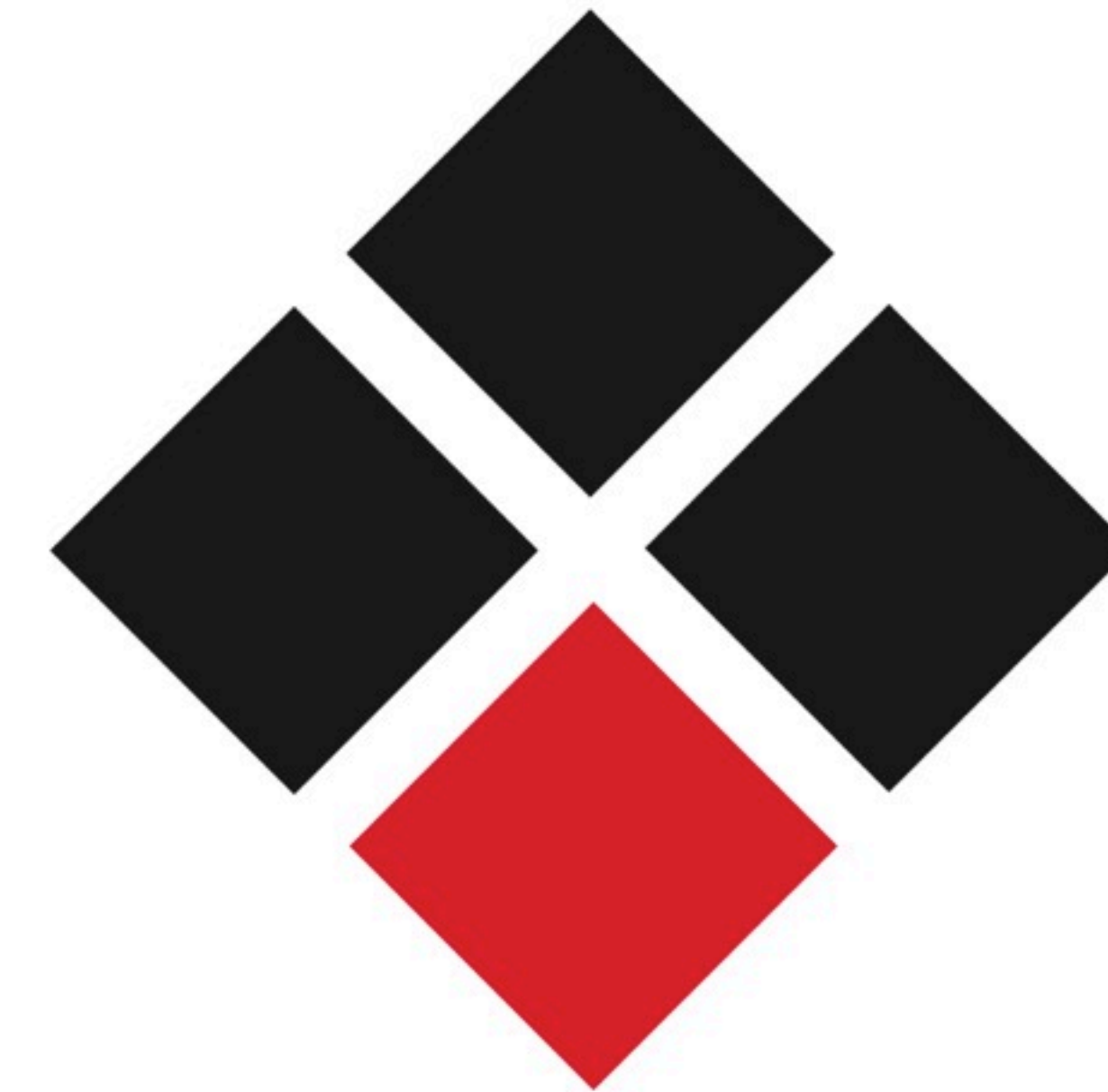


Intro to Ruby on Rails



Enable Labs

The Elevator Pitch




“Ruby on Rails® is an open-source web framework that’s optimized for programmer happiness and sustainable productivity. It lets you write beautiful code by favoring convention over configuration.”

- rubyonrails.org



The Rails Philosophy

- Ruby - less and more readable code, shorter development times, simple but powerful, no compilation cycle
- Convention over configuration
- Pre-defined directory structure, and naming conventions
- Best practices: MVC, Testing, DRY
- Almost everything in Rails is Ruby code
 - ActiveRecord abstracts SQL
 - YAML in database configurations and test fixtures
- Integrated AJAX support
- Web services with REST
- Great community, tools, and documentation
- Extracted from a real application:  Basecamp® from 37Signals



5 Minute Web App

```
$ rails new car_dealership
```

...

```
$ rails generate scaffold car make:string  
model:string year:integer
```

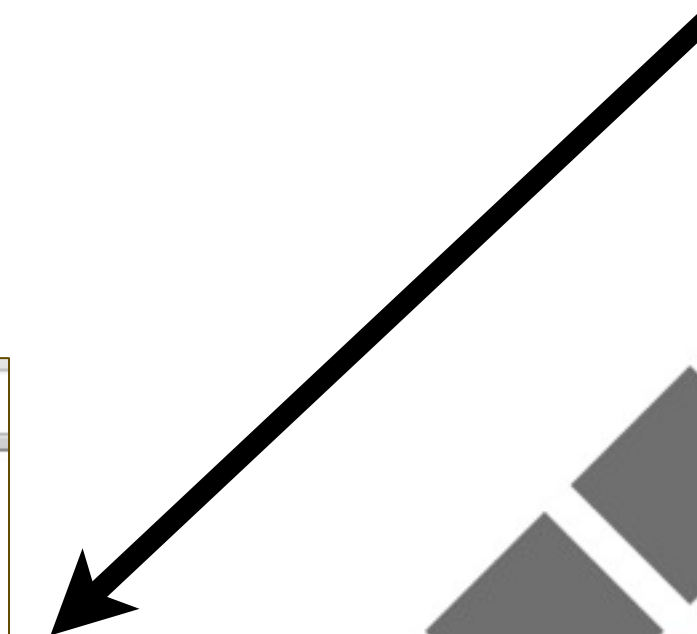
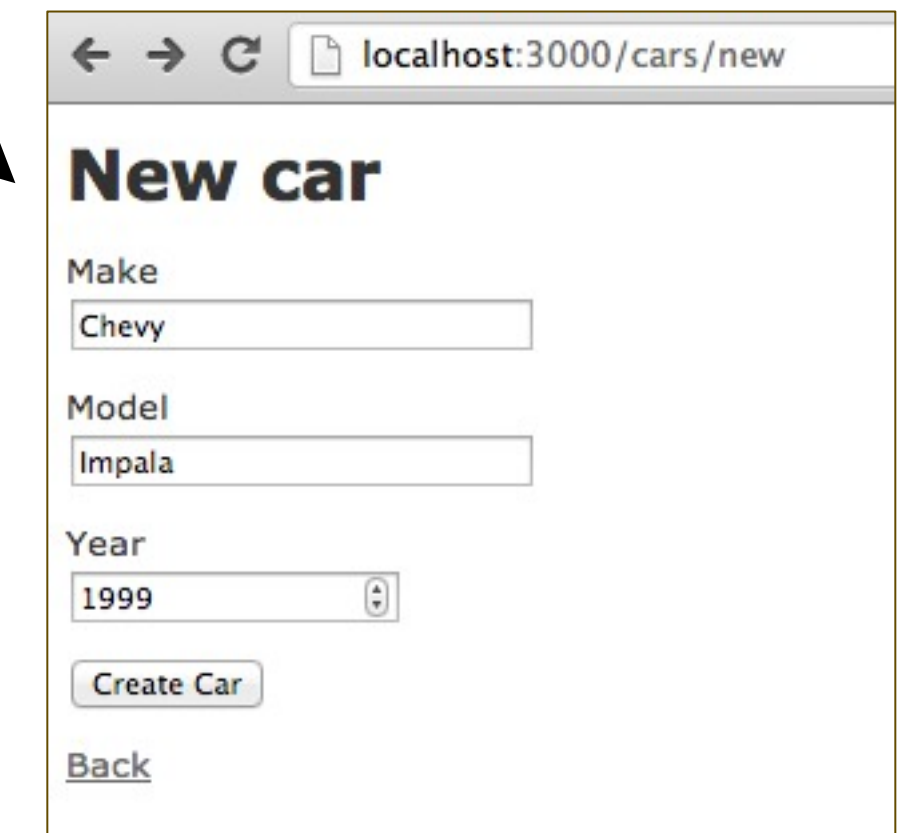
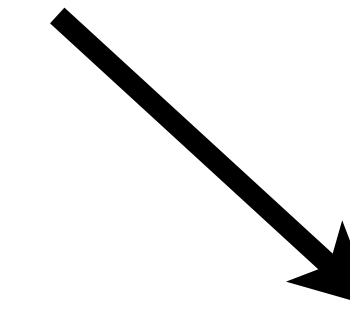
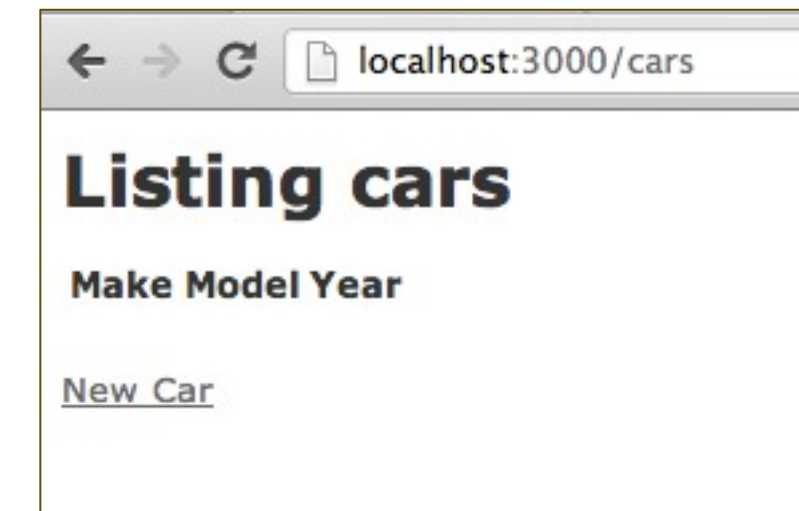
...

```
$ rake db:migrate
```

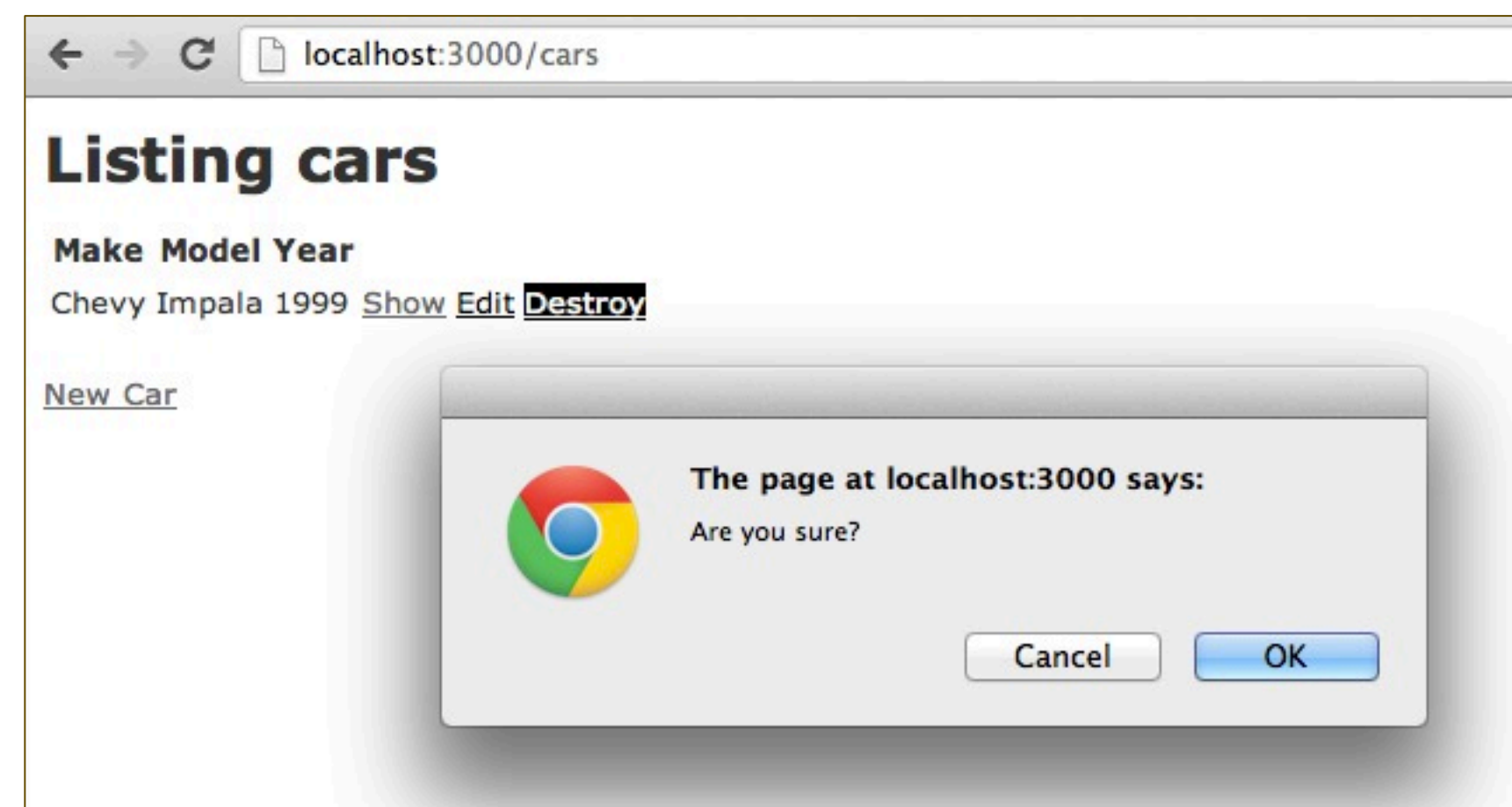
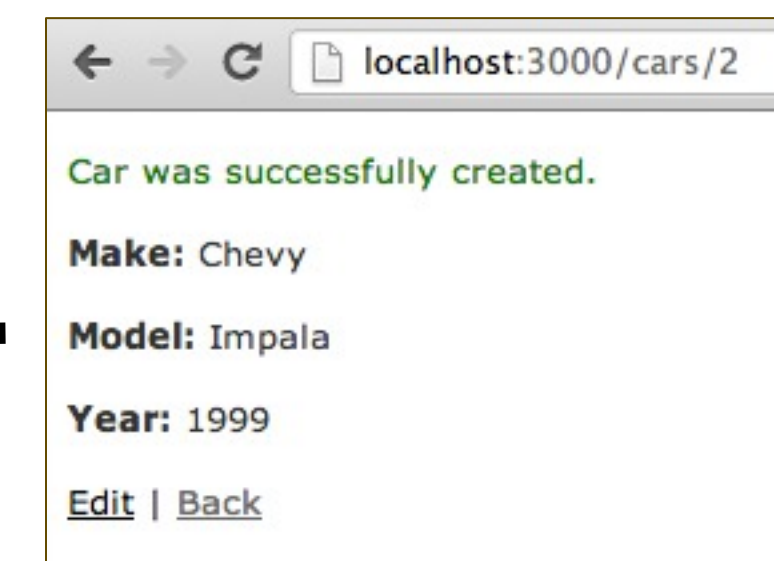
...

```
$ rails server
```

...



Enable Labs



Rails generated content

```
$ rails new car_dealership
  create
  create  README.rdoc
  create  Rakefile
  create  config.ru
  create  .gitignore
  create  Gemfile
  create  app
  create  app/assets/images/rails.png
  create  app/assets/javascripts/application.js
  create  app/assets/stylesheets/application.css
  create  app/controllers/application_controller.rb
  create  app/helpers/application_helper.rb
  create  app/views/layouts/application.html.erb
  create  app/mailers/.gitkeep
  create  app/models/.gitkeep
  create  config
  create  config/routes.rb
  create  config/application.rb
  create  config/environment.rb
  create  config/environments
  create  config/environments/development.rb
  create  config/environments/production.rb
  create  config/environments/test.rb
  create  config/initializers
  create  config/initializers/backtrace_silencers.rb
  create  config/initializers/inflections.rb
  create  config/initializers/mime_types.rb
  create  config/initializers/secret_token.rb
  create  config/initializers/session_store.rb
  create  config/initializers/wrap_parameters.rb
  create  config/locales
  create  config/locales/en.yml
  ...
```

```
...
create  config/boot.rb
create  config/database.yml
create  db
create  db/seeds.rb
create  doc
create  doc/README_FOR_APP
create  lib
create  lib/tasks
create  lib/tasks/.gitkeep
create  lib/assets
create  lib/assets/.gitkeep
create  log
create  log/.gitkeep
create  public
create  public/404.html
create  public/422.html
create  public/500.html
create  public/favicon.ico
create  public/index.html
create  public/robots.txt
create  script
create  script/rails
create  test/fixtures
create  test/fixtures/.gitkeep
create  test/functional
create  test/functional/.gitkeep
create  test/integration
create  test/integration/.gitkeep
create  test/unit
create  test/unit/.gitkeep
create  test/performance/browsing_test.rb
create  test/test_helper.rb
create  tmp/cache
...
```

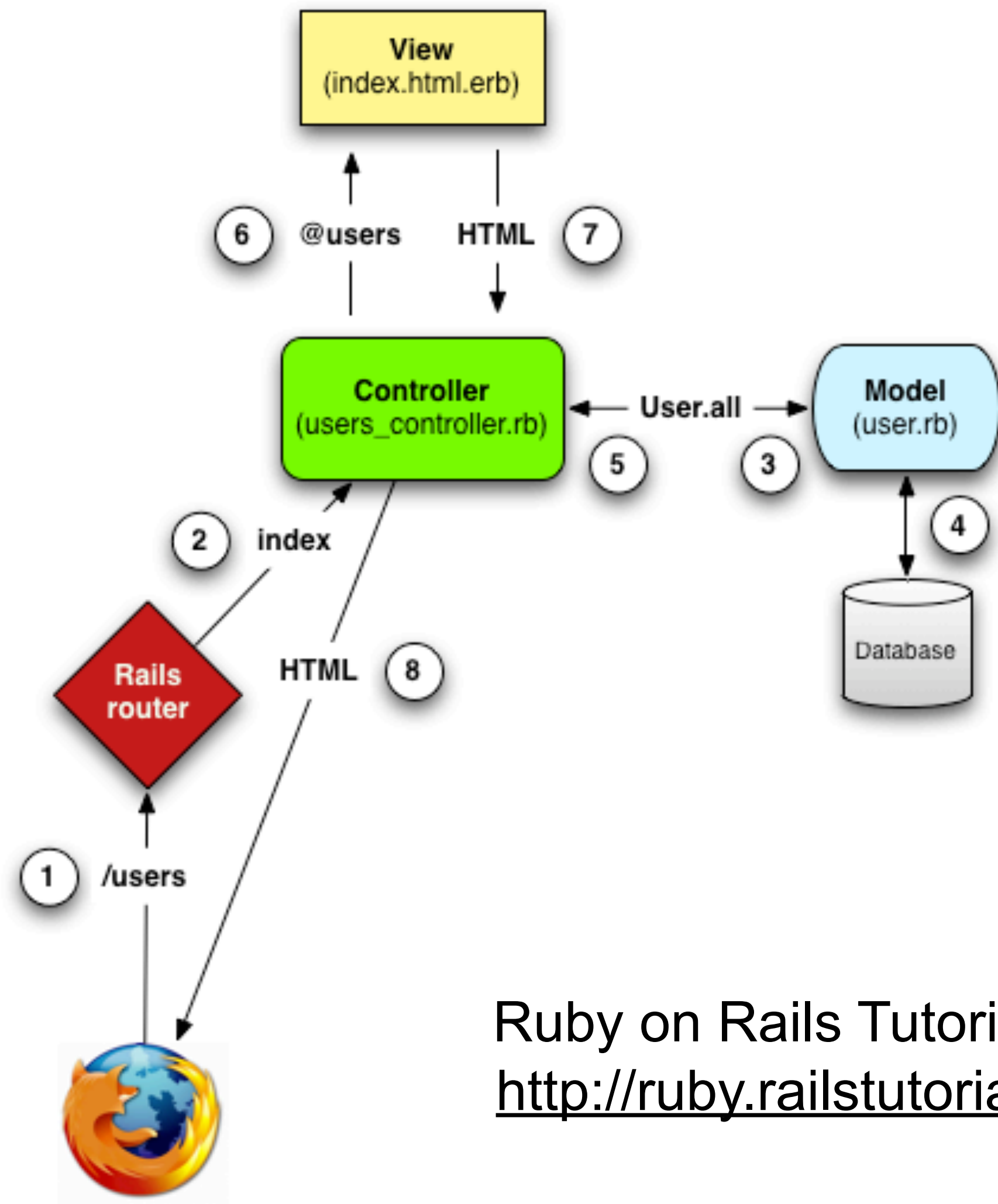
```
...
create  tmp/cache/assets
create  vendor/assets/javascripts
create  vendor/assets/javascripts/.gitkeep
create  vendor/assets/stylesheets
create  vendor/assets/stylesheets/.gitkeep
create  vendor/plugins
create  vendor/plugins/.gitkeep
run  bundle install
```



```
./app
./app/assets
./app/controllers
./app/helpers
./app/mailers
./app/models
./app/views
./config
./config/environments
./db
```



Web Request/Response Cycle



Ruby on Rails Tutorial by Michael Hartl
<http://ruby.railstutorial.org>



Enable Labs

Overview

- Rails is a full stack web framework following the Model-View-Controller (MVC) Pattern

- Model

- Object Relational Mapper (ORM)
- Business Logic

- View

- Template Rendering

- Controller

- Manages web integration

The “Rails” gems

ActiveRecord

ActionPack

ActiveView

ActionController

ActionMailer

ActiveResource

ActiveSupport

RailTies



Extensions to Ruby to support web development

The application libraries to glue it all together (initialize, config, .etc)



Enable Labs

Rake: The Ruby Make

- Rake tasks automate and simplify creating and managing the development of a Rails project
- Rake lets you define a dependency tree of tasks to be executed
- Over 30 Rake tasks predefined for your new Rails app
- Custom `<Rails.root>/lib/tasks/*.rake` files
- List your project's available rake tasks: `$ rake -T`

Sampling:

<code>rake db:create</code>	<code># Create the database from DATABASE_URL or config/database.yml ...</code>
<code>rake db:migrate</code>	<code># Migrate the database (options: VERSION=x, VERBOSE=false).</code>
<code>rake db:rollback</code>	<code># Rolls the schema back to the previous version (specify steps w/ STEP=n).</code>
<code>rake db:seed</code>	<code># Load the seed data from db/seeds.rb</code>
<code>rake db:setup</code>	<code># Create the database, load the schema, and initialize with the seed data ...</code>
<code>rake db:version</code>	<code># Retrieves the current schema version number</code>
<code>rake middleware</code>	<code># Prints out your Rack middleware stack</code>
<code>rake notes</code>	<code># Enumerate all annotations (use notes:optimize, :fixme, :todo for focus)</code>
<code>rake routes</code>	<code># Print out all defined routes in match order, with names.</code>
<code>rake test</code>	<code># Runs test:units, test:functionals, test:integration together ...</code>



Environments

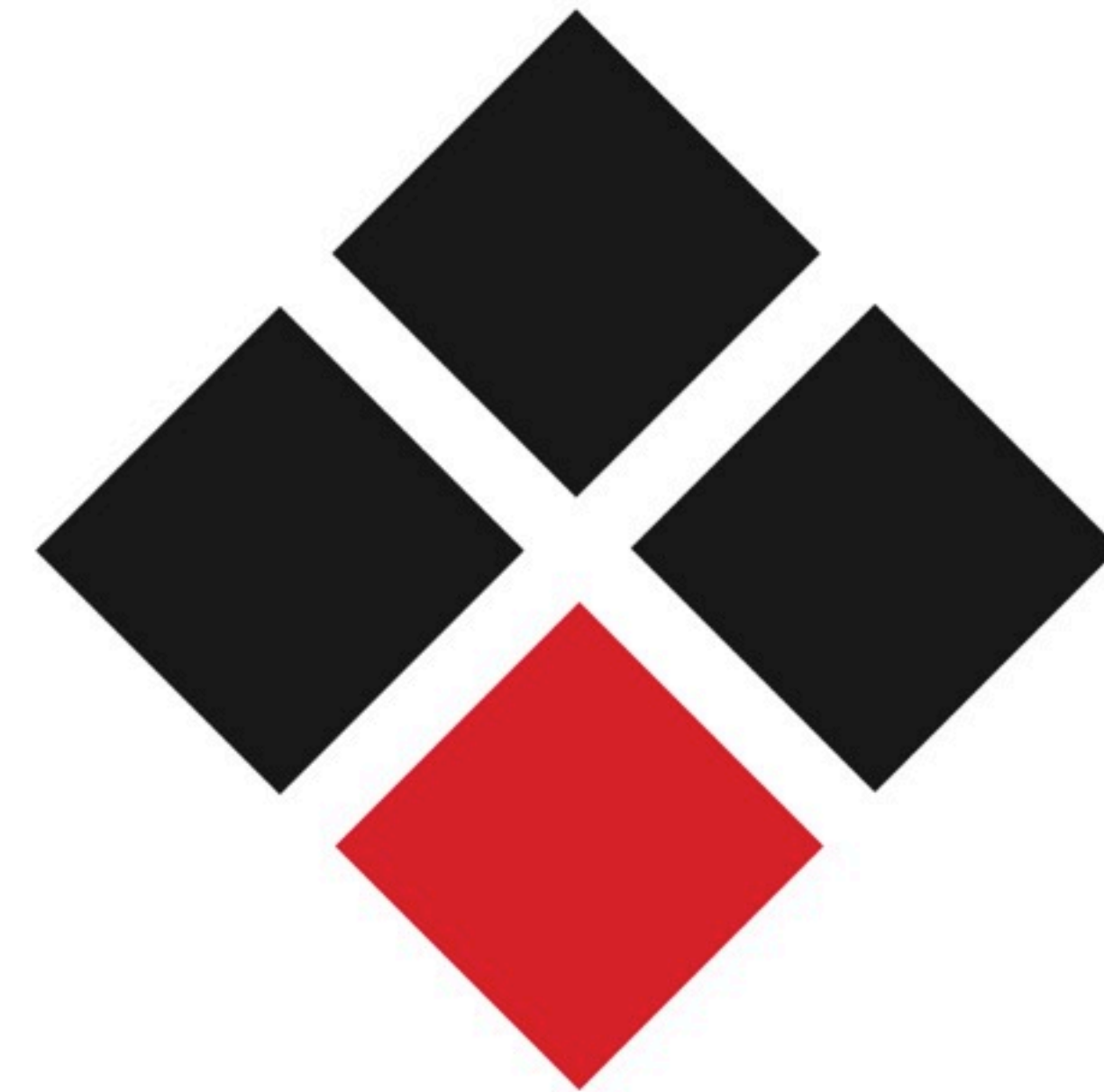
- Rails has support for multiple execution environments
- Environments encapsulate database settings and other configuration
 - separate database for each environment
 - separate config files for each environment
- Typical environments
 - Development
 - Test
 - Production
- Additional environments are easy to add
 - Staging
 - Data Warehouse

```
$ rails c staging  
$ rails server -e staging
```



ActiveRecord

Modeling the World



Enable Labs

ActiveRecord Fundamentals

<Rails.root>/app/models/car.rb

```
class Car < ActiveRecord::Base  
end
```

- One database table maps to one Ruby class
- Table names are plural and class names are singular
- Database columns map to attributes
 - automatically creates get and set methods in the model class
- Tables have an integer primary key called id (by convention)
- Model can reference a legacy database table or view
- Stick to the conventions as much as possible!
- Provide Callbacks (hooks into the object lifecycle)



ActiveRecord Model Example

```
class CreateCars < ActiveRecord::Migration
  create_table "cars" do |t|
    t.integer :year
    t.string :make
    t.string :model
    t.timestamps
  end
end
```

```
class Car < ActiveRecord::Base
  attr_accessible :year, :make, :model
end
```

**Mass-Assignment
Protection!**

Mass-Assignment!

```
car = Car.new
car.year = 1999
car.make = 'Ferrari'
car.model = 'Mondial'
car.save
```

OR

```
car = Car.create(
  year => 1999,
  make  => 'Ferrari',
  model => 'Mondial'
)
```



Enable Labs

CRUD - Create/Read/Update/Delete

- The basic functions of persistent storage

Create

```
car = Car.create(  
  :year => 2004,  
  :make => 'Chevy',  
  :model => 'Camaro'  
)
```

Read

```
car = Car.find(37)
```

Update

```
car.update_attributes(  
  :year => 2004  
)
```

Delete

```
car.destroy
```



Enable Labs

ActiveRecord Query Interface

- `Car.find(:id)`
- `Car.all`
- `Car.where(:make => "Chevy")`
- `Car.find_by_make("BMW")`
- `Car.where(:make => "Mercedes")`
- `Car.where('year = ? AND make = ?', 2009, 'VW')`
- `Car.where(:year => 2012, :make => "Ford")`
- `Car.where(:year => 2012).where(:make => "Ford")`
- `Car.where("make = #{make}")`

Note the use of the query interface on the class! ...for now

DON'T DO THIS, SQL INJECTION VULNERABLE!!!



Advanced Queries

- `Car.where(...)` returns `<ActiveRecord::Relation>`
- relations are chainable
 - `Car.where(:make => "Chevy").where(:model => ["Camaro", "Cruze"]).limit(5).order(:year)`
 - Car Load (0.2ms) `SELECT "cars".* FROM "cars" WHERE "cars"."make" = 'Chevy' AND "cars"."model" IN ('Camaro', 'Cruze') ORDER BY year LIMIT 5`
- `.limit(5)`
- `.offset(1001)`
- `.order("year DESC")`
- `.joins(:dealership)`
- `.select(:make)`
- `.group(:make).sum(:price)`



Transactions

```
Car.transaction do  
  my_account.withdraw(100)  
  dealer_account.deposit(100)  
end
```

The transaction will rollback if an exception is raised.



Lab 2 Part 1: CRUD

Cloning the Lesson Project

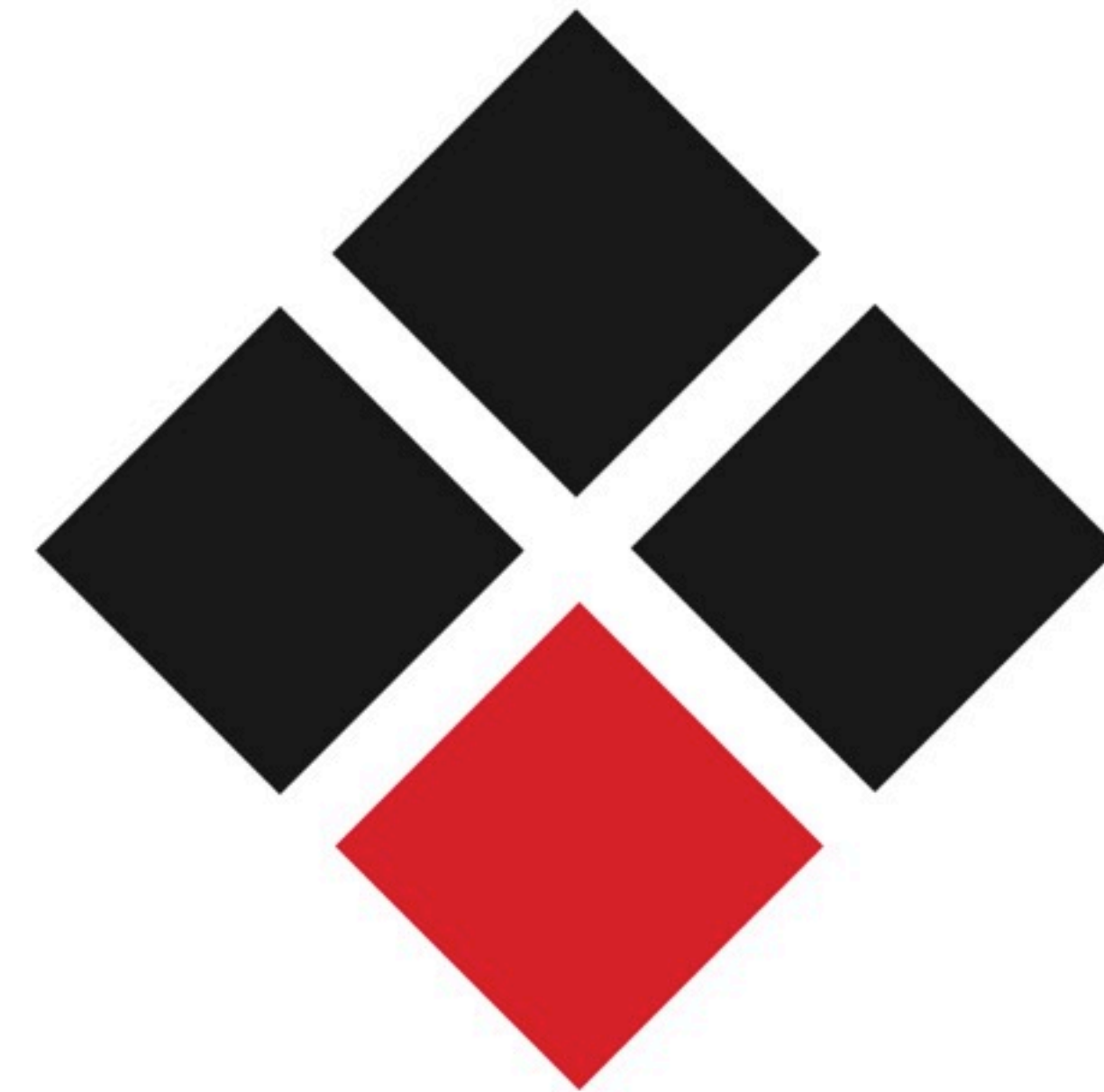
```
$ git clone https://github.com/EnableLabs/rails_training_feb_2013.git
$ cd ./rails_training_feb_2013/week2/car_dealership
  # if asked, please 'trust' the .rvmrc file
$ bundle install
```



Enable Labs

ActiveRecord Migrations

Managing Schema Evolution

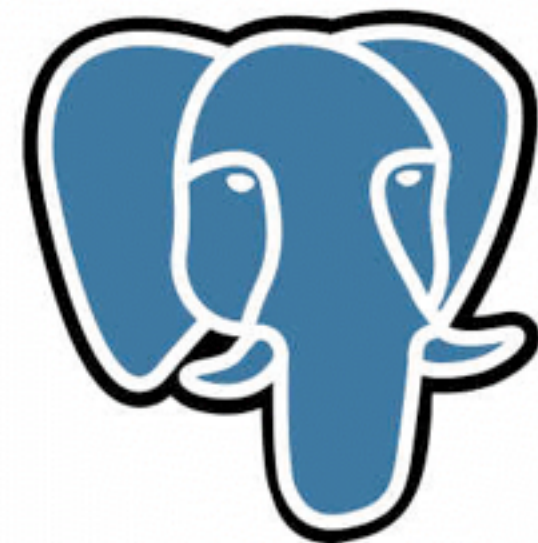


Enable Labs

ActiveRecord Migrations - Managing Data/Schemas

- Manage the evolution of your database schema
- Uses a database independent Ruby API
- Eliminate the need to use SQL for many tasks
- Can execute SQL if necessary!
- Can use the full power of the Ruby language and Rails DSL

PostgreSQL



Typical Migration Tasks

- create_table
- add_column
- change_column
- rename_column
- rename_table
- add_index

```
create_table "users", :force => true do |t|  
  t.string :login  
  t.string :email  
  t.string :remember_token  
  t.string :salt, :encrypted_password, :limit => 40  
  t.datetime :remember_token_expires_at  
  t.timestamps  
end
```



Enable Labs

Migration Example

<Rails Root>/db/migrate/20120919195331_add_is_exporting_to_document_transcriptions.rb

```
class AddIsExportingToDocumentTranscriptions < ActiveRecord::Migration
  def change
    add_column :document_transcriptions, :is_exporting, :boolean, :default => false
  end
end
```

<Rails Root>/db/migrate/2012091921337_add_sum_to_shipments.rb

```
class AddSumToShipments < ActiveRecord::Migration
  def up
    add_column :shipments, :sum, :decimal
    execute("UPDATE shipments set sum = subtotal*1.2")
  end
  def down
    drop_column :shipments, :sum
  end
end
```



Running Migrations

- Migrations are executed in order of their timestamp
- Only un-executed migrations are run
- Migrations can be rolled back*
- List of migrations is stored in schema_migrations table

```
$ rails generate migration CreateCars
  invoke  active_record
  create  db/migrate/20130208161028_create_cars.rb
```

Go edit the migration.

```
$ rake db:migrate
== CreateCars: migrating =====
== CreateCars: migrated (0.0000s) =====
```

```
$ rake db:version
Current version: 20130208161028
```

```
$ rake db:rollback
== CreateCars: reverting =====
== CreateCars: reverted (0.0000s) =====
```

* sometimes automatic, sometime not!



Enable Labs

Lab 2 Part 2: Migrations

Cloning the Lesson Project

```
$ git clone https://github.com/EnableLabs/rails_training_feb_2013.git
$ cd ./rails_training_feb_2013/week2/car_dealership
# if asked, please 'trust' the .rvmrc file
$ bundle install
```

Useful rake commands

```
$ rake db:migrate
```

```
$ rake db:test:prepare
```

Generating an ActiveRecord migration

```
$ rails generate migration CreatePerson
```

```
create_table "users", :force => true do |t|
  t.string :login
  t.string :email
  t.string :remember_token
  t.string :salt, :crypted_password, :limit => 40
  t.datetime :remember_token_expires_at
  t.timestamps
end
```

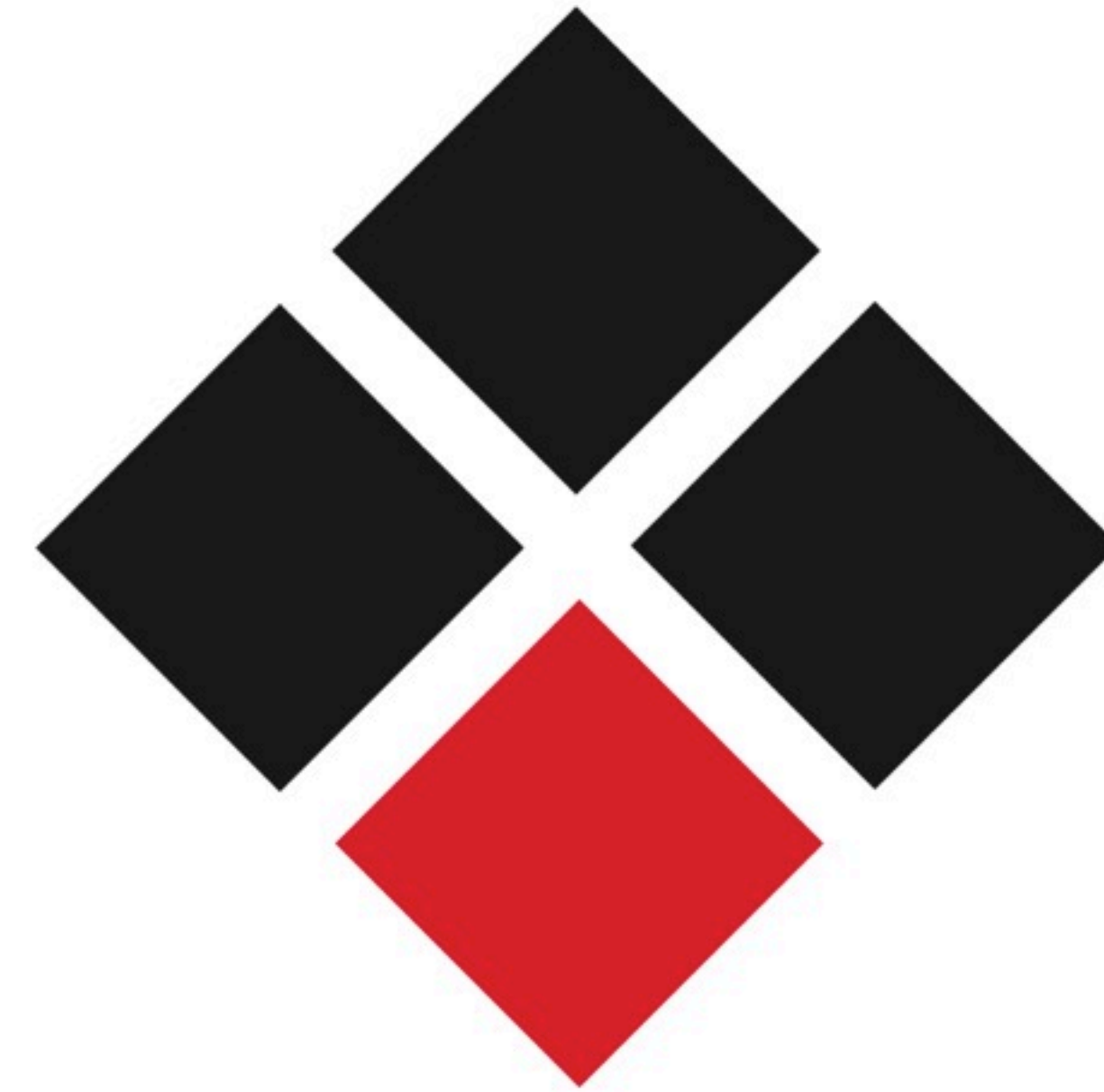
Rails Guides - Useful on-line resource to help you during our lab session

<http://guides.rubyonrails.org/>
- Migrations, Validations, Associations, & Query Interface



ActiveRecord Validations

Ensuring data integrity



Enable Labs

ActiveRecord Validations

- Validations are rules in your model objects to help protect the integrity of your data
- When are validations run?
 - `car.valid?`
 - `car.invalid?`
 - `car.save`
 - `car.update_attributes`
- `car.errors`
 - contains details on all validation errors generated
- `car.valid?`, `car.save`, and `car.update_attributes` return true/false
- `car.save!` and `car.update_attributes!` raise a `RecordInvalid` exception if the object is not valid
- Persisting without validation
 - `car.save(:validate => false)`
 - `car.update_attribute(:make => 'BMW')`



Validation Example: custom &

```
class Person < ActiveRecord::Base
  attr_accessible :name

  validate :person_is_stable, :on => :create

  def validate
    ...
  end

  def person_is_stable
    ...
    some crazy algorithm that calculates stability!
    ...
  end
end
```

**custom
&
on a specific hook
in the lifecycle**



Validation Helpers

- acceptance
- validates_associated `validates :make, :model, :presence => true`
- confirmation
- validates_each `validates :year, :numericality => true`
`validates :year, :numericality => { :only_integer => true }`
- exclusion
- format `validates :vin, :uniqueness => true, :on => :create`
- inclusion
- length
- numericality
- presence
- uniqueness



Validation Macro Examples

```
class User < ActiveRecord::Base
  validates :name, :presence => true
  validates :name, :format => { :with => /^\\w+$/,
                                :message => "may only contain word characters" }
  validates :name, :uniqueness => { :message => "is already in use" }
  validates :password, :length => { :in => 4..40 }
  validates :password, :confirmation => true
  validates :role, :inclusion => { :in => %w(super admin user),
                                :message => "must be super, admin, or user",
                                :allow_nil => true }
  validates :customer_id, :presence => true, :if => Proc.new { |u|
    %w(admin user).include?(u.role)
  }
  validates :weight, :numericality => { :only_integer => true, :allow_nil => true }
end
```



Custom Validations

- Validation fails if errors are not empty

```
class Car < ActiveRecord::Base
  attr_accessible :year, :make, :model
  validate :not_stolen?

  def not_stolen?
    registration = State::Police::CarRegistrationService.find(self.vin)
    errors[:base] << "Car is Stolen!" if registration.car.stolen?
  end

  def validate
    if installed_old_tire?
      errors[:base] << "Tire age exceeds limit"
    end
  end
end
```



Enable Labs

Lab 2 Part 3: Validations

<Rails.root>/app/models/car.rb

```
class Car < ActiveRecord::Base
  validates :name, :presence => true
end
```

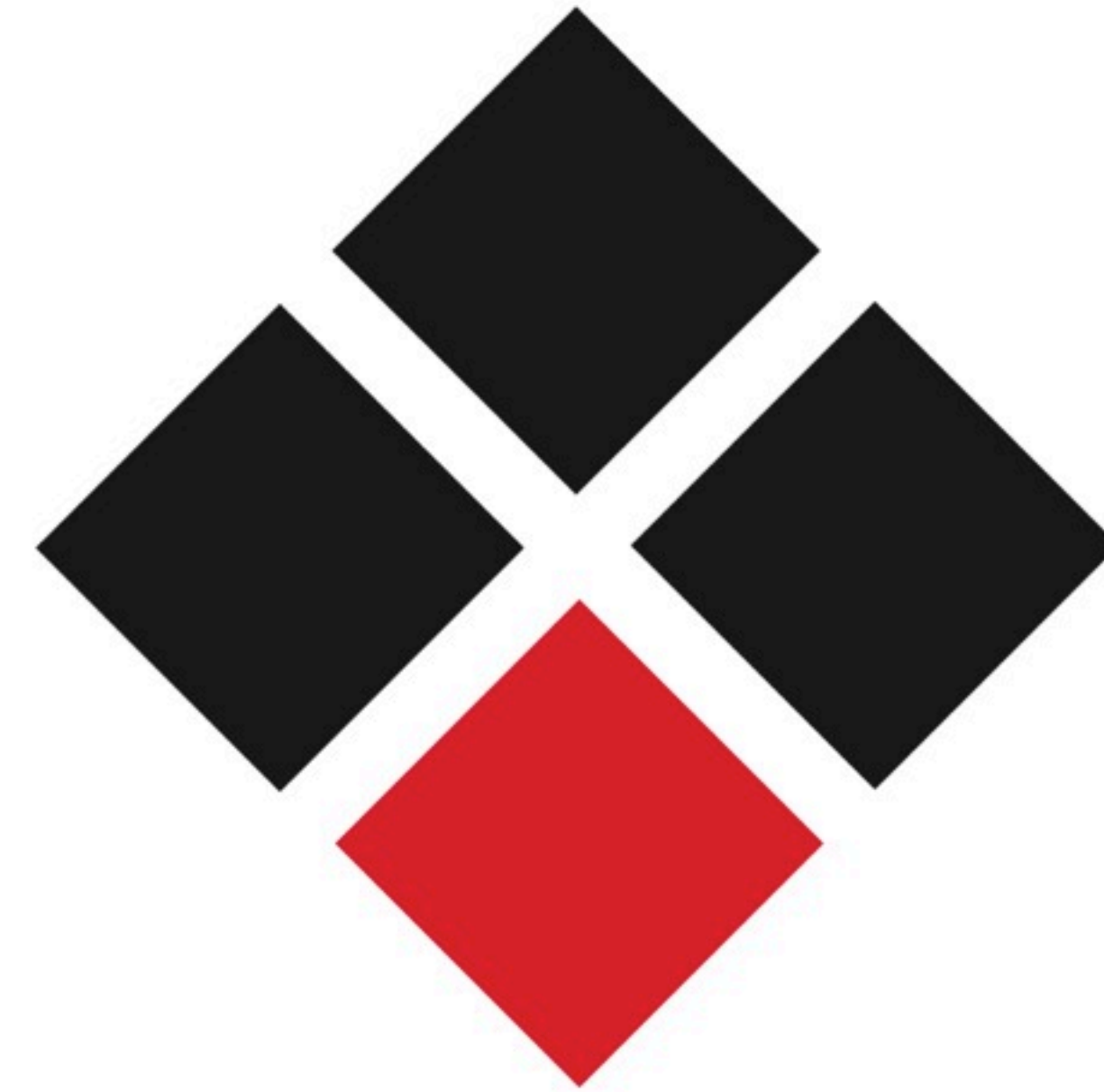
Rails Guides - Useful on-line resource to help you during our lab session

<http://guides.rubyonrails.org/>
- Migrations, Validations, Associations, & Query Interface



ActiveRecord Associations

Joining Things Together



Enable Labs

Association Types

- Defining the relationship between objects through foreign keys
 - **has_one**
 - **has_many**
 - **belongs_to**
 - contains the foreign key field
 - **has_and_belongs_to_many**
 - utilizes a join table with two foreign keys
 - **has_many (:through)**
 - “through” association contains the foreign key field
 - Allows for metadata about the relationship

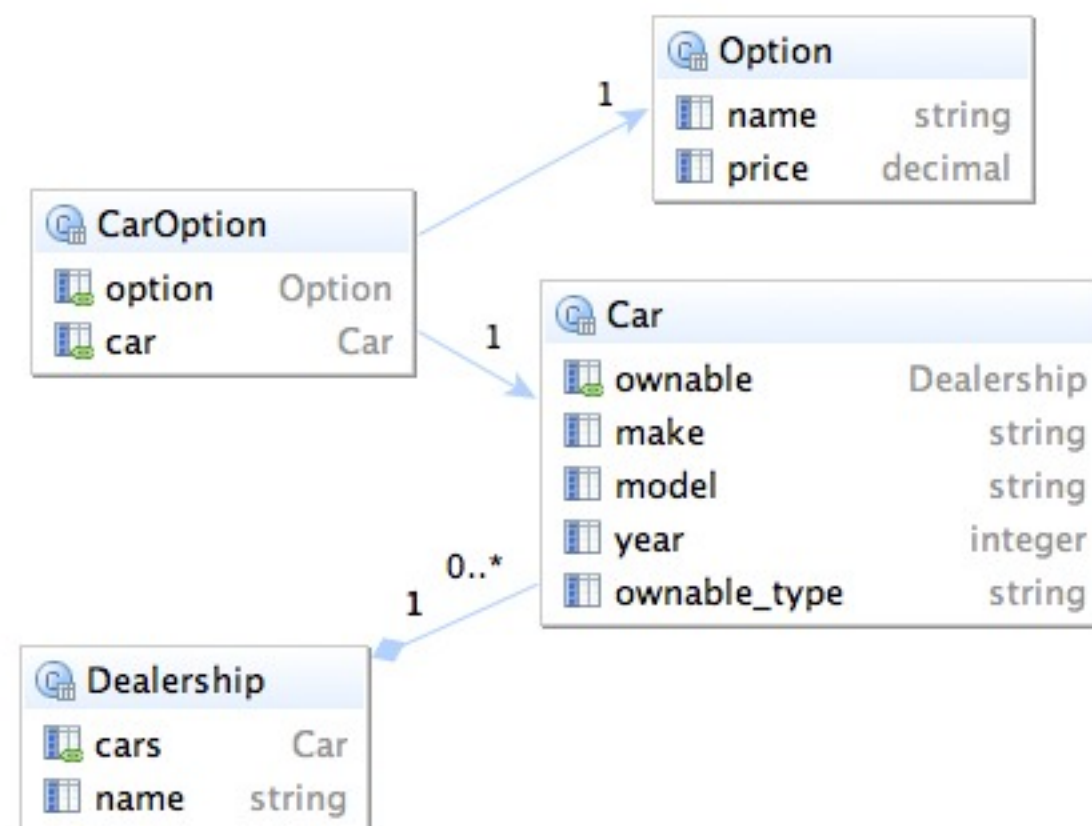
```
class Car < ActiveRecord::Base
  has_one :vehicle_description
  has_many :vehicle_options
end
```

```
car = Car.where(:make => 'BMW').last
puts car.vehicle_description
car.vehicle_options.where(:type => LIGHTING)
```

Look! The query interface used on an association!



ActiveRecord Association: has_one

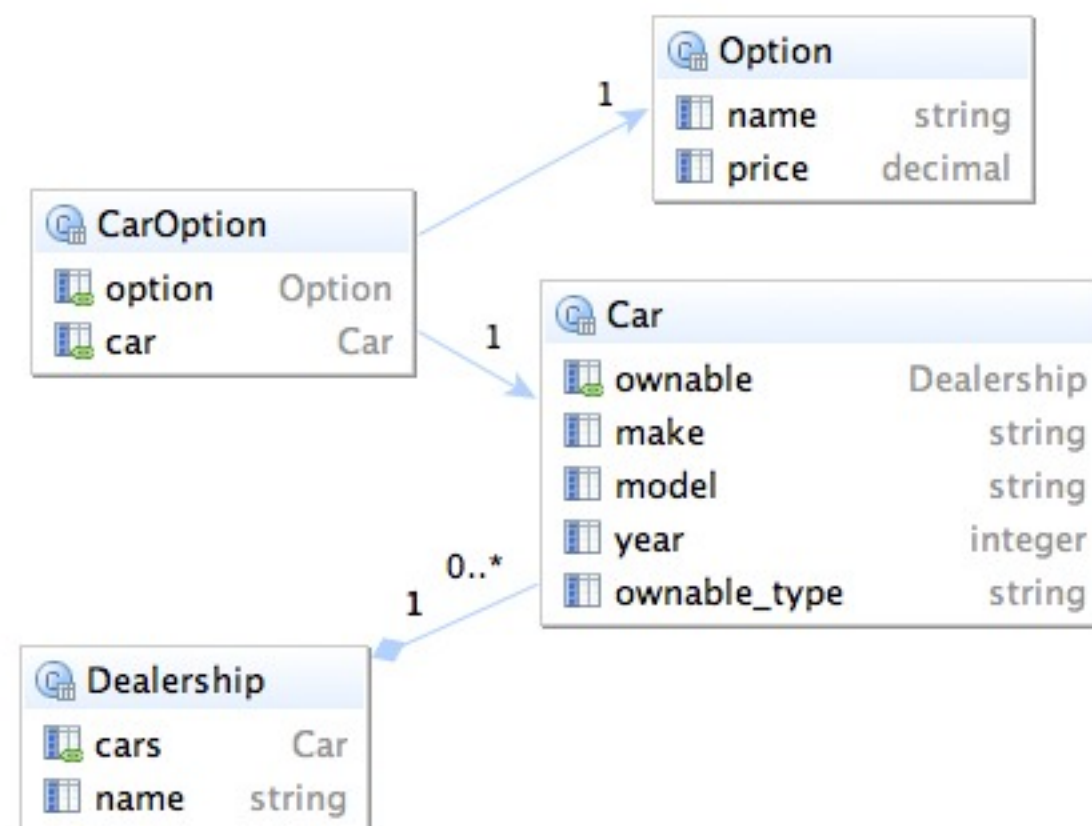


```
class Car < ActiveRecord::Base
  has_one :vehicle_description
end
```

```
class VehicleDescription < ActiveRecord::Base
  belongs_to :car
end
```



ActiveRecord Association: has_many



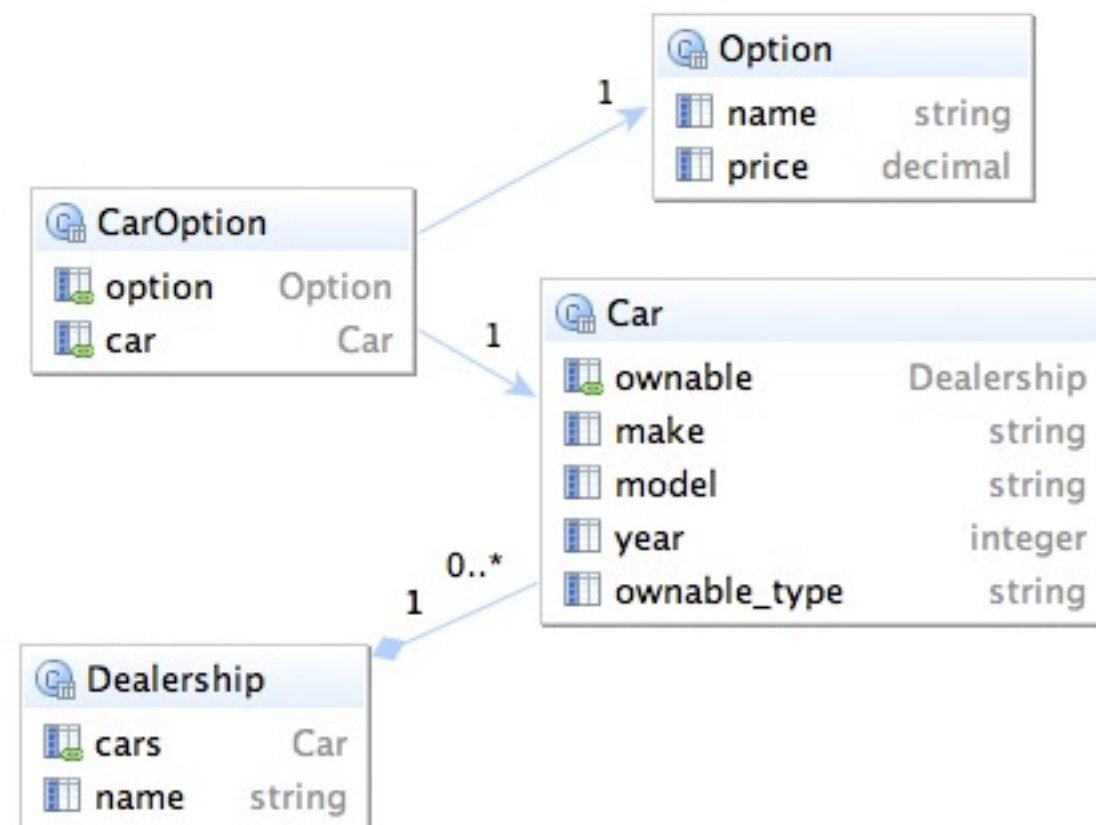
```
class Dealership < ActiveRecord::Base
  has_many :sales_people
end
```

```
class SalesPerson < ActiveRecord::Base
  belongs_to :dealership
end
```



Enable Labs

ActiveRecord Association: has_many through



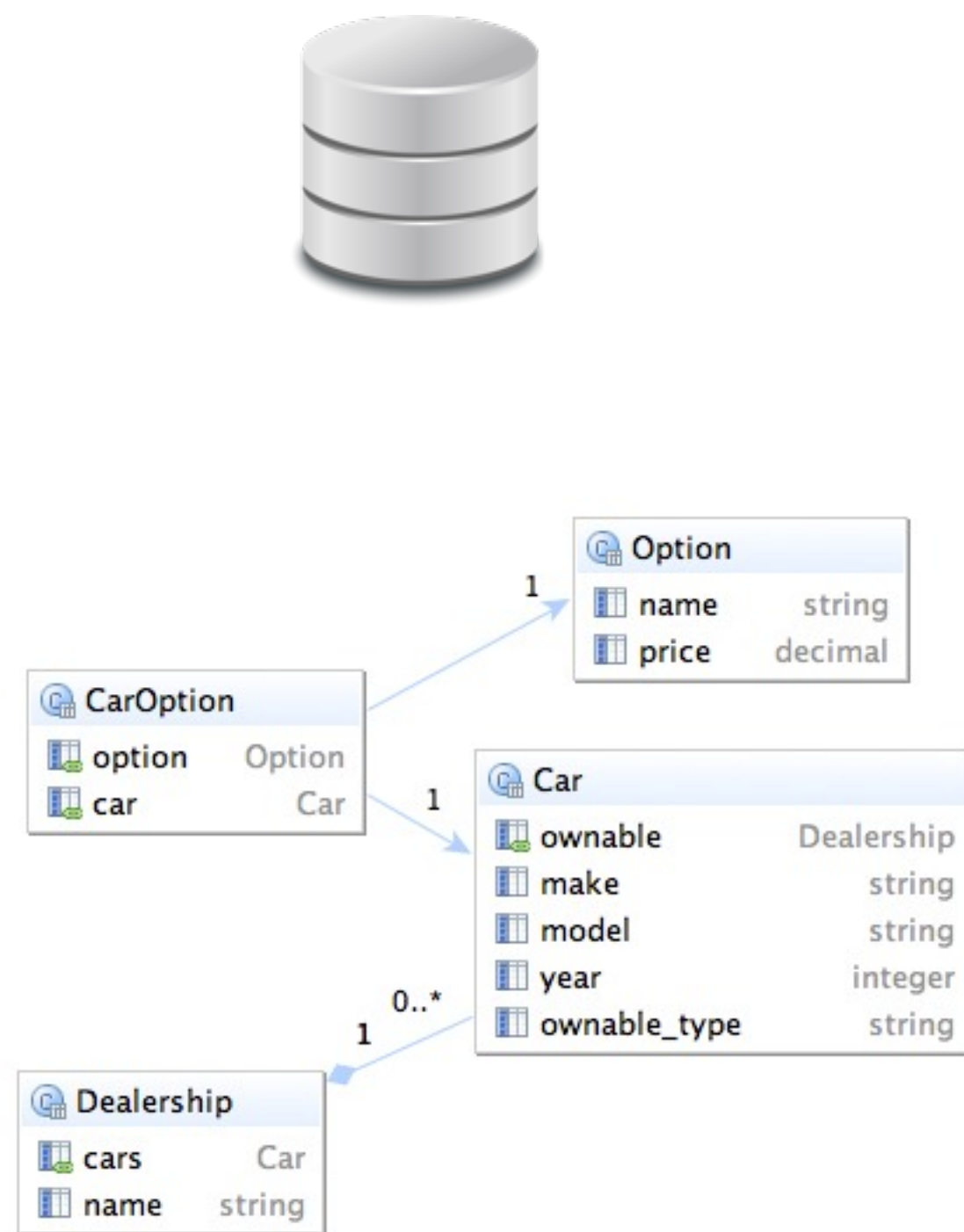
```
class Dealership < ActiveRecord::Base
  has_many :contracts
  has_many :customers, :through => :contracts
end
```

```
class Contract < ActiveRecord::Base
  attr_accessible :terms, :is_signed
  belongs_to :dealership
  belongs_to :customer
end
```

```
class Customer < ActiveRecord::Base
  attr_accessible :name
  has_many :contracts
end
```



ActiveRecord Association: polymorphic



```
class Car < ActiveRecord::Base
  belongs_to :ownable, :polymorphic => true
end
```

```
class Dealership < ActiveRecord::Base
  has_many :cars, :as => :ownable
end
```

```
class Customers < ActiveRecord::Base
  has_many :cars, :as => :ownable
end
```



Enable Labs

Dynamic Association Methods: Creating

- Associations add methods to the class
 - This is an excellent example of meta-programming
- Added methods allow easy management of the associated models.
 - `dealership.cars << Car.new(...)`
 - `dealership.cars.create(...)`
 - `dealership.cars.build(...)`
 - `car.vehicle_description = VehicleDescription.new(...)`
 - `car.create_vehicle_description(...)`
 - `car.build_vehicle_description(...)`

build vs. create



Dynamic Association Methods: Querying

- These are ActiveRecord::Relations therefore...

```
cars.where(:dealership_id => dealership.id, :make => 'BMW')
```

=

```
dealership.cars.where(:make => 'BMW')
```



Enable Labs

Lab 2 Part 4: Associations

```
class Dealership < ActiveRecord::Base
  has_many :sales_people
end
```

```
class SalesPerson < ActiveRecord::Base
  belongs_to :dealership
end
```

Rails Guides - Useful on-line resource to help you during our lab session

<http://guides.rubyonrails.org/>
- Migrations, Validations, Associations, & Query Interface

