to fit or look as good. In this case, just change the `min-width` value for a particular breakpoint and add your styles inside that media query (for instance, change `550px` to `650px`). In fact, you probably won't need all of the media queries listed in the code starting on page 507, so feel free to remove them from the style sheet.

The best way to understand mobile-first design is to see it in action, which you'll get a chance to do in the tutorial coming up next.

Tutorial: Using a Grid System

This tutorial shows you how to apply a grid system to a web page using Skeleton. You'll start with some basic HTML, add the required CSS files, add and structure your HTML to create grids, and finally, add custom CSS rules to make the page look great—including on mobile devices.

To get started, download the tutorial files located on this book's companion website at *https://github.com/sawmac/css_mm_4e*.

Adding a Grid

To get started, you'll attach the required CSS files and add some basic HTML to create a grid container with rows and columns.

1. **In your text editor, open the file *16→index.html*.**

   This is a basic HTML file. It has a head and body, but no HTML in the body yet. Before you add HTML, however, add links to a few CSS files. There's already a link to a Google Font (page 140), but you also need links to the Skeleton files.

2. **On the empty line just before the closing `</head>` tag, add the following three lines.**

   ```
   <link rel="stylesheet" href="css/normalize.css">
   <link rel="stylesheet" href="css/skeleton.css">
   <link rel="stylesheet" href="css/custom.css">
   ```

   The first line of code loads the *normalize.css* file. *normalize.css* is a very popular CSS reset file (page 579) and is used in many projects. The second file is the basic *skeleton.css* file, which includes the CSS for the grid system. The final CSS file, *custom.css*, is mostly empty; it has a few media queries to add styles for different screen widths. You'll use this file to add styles to customize the look of the page.

   Next, you'll add a Skeleton container.

3. **Just below the HTML comment `<!-- top section -->` add the following HTML:**

   ```
   <!-- top section -->
   <div class="container">

   </div>
   ```

As explained on page 495, Skeleton uses the class name `container` to define the div that holds the rows you'll add to a page. You can have any number of containers on a page, or use just a single container div for all of your rows. As you'll see in a minute, there's a benefit of having more than one container.

Time to add your first row.

4. **Inside the `<div>` you just added, insert two divs to create two rows (additions are in bold):**

```
<!-- top section -->
<div class="container">
  <div class="row">

  </div>
  <div class="row">

  </div>
</div>
```

These divs create two rows. To the first row, you'll add two columns—one for the site name and the other for navigation.

5. **Inside the first `<div>` with the class `row`, type the following HTML:**

```
<div class="four columns">
  <p class="logo">My Company</p>
</div>
<div class="eight columns nav">

</div>
```

The Skeleton grid system divides a row into 12 units. You can divide those units to make individual columns, so here you have two columns, one 4 units and the other 8 units. Visually, you have one column that's 1/3, or around 33%, of the container width; the second is 2/3, or around 66%, of the container width.

The second column is wider, because it's going to hold a navigation bar. In fact, notice that there's a class name—nav—added to this tag. That's not a requirement of the Skeleton framework; you're adding it now so you can use it later to style the navigation bar.

6. **Open the file *01-nav.html*. Copy the HTML inside it and paste it into the 9-unit-wide div so the HTML looks like this (additions in bold):**

```
<div class="eight columns nav">
  <a href="#">Home</a>
  <a href="#">Our Clients</a>
  <a href="#">About Us</a>
  <a href="#">Careers</a>
</div>
```

These are just simple links and don't look like much. However, Skeleton includes some very fine-looking styles to format any element as a button. You simply add a class name to the <a> tags.

7. **Add a class attribute to each link, and assign a** `button` **class, like this:**

```
<div class="eight columns nav">
  <a class="button" href="#">Home</a>
  <a class="button"href="#">Our Clients</a>
  <a class="button" href="#">About Us</a>
  <a class="button" href="#">Careers</a>
</div>
```

Skeleton also includes a special class named `button-primary` that assigns a special look to a button. It's a great way to make the link for the current page stand out from the others, so visitors can tell what page they're on. The page you're working on is the home page, so you'll add that special class to the Home link.

8. **Add the class name** `button-primary` **to the first** <a> **tag:**

```
<div class="eight columns nav">
  <a class="button button-primary" href="#">Home</a>
  <a class="button" href="#">Our Clients</a>
  <a class="button" href="#">About Us</a>
  <a class="button" href="#">Careers</a>
</div>
```

There's one last row that needs content. This one is simple, because you want just a single row that spans the entire width of the container.

9. **Open the file *02-action.html*. Copy the HTML, return to the *index.html* file, and paste it inside the second row div inside the container div.** The final HTML of the container div will look like this (additions from this step are in bold):

```
<!-- top section -->
<div class="container">
  <div class="row">
    <div class="four columns">
      <p class="logo">My Company</p>
    </div>
    <div class="eight columns nav">
      <a class="button button-primary" href="#">Home</a>
      <a class="button" href="#">Our Clients</a>
      <a class="button" href="#">About Us</a>
      <a class="button" href="#">Careers</a>
    </div>
  </div>
  <div class="row">
    <h1>We do great things</h1>
    <h2>Donec pulvinar ullamcorper metus</h2>
```

```
      <a href="#" class="button button-primary">Learn More</a>
    </div>
  </div>
```

The content in this row will span the entire width of the container—in other words, there's only a single column. In this case you don't add any additional divs; you just put the HTML inside the row div.

10. **Add a class, `action`, to the last row:**

```
<div class="row action">
  <h1>We do great things</h1>
  <h2>Donec pulvinar ullamcorper metus</h2>
  <a href="#" class="button button-primary">Learn More</a>
</div>
```

You'll use that class name later when you style this area of the page. The name, `action`, doesn't have anything to do with Skeleton—it's just a class name you can use can style this section of the page.

11. **Save the *index.html* file and preview it in a web browser.**

The page should look like Figure 16-6. At this point it doesn't look like a marvelously organized grid layout. But it will. Let's add another row and three columns this time. To keep your fingers from going numb, we'll supply you with the basic HTML for the main content on this page.

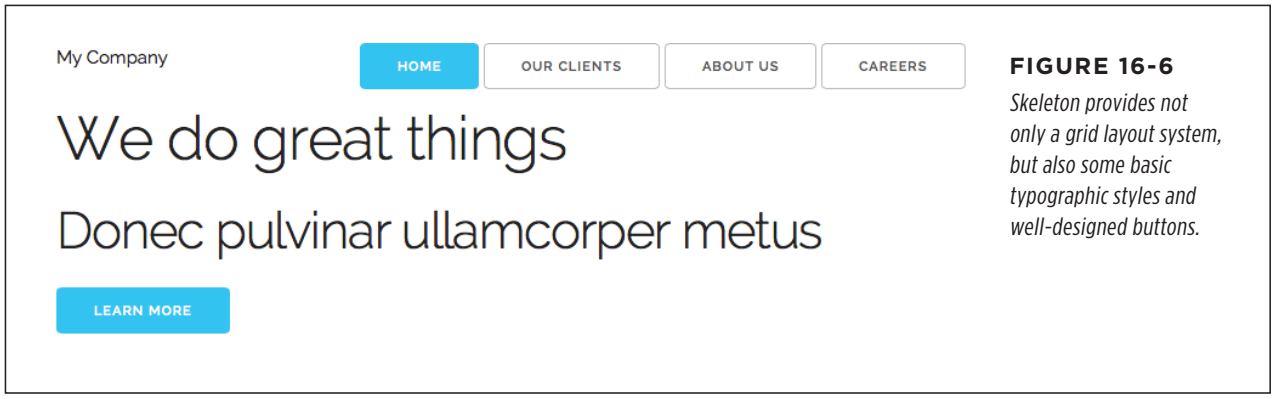

FIGURE 16-6
Skeleton provides not only a grid layout system, but also some basic typographic styles and well-designed buttons.

12. **Open the file *03-info.html*. Copy the HTML and, in the *index.html* file, paste it just below the comment `<!-- info section -->`.**

This is just a basic set of nested divs. There's one `<div>` that represents a container for rows, and two `<div>` tags inside that represent rows. In the first row there's only an `<h2>` tag, which will be a single column that spans the entire container. In the second row are three additional divs, which will be three columns.

The HTML doesn't yet have any of Skeleton's class names—the secret sauce that applies the grid layout—so you'll add those next.

13. **Add** `class="container"` **to the first** `<div>` **in the HTML you just pasted into the page.**

```
<!-- info section -->
<div class="container">
```

This applies Skeleton's container style to the tag, creating a place to insert rows. Next you'll add some rows.

14. **Add** `class="row"` **to the next two divs inside the container, like this (additions in bold):**

```
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
  </div>
  <div class="row">
```

Now that you've created the rows, you need to create three columns in that second row. You can do so by adding a class attribute with two class names: The first name is the number of units the column should take up, and the second is the name `columns`.

15. **Add** `class="four columns"` **to each of the** `<div>` **tags inside the second row within the "info" section of this page. While you're at it, add the button class to the three links in each column.** The finished HTML for the "info" section should look like this:

```
<!-- info section -->
<div class="container">
  <div class="row">
    <h2>Graeco feugiat intellegam at vix</h2>
  </div>
  <div class="row">
    <div class="four columns">
      <p>Mei te possit instructior…</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
    <div class="four columns">
      <p>Mei te possit instructior…</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
    <div class="four columns">
      <p>Mei te possit instructior..</p>
      <p><a href="#" class="button">More info</a></p>
    </div>
  </div>
</div>
```

(The HTML shown here has been condensed by not including all of the Latin gibberish in each column.) Finally, you'll add a footer to the page.

16. **Open the file *04-footer.html*. Copy the HTML and, in the *index.html* file, paste it just below the comment** <!-- footer -->**.**

    This HTML contains the proper Skeleton classes already in place (by now you probably know how Skeleton works, and don't need any more practice typing it all out). The HTML you've just added inserts another container with one additional row and two columns—one 8 units wide and the second 4 units wide.

17. **Save the *index.html* file and preview it in a browser.**

    The page should look like Figure 16-7. The design is coming together: There are two columns in the header (the logo and navigation), three columns of main text, and two columns in the footer.

    If you resize your browser window as thin as it will go, you'll notice that the columns eventually collapse and stack on top of each other. You're watching the responsive part of the Skeleton framework in action. It uses media queries, just like the ones you read about in the previous chapter, to create a single-column design for mobile devices.

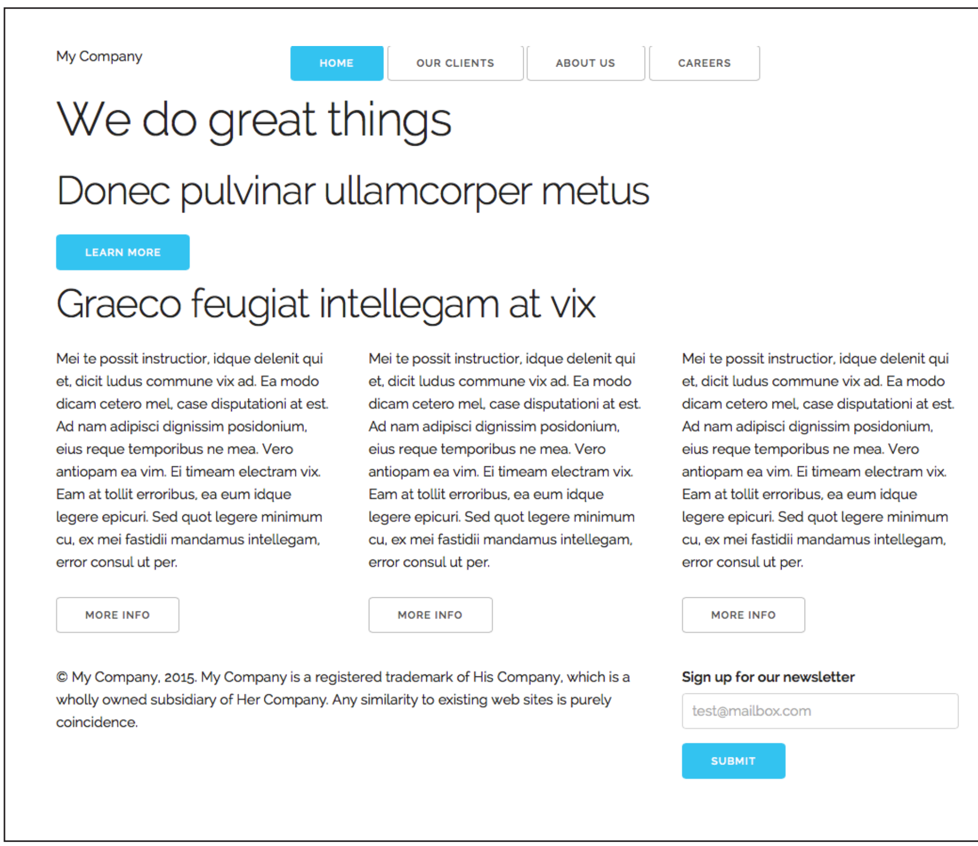

FIGURE 16-7

With just some well-structured HTML, a handful of class names, and the Skeleton CSS file, creating multicolumn layouts is a breeze.

Adding Style

Skeleton takes your designs a long way toward an organized visual presentation, but it's rather plain. To add skin and clothes to this structure, you need to add your own CSS. In this section of the tutorial, you'll add a few design touches.

1. **In your text editor, open the file *16→css→custom.css*.**

   This is a mostly empty CSS file. It only includes media queries set to different breakpoints—the screen widths at which you might want to change the size and look of page elements. For example, on a phone you'll probably want to decrease the size of headlines and eliminate excessive margins and padding.

   You'll start by seeing what happens if you add a background color to a container.

2. **On the empty line following the** `/* Mobile first queries */` **comment, add the following style:**

```
/* Mobile first queries */
/* applies to ALL widths */
.container {
  background-color: pink;
}
```

   Remember that your page has several `<div>` tags with the class `container`. These are the tags that contain rows.

3. **Save the *custom.css* file. Preview the *index.html* file in a browser.**

   The page should look like Figure 16-8. Notice that the pink color is constrained to the width of the content, and is centered in the browser window. If you look back at the final design in Figure 16-4, you'll see that you want a photo in the background of the header area, and that photo should extend the entire width of the browser window. Skeleton containers don't do that, so you need to add another div that will span the entire width of the browser window.

4. **Open the *index.html* file in a text editor. Locate the** `<!-- top section -->` **comment and add another div, just below it:**

```
<!-- top section -->
<div class="section header">
  <div class="container">
```

   Here, you're beginning to wrap that container div with another div. You've given it a couple of class names—`section` and `header`—so you can style this area separate from other parts of the page. Now you need to close the div.

5. **Add a closing** `</div>` **just *before* the** `<!-- info section -->` **comment:**

```
</div>
<!-- info section -->
```

Now there's a new div that wraps around the container. By default, like all block-level elements, divs have a width of 100%. That is, they stretch to fill their parent element, which in this case is the page's body. So this new div will fill the browser window. Now you can add a photo that will fill the width of the browser window, but only sit inside the header section.

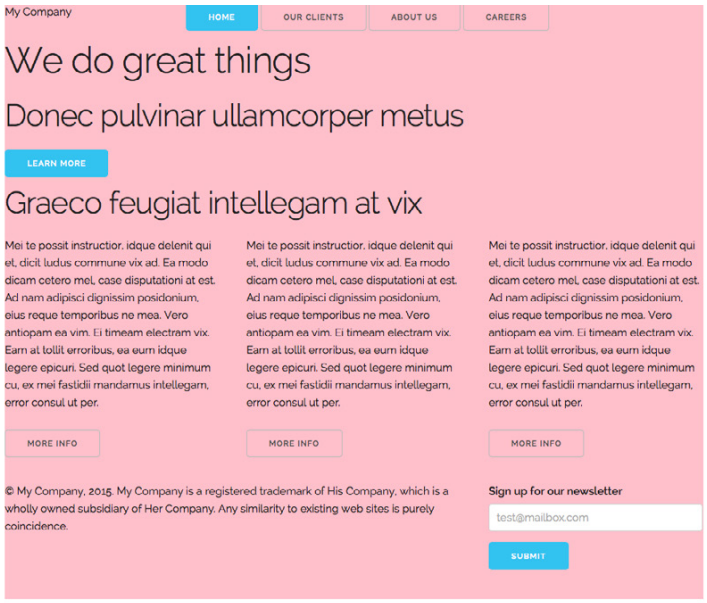

FIGURE 16-8
Skeleton's `container` *class doesn't extend to fill the width of a browser window. This is a problem if you add a background color or image to the container.*

6. **Save the *index.html* file, and return to the *custom.css* file in your text editor. Delete the `.container` rule you added in step 2 above, and replace it with this rule:**

```
/* Mobile first queries */
/* applies to ALL widths */
.header {
  padding: 50px 0 70px 0;
  background-image: url(../imgs/header.jpg);
  background-size: cover;
}
```

The `padding` property just adds a little breathing room at the top and bottom of this region of the page. The `background-size` property (discussed on page 245) scales the image added by the `background-image` property so it always fills the background. If you save the files now and preview the *index.html* file, you'll notice that the picture does span the browser window but only fills the header area.

Now you can format that header area.

7. **Just below the** `.header` **style, add a style for the logo:**

```
.logo {
  font-weight: 600;
  color: rgb(255, 209, 143);
}
```

This page uses a Google font (page 140) named Raleway, which includes three font weights. As you read on page 157, you can use numbers to indicate different font styles from very thin to very heavy and bold. 600 is a bold version of the font.

Next, you'll align the navigation to the right side of the page and improve the way the buttons look against the photo background.

8. **Add three more styles after the** `.logo` **style:**

```
.nav {
  text-align: right;
}
.button {
  background-color: rgba(255,255,255,.5);
}
.button:hover {
  background-color: rgba(255,255,255,.3);
}
```

The `.nav` style formats the column containing the navigation buttons. In step 5 on page 510, you added `nav` to the class attribute for that div. The other two styles change the appearance of the buttons so that they stand out better from the photo background.

Lastly, you'll center the text and button that appears below the navigation area.

9. **Add three more styles below the button styles you just added:**

```
.action {
  text-align: center;
  padding-top: 75px;
}
.action h1 {
  margin: 0;
}
.action h2 {
  margin: 0 0 20px 0;
}
```

You added the `action` class in step 10 on page 512. Here, you're centering the content in that area, and removing some of the spacing around the headlines.

10. **Save the files and preview the *index.html* file in a browser.**

    The page should look like Figure 16-9.

FIGURE 16-9
An image and a few styles of your own add some skin to Skeleton's bones.

The Mobile Design

As you read on page 507, Skeleton uses a mobile-first approach to its CSS. That is, you start with a design for small mobile devices like phones. Then you add styles inside a series of media queries that define progressively wider screen widths. Each set of styles from the previous query applies to the current query and all other queries with greater `min-width` values.

Think about a few enhancements you could make for visitors using mobile phones. Figure 16-10 shows what the current page would look like in an iPhone 5: There's a lot of space at top of the screen. You can fix the logo style to put the company name at the top of the screen, and fix the navigation buttons to fit better.

1. **In your text editor, open the file *16→css→custom.css*.**

   The styles you added to this file earlier in this tutorial are at the top of the file, outside any media queries. In keeping with the mobile first approach, the styles here will apply regardless of the browser width, on both small and super-wide desktop displays.

   First, you'll refine the `.logo` style so that it looks best on a small phone.

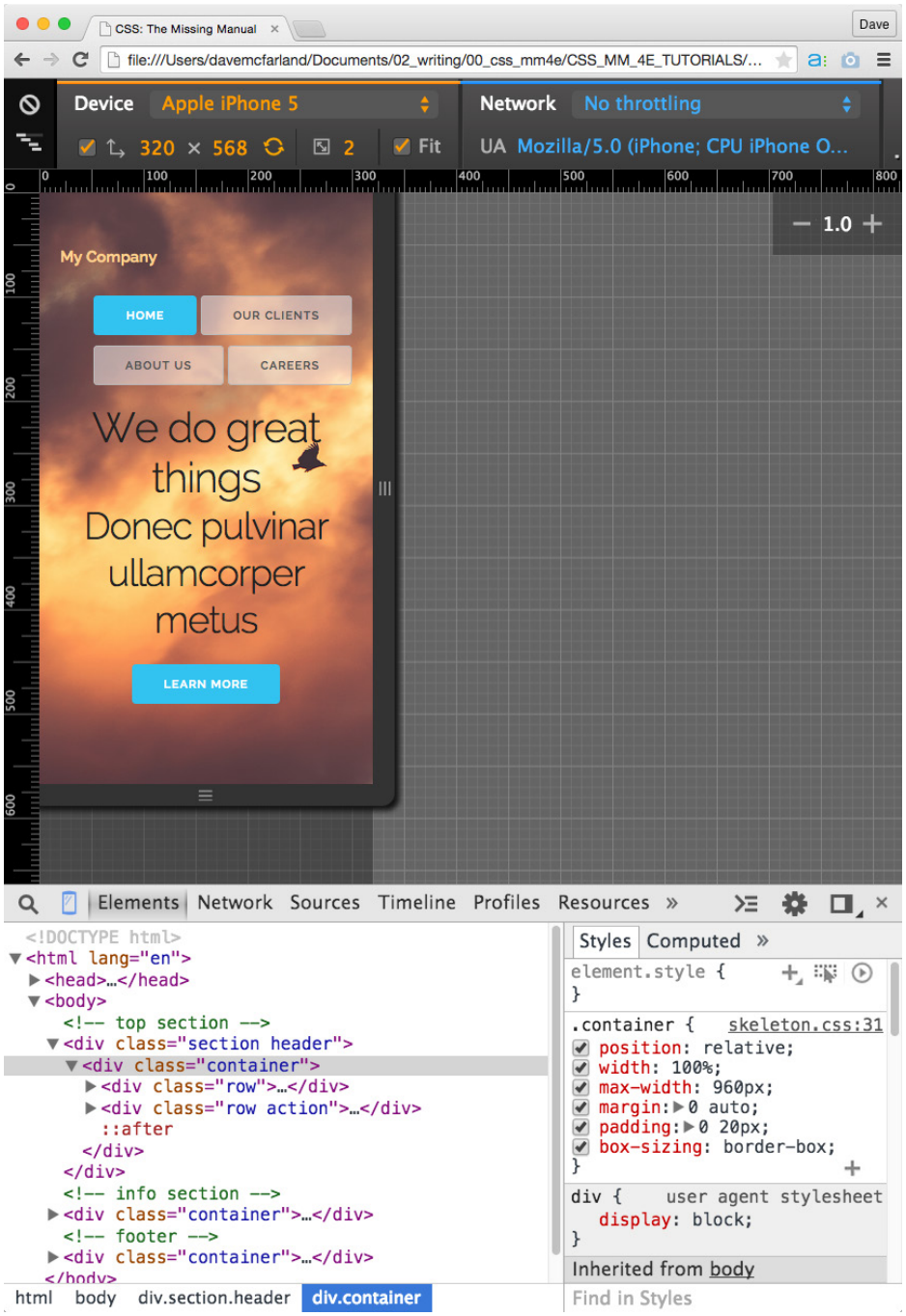

FIGURE 16-10

Using Google Chrome's Developer Tools, you can simulate the screen size of many different devices, including an iPhone 5 (pictured). See page 475 for more information on this cool tool built into Chrome.

2. **Locate the** `.logo` **style and add seven new declarations (additions in bold):**

```
.logo {
  font-weight: 600;
  color: rgb(255, 209, 143);
  background-color: black;
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  padding-left: 10px;
  z-index: 100;
}
```

The `background-color` setting makes the company name stand out. The `position`, `top`, `left`, and `right` settings fix this element to the top of the browser window. The `padding` setting adds a little breathing room on the left of the screen. Finally, the `z-index` setting (page 439) makes sure that as a visitor scrolls down the page, the logo sits above and on top of other page content (Figure 16-11, left).

The navigation buttons would look better if they were the same width and didn't have so much space around them.

3. **Below the** `.nav` **rule in the *custom.css* file, add the following style:**

```
.nav .button {
  width: 48%;
  margin: 2px 0;
}
```

This style makes the buttons in the navigation bar all the same width. It also adjusts the margins so the buttons don't touch (Figure 16-11, top right).

There's still too much space at the top of the page.

4. **Locate the first style in the *custom.css*—the** `.header` **rule—and change the** `padding` **values to** `30px 0 20px 0`**; to remove the space at top.**

Now the navigation and logo fit better on a small screen, and there's not so much space below the "Learn More" button (Figure 16-11, bottom left). Lastly, it would be good to make that big headline a tad smaller and add more space below the nav buttons.

5. **Locate the** `.action` **rule and change its** `padding-top` **value to** `37px`**.**

This adds some space between the first headline and the nav buttons. Now to shrink the font size of those headlines.

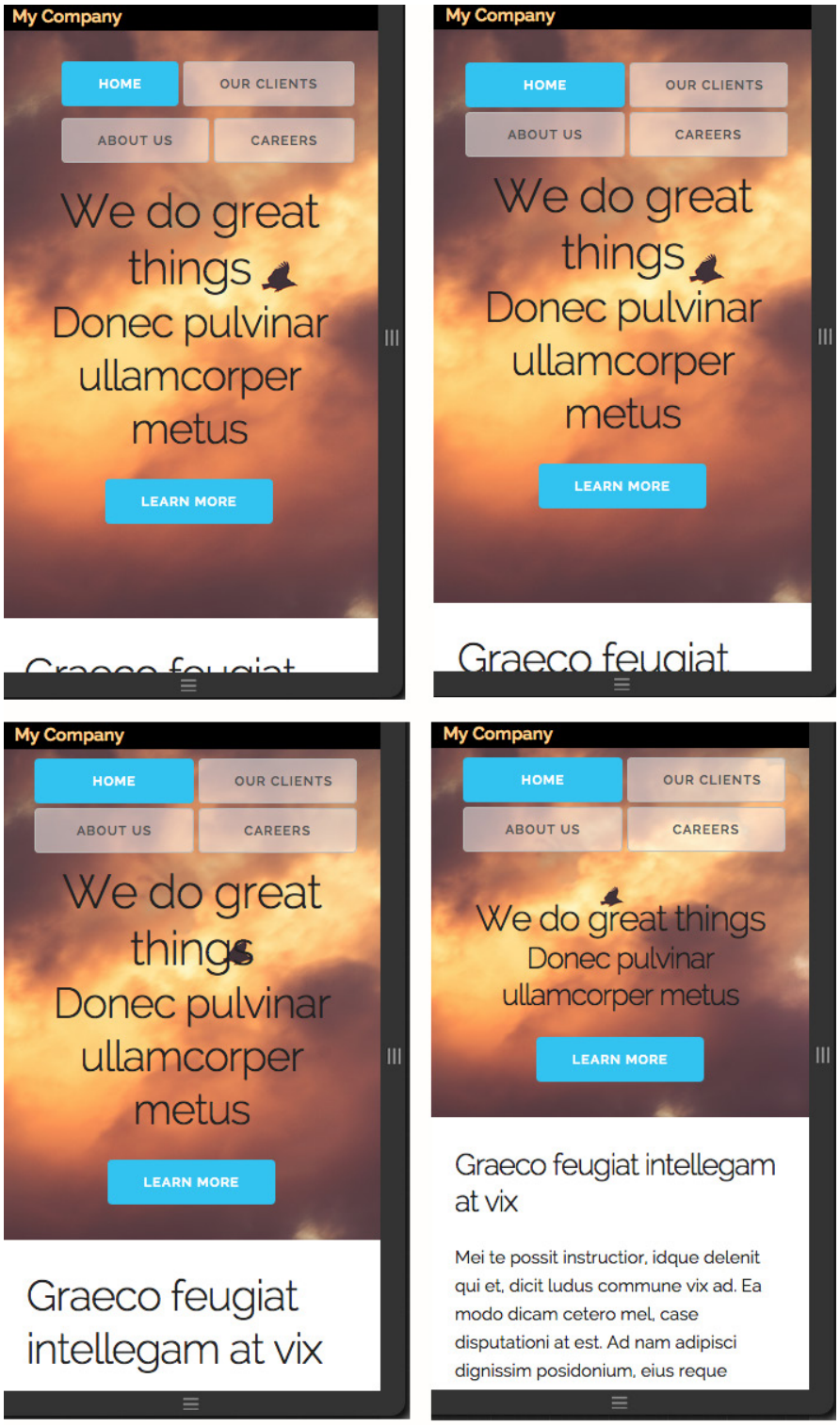

FIGURE 16-11

Creating a mobile-first design requires constant iteration—adjusting styles to adjust spacing, font size, and element placement for different types of devices with different screen sizes.

6. **In the *custom.css* file, after the `.action h2` rule, add the following two styles:**

```
h1 {
  font-size: 3rem;
}
h2 {
  font-size: 2.5rem;
}
```

If you save the file and preview it using an iPhone 5 setting in Chrome's dev tools you'll see something like the bottom right image in Figure 16-11.

Styling a Breakpoint

So far you've made a great start on a mobile-friendly design. It even looks good on screens up to around 550 pixels. In other words, you don't need to worry about adding any styles to the first media query—the one with the `min-width` value of 400 pixels. However, at 550 pixels, the design starts to look a little weird. This is the point at which Skeleton's built-in grid kicks in, and the design changes from stacked divs to columns.

As you can see in Figure 16-12, the navigation buttons take up a lot of space and aren't aligned. It's time to make your design look better on progressively wider screens.

1. **Open the *custom.css* file in your text editor. Locate the media query with a `min-width` of `550px` and add the following styles inside it (additions in bold):**

```
@media (min-width: 550px) {
  .header {
    padding: 40px 0 50px 0;
  }
  .logo {
    position: static;
    background-color: transparent;
    font-size: 2rem;
    line-height: 1;
  }
  .nav .button {
    width: auto;
  }
  h1 {
    font-size: 5rem;
  }
  h2 {
    font-size: 4.2rem;
  }
}
```

FIGURE 16-12
Skeleton's CSS grid springs into action at 550 pixels. This is when a media query kicks in and the styles inside it follow the column settings you specified in the HTML.

These styles reset the spacing in the header section of the page; change the logo so it's no longer fixed on the screen; adjust the width of the buttons; and increase the font size of the headlines (Figure 16-13, top right).

Finally, you'll just pump up the size of the logo text.

2. **In the media query with a `min-width` of `750px`, add one last style:**

```
/* applies to all widths 750px and greater */
@media (min-width: 750px) {
  .logo {
    font-size: 3rem;
    position: relative;
    top: 5px;
  }
}
```

This changes the size of the company name and moves it over slightly.

FIGURE 16-13

Skeleton's responsive grid system makes it easy to create web pages that adjust to a variety of different browser widths, from phones to television screens.

3. **Save the *custom.css* file. Open the *index.html* file in a web browser, and shrink the window to less than 550 pixels wide. Then widen it slowly.**

   The page design when the browser window is less than 550 pixels looks like the top-left image in Figure 16-13. As you widen the browser, the page will change two times: once at 550 pixels (top right) and once again at 750 pixels (bottom).

   There's a lot more you can do to perfect this design at different breakpoints. Play around by adding different styles at the different media query breakpoints and see how you can improve this design.

   You'll find the finished design in the *16_finished* folder inside the tutorials folder.

As you can see, a grid system is a handy tool that makes creating consistent column widths easy. However, underneath the hood, you'll find the same principles you learned earlier in this section of the book: floats, percentage widths, and so on. If you spend some time looking at the *skeleton.css* file, you can learn all about how it works.

TIP For a great discussion on how to build your own grid system with CSS read the article at *https://css-tricks.com/dont-overthink-it-grids/*.