in Figure A-15, the control shows the code in a relatively small text area but provides a button to expand the text area into the full Code Editor, as shown in Figure A-16.
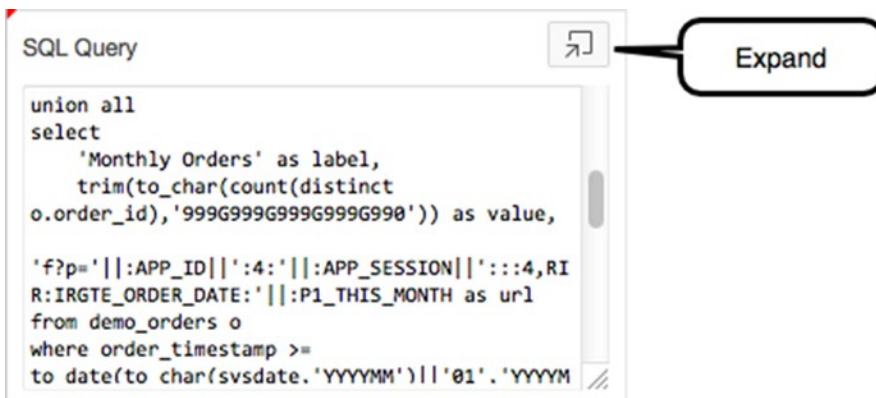
Figure A-15. *The SQL Query attribute of a report with the Code Editor button in the upper right*

```
Code Editor - SQL Query                                                    ×

   ↺   ↻    Q   ↔    T   A··   ⊘                                          ⚙ ⌄

 1  select
 2       'Monthly Sales' as label,
 3       '$'|| trim(to_char(nvl(sum(o.order_total),0),'999G999G999G999G990')) as value,
 4       'f?p='||:APP_ID||':4:'||:APP_SESSION||':::4,RIR:IRGTE_ORDER_DATE:'||:P1_THIS_MONTH as url
 5  from demo_orders o
 6  where order_timestamp >= to_date(to_char(sysdate,'YYYYMM')||'01','YYYYMMDD')
 7  union all
 8  select
 9       'Monthly Orders' as label,
10       trim(to_char(count(distinct o.order_id),'999G999G999G999G990')) as value,
11       'f?p='||:APP_ID||':4:'||:APP_SESSION||':::4,RIR:IRGTE_ORDER_DATE:'||:P1_THIS_MONTH as url
12  from demo_orders o
13  where order_timestamp >= to_date(to_char(sysdate,'YYYYMM')||'01','YYYYMMDD')
14  union all
15  select 'Total Products' as label,
16          trim(to_char(count(distinct p.product_name),'999G999G999G999G990')) as value,
17          'f?p='||:APP_ID||':3:'||:APP_SESSION||':::' as url
18  from demo_product_info p
19  union all
20  select 'Total Customers' as label,
21          trim(to_char(count(*),'999G999G999G999G990')) as value,
22          'f?p='||:APP_ID||':2:'||:APP_SESSION||':::' as url
23  from DEMO_CUSTOMERS

                                                            Cancel      OK
```

Figure A-16. *Example Code Editor dialogue*

The Code Editor contains a toolbar across the top of the dialogue. Figure A-17 shows the toolbar and indicates the function of each of its icons.
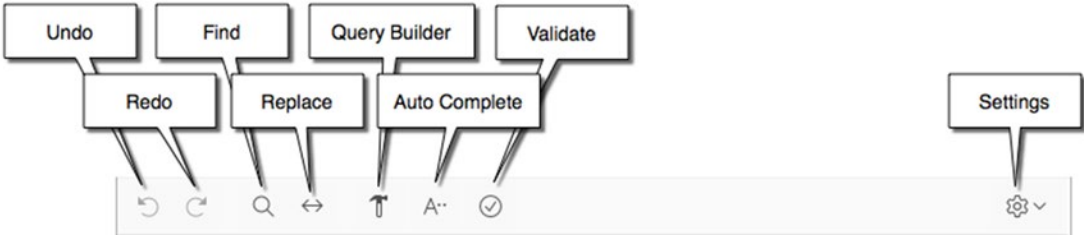
Figure A-17. *The Code Editor toolbar with icon explanations*

Most of the icons are fairly self-explanatory, but for completeness their functionality is listed below:

Undo: Rolls back the last set of changes to the text in the Code Editor dialogue.

Redo: Reapplies the last change that was rolled back using the Undo button.

Find: Displays a region below the toolbar allowing the developer to search the contents of the Code Editor dialogue. Search options include Match Case and Regular Expression.

Replace: Displays a region below the toolbar allowing the developer to sear the contents of the Code Editor and replace occurrences of the found search string with another string. Search options include Match Case and Regular Expression. Replace options include Replace, Replace All, and Skip.

Query Builder: Opens a new window with the Drag & Drop Query builder (See Chapter 4). It only displays for Code Editor dialogues expecting a SQL query.

Auto Complete: Provides context-sensitive Auto Complete functionality while editing the text in the Code Editor. When editing SQL or PL/SQL, Auto Complete provides autocomplete for database objects in the current "parse as" schema as well as Oracle-defined functions and reserved words. When editing HTML text, Auto Complete provides assistance with HTML tags.

Validate: When editing SQL or PL/SQL, Validate will parse the code provided for syntax errors. If a syntax error is encountered, the error is indicated by an error message inline with the code prior to the line with the error. If no errors are found, a "Validation Successful" message is displayed at the top of the dialogue.

Settings: Provides a set of dialogue-specific settings that will be remembered across user sessions for each specific user:

> **Use Plain Text Editor**: Switches from the syntax-highlighting editor to a plain text editor.
>
> **Tab Inserts Spaces**: When selected and the Tab key is pressed, the editor will place a fixed number of spaces instead of a tab character. When deselected, a tab character will be used.
>
> **Tab Size**: Number of spaced to be used to replace a tab character when the previous option is enabled.
>
> **Indent Size**: Sets the auto-indent size for languages (such as JavaScript) that support auto indenting.
>
> **Themes**: Applies one of a number of different visual themes to the Code Editor. The new theme will not be visible until the setting is OK'ed and the Code Editor selected again.
>
> **Show Line Numbers**: When selected, a gutter with line numbers will be added to the left-hand side of the code.
>
> **Show Ruler**: When selected, a dotted line appears at the eighty-character mark within the Code Editor.

Gallery

The Gallery Pane displays directly below the Central Pane and provides a palette of components and controls that can be used to build the page in the Grid Layout. The pane has three types of components selectable via the buttons at the top left, as shown in Figure A-18.
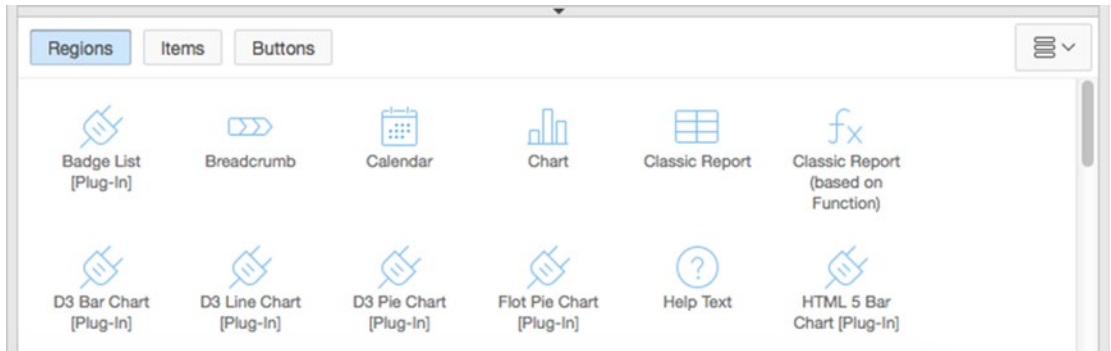
Figure A-18. *The Gallery Pane showing some of the available region types*

By default, only controls and components that are supported in the current user interface are displayed. Clicking on the Action Menu in the upper right, you can opt to show unsupported components that are to be used "at your own risk." Some of these components are considered experimental and may not work in all browsers.

Each component can be placed into the Grid Layout by simply dragging and dropping the component into the appropriate area. When dragging items into position, a yellow background will indicate the areas where the object may be dropped. As you hover over a placement area, a grey box will appear within the yellow background (refer to Figure A-6) indicating where the object will be dropped.

You may also right-click on a component and use the context menu to place an item. The ***Add To*** context menu option allows you to select where to place the item using a layered menu representation of the page structure.

Keyboard Shortcuts

Another benefit of the new Page Designer is the introduction of a set of keyboard shortcuts to perform common tasks. Table A-1 shows the keyboard shortcuts for both PC/Linux and Macintosh systems.

Table A-1. *A List of Keyboard Shortcuts Available in the Page Designer*

Function	PC/Linux	Macintosh
Save	Ctrl+Alt+S	Ctrl+Option+S
Save and Run Page	Ctrl+Alt+R	Ctrl+Option+R
Undo	Ctrl+Z	Ctrl+Z
Redo	Ctrl+Y	Ctrl+Y
Go to Rendering	Alt+1	Option+1
Go to Dynamic Actions	Alt+2	Option+2
Go to Processing	Alt+3	Option+3
Go to Page Shared Components	Alt+4	Option+4
Go to Grid Layout	Alt+5	Option+5
Go to Property Editor	Alt+6	Option+6
Go to Gallery Regions	Alt+7	Option+7
Go to Gallery Items	Alt+8	Option+8
Go to Gallery Buttons	Alt+9	Option+9
Display From Here	Ctrl+Alt+D	Ctrl+Option+D
Display From Page	Ctrl+Alt+T	Ctrl+Option+T
Restore/Expand	Alt+F11	Option+F11
Toggle Hide Empty Positions	Ctrl+Alt+E	Ctrl+Option+E
Go to Help	Alt+F1	Option+F1
Go to Messages	Ctrl+F1	Ctrl+F1
Page Search	Ctrl+Alt+F	Ctrl+Option+F
Keyboard Shortcuts	Alt+Shift+F1	Option+Shift+F1

■ **Note** Some platforms (especially Mac) may already have keyboard mappings in place that might interfere with the function of the key combinations outlined in this table. If any of the functionality doesn't work as expected, check to see if there are any conflicting shortcuts already in place at the Operating System level.

Summary

While there is a lot to learn about the new Page Designer, after spending some time getting familiar with its layout and functionality I believe that developing APEX applications will become a much more efficient endeavor. I urge you to spend some time familiarizing yourself with the contents of this appendix before you delve deeply into the development chapters of this book. Remember, you can always come back and reference this appendix if you ever get lost.

Index

■ C

■ H

■ I, J, K

■ W, X, Y, Z

IOUG
independent oracle users group

For the Complete Technology & Database Professional

IOUG represents the **voice of Oracle technology and database professionals** - empowering you to be **more productive in your business** and career by **delivering education,** sharing **best practices** and providing technology direction and **networking opportunities.**

Context, Not Just Content

IOUG is dedicated to helping our members become an #IOUGenius by staying on the cutting-edge of Oracle technologies and industry issues through practical content, user-focused education, and invaluable networking and leadership opportunities:

- *SELECT Journal* is our quarterly publication that provides in-depth, peer-reviewed articles on industry news and best practices in Oracle technology

- Our #IOUGenius blog highlights a featured weekly topic and provides **content driven by Oracle professionals and the IOUG community**

- Special Interest Groups provide you the chance to collaborate with peers on the specific issues that matter to you and even take on leadership roles outside of your organization

- COLLABORATE is our once-a-year opportunity to connect with the members of not one, but three, Oracle users groups (IOUG, OAUG and Quest) as well as with the top names and faces in the Oracle community.

Who we are...

... more than 20,000 database professionals, developers, application and infrastructure architects, business intelligence specialists and IT managers

... a community of users that share experiences and knowledge on issues and technologies that matter to you and your organization

Interested? Join IOUG's community of Oracle technology and database professionals at **www.ioug.org/Join.**