

# Associations and Active Record

# Our Goals

- Understand Object Relational Mapping
- Understand Convention over Configuration
  - Specifically how they apply to Active Record
- Be able to use Active Records methods to perform common CRUD apps
- See our first association

# What is ORM?

- Object relational mapping
- It is one of the guiding principles of Object Oriented Programming
- It's the idea of rich objects. Meaning it should be easy, logical, consistent and concise to:
  - Create new objects
  - Retrieve old objects
- Works particularly well with databases

# Active Record as an ORM

- Represent models and their data

# Active Record as an ORM

- Represent models and their data
- Represent associations between these models

# Active Record as an ORM

- Represent models and their data
- Represent associations between these models
- Represent inheritance hierarchies through related models

# Active Record as an ORM

- Represent models and their data
- Represent associations between these models
- Represent inheritance hierarchies through related models
- Validate models before they get persisted to the database

# Active Record as an ORM

- Represent models and their data
- Represent associations between these models
- Represent inheritance hierarchies through related models
- Validate models before they get persisted to the database
- Perform database operations in an object-oriented fashion



# Active Record as an ORM

- Represent models and their data
- Represent associations between these models
- Represent inheritance hierarchies through related models
- Validate models before they get persisted to the database
- Perform database operations in an object-oriented fashion
- Protects us from writing our own SQL - much more secure

# Convention over Configuration

- Convention over configuration is a great rule in general
- But Active Record requires it. If you follow the rules, it will:
  - Create all the connections
  - Deal with associations etc.

# Convention over Configuration

## **Database Tables**

Plural with underscores separating words (articles, line\_items etc.)

## **Model / Class Names**

Singular with the first letter of each word capitalized (Article, LineItem etc.)

# How to work with it?

```
# Load the gem so you can work with it

require 'active_record'

# Set up the connection to the database

ActiveRecord::Base.establish_connection(
  :adapter => 'sqlite3',
  :database => 'database.db'
)

# Not necessary, but prints the generated
# SQL to the console

ActiveRecord::Base.logger = Logger.new(STDERR)
```

# Close the connection!

```
# Make sure you close the connection after  
# every route!  
  
after do  
  ActiveRecord::Base.connection.close  
end
```

# How to work with it?

```
class Animal < ActiveRecord::Base  
  # associations go here  
end
```

```
class Post < ActiveRecord::Base  
end
```

```
class Person < ActiveRecord::Base  
end
```

# Parallels

	<b>Verb</b>	<b>SQL</b>	<b>Active Record</b>
Create	POST	INSERT	.create or .new/.save
Read	GET	SELECT	.find, .find_by or .where
Update	PATCH/POST/PUT	UPDATE	.update or .find/.save
Delete	DELETE/POST/GET	DELETE	.destroy or .destroy_all

# CRUD - Create

```
plant = Plant.new
plant.name = "Hibiscus"
plant.flowers = true
plant.save # Necessary!

# These will run .save automatically

Plant.create( :name => "Hibiscus", :flowers => true )

user = User.new do |u|
  u.name = "David"
  u.occupation = "Code Artist"
end
```



# CRUD - Read

```
Plant.all
Plant.first
Plant.last
Plant.find( 10 ) # Find with an ID

# Returns the first plant that this works with
Plant.find_by( :name => "Hibiscus" )

# Returns all instances where this is appropriate
Plant.where( :name => "Hibiscus" )
```

# CRUD - Update

```
plant = Plant.find_by( :name => "Hibiscus" )  
plant.name = "Hibiscus 2"  
plant.save  
  
# This will save automatically  
plant = Plant.find_by( :name => 'Hibiscus 2' )  
plant.update( :name => 'Hibiscus' )
```

# CRUD - Delete

```
plant = Plant.find_by( :name => 'Hibiscus' )  
plant.destroy  
  
Plant.destroy_all
```

# Associations

```
class Plant < ActiveRecord::Base
  has_many :butterflies
end

# The "belongs_to" always has an ID!
#   In this case, the Butterfly needs to
#   have a column called plant_id that has the
#   type of integer

class Butterfly < ActiveRecord::Base
  belongs_to :plant
end
```

# Important Links

- [Active Record Basics](#)
- [Active Record Query Interface](#)

Here is **your homework**