- Get started with **rake db:migrate** to generate an example **`db/models.yml`** file.
- Define your models and their attributes in **`db/models.yml`**
- Whenever you change db/models.yml, be sure to **rake db:migrate** to update your database
- Models are simple Ruby classes that represent real-world things
- In Rails, models are expected to be under **/app/models**
- Database-backed models should derive from ActiveRecord::Base
- Database-backed models map object instances to rows, and attributes to columns
- Every column in a table corresponds to an object property of the same exact name.
- Use rails console to load your Rails app and interact with your database
- Learn model-based CRUD: **`create`**, **`read`**, **`update`**, and **`delete`**
- Creating a new row is usually a 2-stop process: **`.new`** and then **`.save`**
- Read rows of data using **`.where`** or a single row using **`.find_by`**
- Every model has a primary key column named **`id`**
- You will sometimes need to learn a tiny bit of SQL for **`where()`** fragments
- Typical usage: **`where(:title => "Apollo 13")`**
- Use **`LIKE`** with `%` wildcards to perform fuzzy searches
- Typical usage: **`where("title LIKE %ollo%")`**
- When two real-world things are related to each other in a one-to-many relationship, put a foreign-key column on the “many” side of the relationship
- Use **`<input>`** tags within a form
- You can use **`<button type="submit">`** or **`<input type=”submit”>`**
- You can use a **`<select>`** tag in HTML to help the user associate things together

`new` **`save`** **`delete`** **`delete_all`**

`count` **`all`** **`where`** **`find_by`**

`limit` **`order`** **`update`**

Suppose we are building Amazon.com and we identify that we will need a software domain model named *product* to represent real-world products.

Here is a simple 3-Step Recipe:

1. Add the model and its attributes to **`db/models.yml`**:

```
Product
  title: string
  sku: string
  price: integer
  photo: string
```

2. **`rake db:migrate`**

3. **`Verify Everything`**

Verify that you now have a file named app/models/product.rb

Finally verify that your local database has your new table defined:

```
$ rails console
> Product.count
=> 0
```

Models are Always Singular!

Product

not

Products

config/models.yml

```
Movie
    title: text
    year: integer
    poster_url: text
```

app/models/movie.rb

```
class Movie < ActiveRecord::Base

end
```

movies_

id	title	year	poster_url
1	Apollo 13	1995	http://....
2	Guardians of the Galaxy	2014	http://....
3	Backdraft	1991	http://....
4	Star Wars	1977	http://....
5	Toy Story	1995	http://....

```
irb> Movie.count
=> 5

irb> Movie.where(:year => 1995).count
=> 2

irb> Movie.find_by(:id => 4).title
=> “Star Wars”

irb> Movie.where(“title LIKE %St%”).count
=> 2
```

config/models.yml

```
Movie
    title: text
    year: integer
    poster_url: text
    studio_id: integer

Studio
    name: string
```

movies

id	title	year	poster_url	studio_id
1	Apollo 13	1995	http://....	4
2	Guardians of the Galaxy	2014	http://....	2
3	Backdraft	1991	http://....	4
4	Star Wars	1977	http://....	1
5	Toy Story	1995	http://....	3

studios

id	name
1	LucasFilm
2	Marvel
3	Pixar
4	Other

```
irb> m = Movie.find_by(:id => 5)
irb> Studio.find_by(:id => m.studio_id).name

=> "Pixar"
```

Cheat Sheet: Model CRUD in the Rails Console

Models are our gateway to our application's data set. Each model is a table of data. We can have as many rows in the table as we want. The columns are defined by the model's schema.

Once a model's table schema has been defined in the `db/models.yml` file and the corresponding `app/models` class file exists, we can create new rows to the table by having the user fill out forms in our web interface or by manually adding rows by using the `rails console` tool. We can also read rows from the table, update individual rows, and delete rows.

These four activities --- create, read, update, and delete --- are often referred to by their acronum, `crud` .

This cheat sheet summarizes how to CRUD any given model using the Rails console.

For the purposes of this cheat sheet, we will presume that our `db/models.yml` file looks like this:

```
Book:
  title: string
  author: string
  summary: text
  hardcover: boolean
```

and that you've run the `rake db:migrate` command. You should end up with a Book model class defined in `app/models/book.rb` that looks like this:

```
class Book < ActiveRecord::Base
end
```

and the `db/schema.rb` file should look something like this:

```
ActiveRecord::Schema.define(version: 20140402002437) do
```

```
    create_table "books", force: true do |t|
      t.string  "title"
      t.string  "author"
      t.text    "summary"
      t.boolean "hardcover"
    end

  end
```

Start the Rails Console

To begin, make sure you've started the console by opening a command prompt at your application folder:

```
rails console
```

You should see something like this:

```
Welcome to the Rails Console.
------------------------------------------------------------

Use this console to add, update, and delete rows from the database.

Models: Book

HINTS:
* Type 'exit' (or press Ctrl-D) to when you're done.
* Press Ctrl-C if things seem to get stuck.
* Use the up/down arrows to repeat commands.
* Type the name of a Model to see what columns it has.

Loading development environment (Rails 4.0.4)
irb(main):001:0>
```

IMPORTANT: Don't forget to read the HINTS section that's displayed for you!

Creating New Rows

Adding new rows to a model's data table is pretty easy. We just use the `.create` method on our model class, and provide a *hash* of data that assigns cell values for each column in our new row.

```
irb(main):001:0> Book.create(title: "Sherlock Holmes", author: "Arthur Conan Dc
   (0.1ms)  begin transaction
  SQL (0.4ms)  INSERT INTO "books" ("title") VALUES (?)  [["title", "Sherlock H
   (1.1ms)  commit transaction
=> #<Book {"id"=>3, "title"=>"Sherlock Holmes", "summary"=>nil, "author"=>"Arth
```

There are some important things to notice in the above example:

- We don't have to provide values for every column. If you don't provide a value for a `boolean` column, it will be assigned as `false`. For all other column types, they will be `nil`.
- The INSERT jargon you see above is Ruby talking to the database on our behalf, instructing it to insert a new row into the table.
- The final line that starts with `=>` shows us the grand result of our `Book.create` instruction. We've created a new `Book` object, and the database assigned it an `id` value of `3`.

Reading Rows of Data

Reading, or *querying*, our model is also pretty easy. There are lots of ways we to read data from the table, depending in what question you'd like to ask.

How Many Rows Are There? `Book.count`

Display All Rows `Book.all`

What is the first book? `Book.first`

What is the last book? `Book.last`

Retrieve the book that has an `id` of 1. `Book.find_by(id: 1)`

Retrieve all hardcover books. `Book.where(hardcover: true)`

Retrieve the first book in the table that was written by Plato. `Book.find_by(author: "Plato")` `Book.where(author: "Plato").first`

How many paperback books are there? `Book.where(hardcover: false).count`

Accessing a Row's Attributes

Once you have a row's worth of data in hand, you can drill down to a specific attribute. To make things a little easier to follow in these examples, we show how to first capture the result of a query into a variable, then ask for a specific attribute.

Who wrote 'Sherlock Holmes'?

```
mystery_book = Book.find_by(title: "Sherlock Holmes")
mystery_book.author
```

Is 'Sherlock Holmes' in paperback or hardcover?

```
mystery_book = Book.find_by(title: "Sherlock Holmes")
mystery_book.hardcover?
=> false
```

Updating a Row

Another thing you can do once you have retrieved a specific row is update it's attribute values.

```
mystery_book = Book.find_by(title: "Sherlock Holmes")
mystery_book.title = "The Adventures of Sherlock Holmes"
mystery_book.save
```

Notice that we have to call the `.save` method to update the data table based on the contents of our `mystery_book` variable.

Deleting a Row

Finally, here's how we delete a row:

```
mystery_book = Book.find_by(title: "Sherlock Holmes")
mystery_book.delete
```

If you want to delete every row in the table in one fell swoop, you can. Here, we will ask for a before-and-after count to prove that we did indeed lose all of our data:

```
Book.count
=> 1

Book.delete_all
=> 1

Book.count
=> 0
```