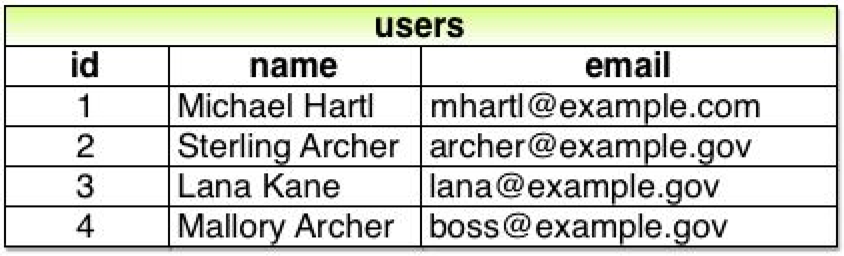
Chapter 6: Modeling Users

Making User Models

```
$ rails generate model User name:string email:string
```

Model entries each have unique integer IDs in addition to specified attributes.



users		
id	**name**	**email**
1	Michael Hartl	mhartl@example.com
2	Sterling Archer	archer@example.gov
3	Lana Kane	lana@example.gov
4	Mallory Archer	boss@example.gov

That command generates a migration file (*app/db/migrate/[timestamp]_modelname.rb*) that details what will be done to change the database when an update, or migration, happens.

```
$ rails db:migrate
```

On first migrate, it creates the database in the db directory with a .sqlite3 extension.

http://sqlitebrowser.org/ - a program to open and view the contents

To mess around with the database from the command line, this command will open a dev console that rolls back changes when exited: $ rails console --sandbox

Here's an example of some of the things you can do:

foo = User.create(name: "Foo", email: "foo@bar.com") **to establish an object variable**
foo.save **to put it in the database**
foo.destroy **to destroy the object and remove it from the database**
foo.email = "mhartl@example.net" **to set a specific attribute (email)**
foo.save **to apply**
foo.update_attributes(name: "The Dude", email: "dude@abides.org") **also works**
foo.reload.email **to reset a attribute (email) to what it was if it hasn't been saved yet**

User.find(1) **gets the entry with id 1**
User.find_by(email: "mhartl@example.com") **checks a specific field**
User.first **gets the first user**
User.all **(effectively) returns an array of all users**

User Validation

You probably don't want people to be able to put blank names and emails, so in order to start setting up for validating those fields, the tutorial recommends setting up tests. Like most other aspects of the application, models have their own directory for tests in the test directory.

The approach used in the tutorial:

```
def setup
@user = User.new(name: "Example User", email: "user@example.com")
end

test "should be valid" do
assert @user.valid?
end

test "name should be present" do
@user.name = "   "
assert_not @user.valid?    #note the _not
end
```

The third block isn't valid until there is a use of validates in the app/models/user.rb file of the model:

validates :name, presence: true, length: { maximum: 50 }

Other useful things for the model file:

format: { with: /<regular expression>/ } to test for a certain format, like an email format http://www.rubular.com/ good for testing regular expressions! It also has information on how to construct them.

Asserts can have custom messages. For example,
assert_not @user.valid?, "#{invalid_address.inspect} should be invalid"

The model's .rb file can also contain a "before_save {}" block, which will execute some code on the model before saving any new entry:
before_save { self.email = email.downcase }

It's necessary to use that syntax! You can't just do email = email.downcase, because the self. is optional on the right side, but not on the left.

uniqueness: true will make sure there isn't a duplicate entry with the same information.
uniqueness: { case_sensitive: false } if case sensitivity doesn't matter, does same thing

An important caveat with `uniqueness: true` is that it does not guarantee database uniqueness!! This is a big problem, because if this is part of a form and a user hits submit more than once, it can send a bunch of requests at the same time and create duplicates.

Thankfully, it's not hard to add some rudimentary database-side validation to prevent that from happening. In the case of our tutorial app, you can do this by generating a migration and manually editing the migration file to add a uniqueness check to a certain column.
(`$ rails generate migration add_index_to_users_email` used in book)

The contents that the tutorial has us add to the migration file (before migrating, obviously):

```
class AddIndexToUsersEmail < ActiveRecord::Migration[5.0]
    def change
        add_index :users, :email, unique: true
    end
end
```

In order to make this not throw an error, you'll need to change a fixtures file (test/fixtures/(users).yml) which has some lines that will create identical test entries in the database. The tutorial has us delete everything in it, so it probably isn't valuable until way later.

Adding a Password

Having a secure password for your users is as easy as putting "has_secure_password" in the the user model. It's a method that adds the following functionality:

- The ability to save a securely hashed password_digest attribute to the database, so it isn't stored in plaintext
- A pair of virtual attributes (password and password_confirmation), including presence validations upon object creation and a validation requiring that they match
- An authenticate method that redirects the user to a page of your choice when the password is correct (and false otherwise)

The only requirement for has_secure_password to work its magic is for the corresponding model to have an attribute called "password_digest".

To add it to our users in the sample app, we used

```
$ rails generate migration add_password_digest_to_users password_digest:string
```

Which generated a migration file with everything that we needed.

Also, put gem 'bcrypt' in your gemfile so that the has_secure_password method can use a decent hashing method.

Once everything is in place and you've updated pre-password-adding-migration users to have passwords and password confirmations, you can use the method authenticate to give a password and check if it's right. For example, if you were in the console and had a user set to a user variable, you could do the following:

```
>> user.authenticate("not_the_right_password")
false
>> user.authenticate("foobaz")
false
>> user.authenticate("foobar")
=> #<User id: 1, name: "Michael Hartl", email: "mhartl@example.com",
created_at: "2016-05-23 20:36:46", updated_at: "2016-05-23 20:36:46",
password_digest:
"$2a$10$xxucoRlMp06RLJSfWpZ8hO8Dt9AZXlGRi3usP3njQg3...">
```

As you can see, putting in the correct password will return the user object itself.