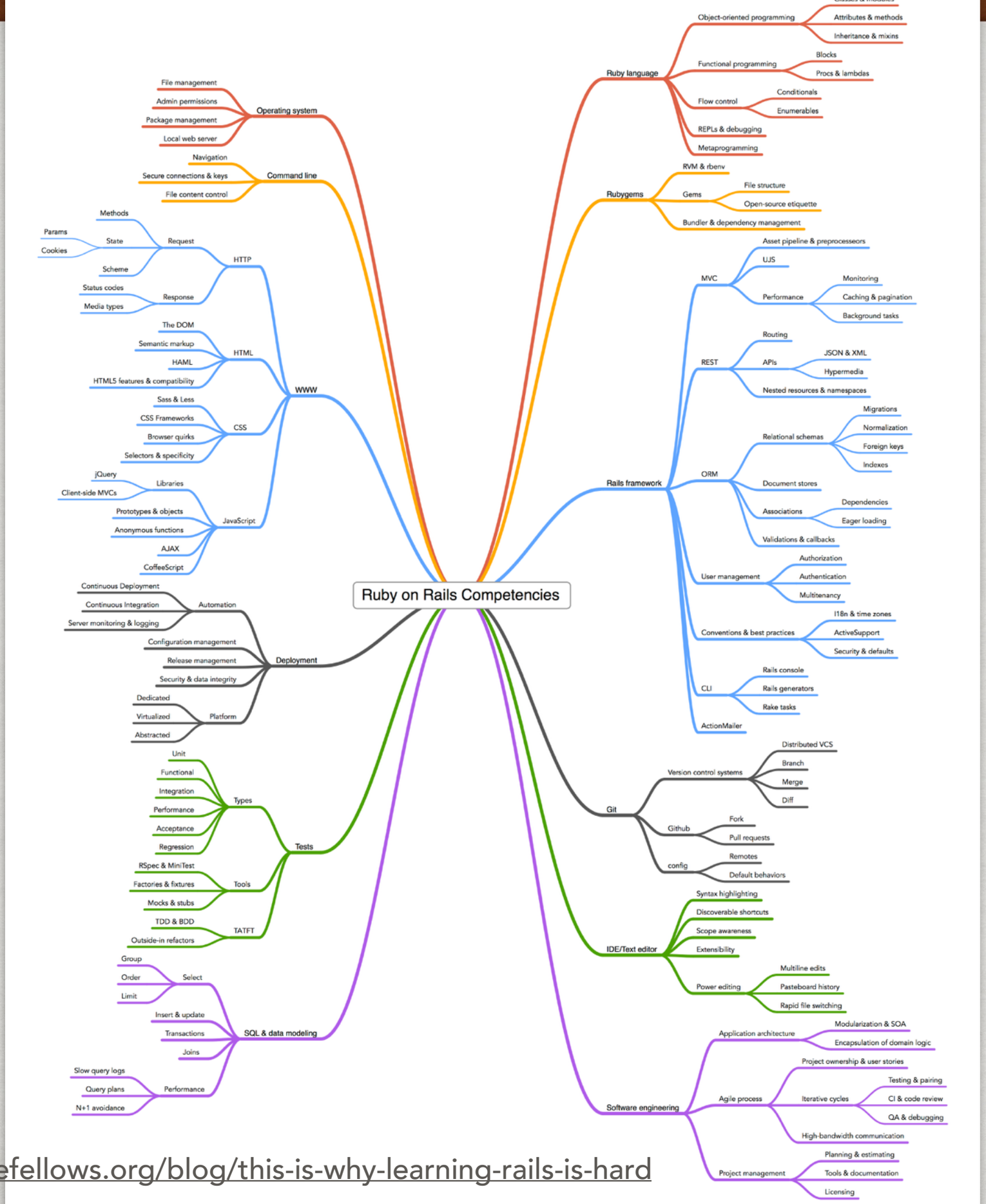
INTRO TO RAILS ACTIVE-RECORD

...IN MINUTES

Ben Leadholm

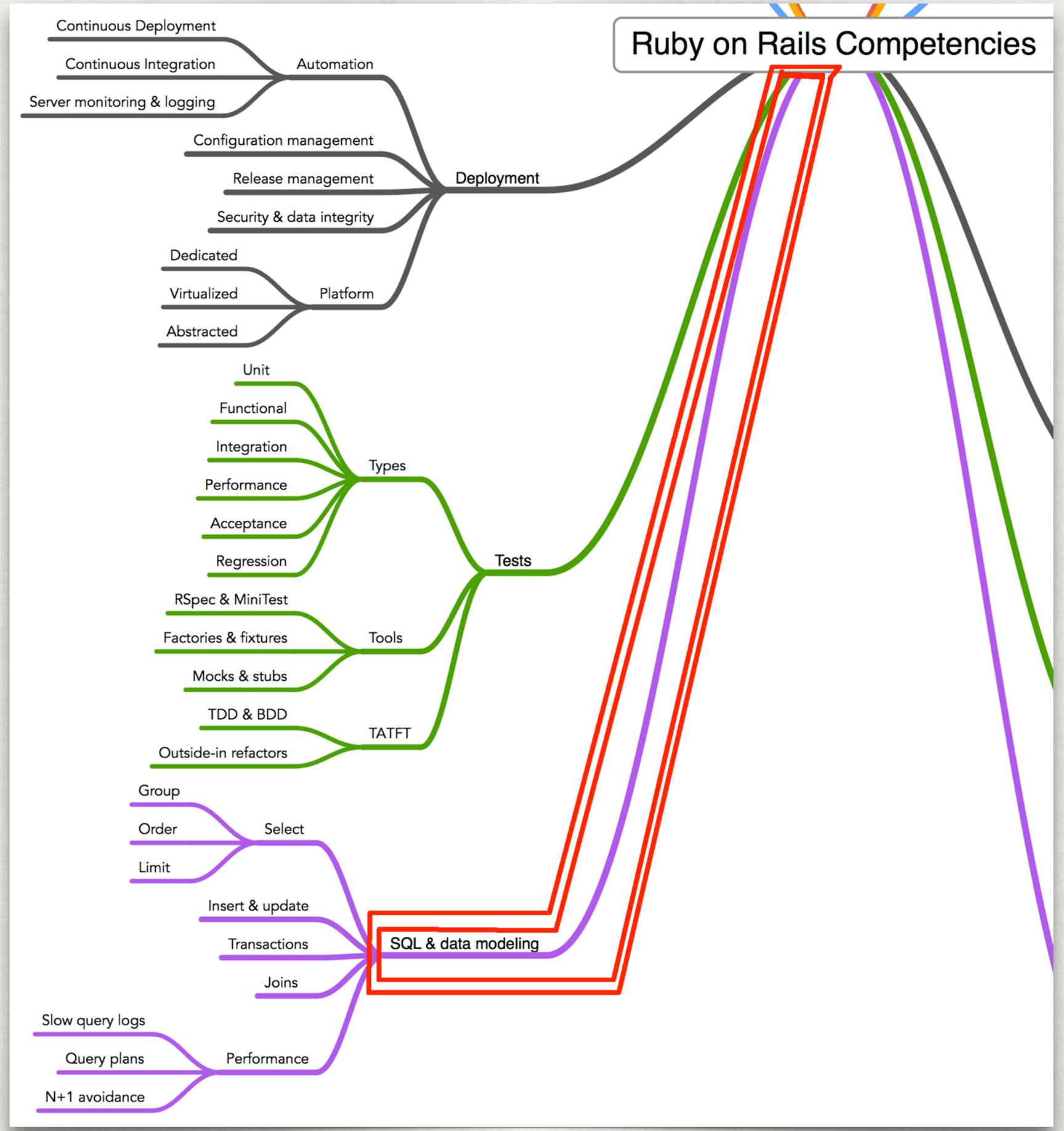
I'M GOING TO FOCUS ON RAILS ACTIVE-RECORD CONVENTIONS

There is
SO MUCH
TO LEARN !!



source: https://www.codefellows.org/blog/this-is-why-learning-rails-is-hard

We will focus on this branch of the octopus



ACTIVE-RECORD

DATABASE CONCEPTS

- A relational database is similar to an Excel workbook, where a table would be the equivalent of a spreadsheet
- Tables have rows (representing individual objects), and columns (representing an object's characteristics)
- Relational databases represent an object through one or more tables that are joined together through identity key columns and foreign key references to the identity key columns
- By breaking up an object's description into multiple tables (through a process called **normalization**, one can accurately describe an object with a minimum amount of disruption when a change must happen

ACTIVE-RECORD

ACTIVE-RECORD HIGH LEVEL CONCEPTS

- ActiveRecord is an Object-Relational Model (ORM) framework used by Rails to represent an object in a relational database
- ActiveRecord, like Rails, follows the ‘convention over configuration’ concept to minimize the amount of code one has to write to see results (known as the ‘happy path’)
- We’ll cover some of these conventions
- The reason why ‘convention over configuration’ works so well is that any Rails developer can open any Rails project and quickly understand where logic resides for views, models, and routing (controllers)

ACTIVE-RECORD

CONVENTION OVER CONFIGURATION

- Q: As of 2013, there are 2.7 million talented, qualified computer programmers in India alone. Why wouldn't companies use this talent over local programming resources to reduce their costs?

  http://www.computerworld.com/article/2483690/it-careers/india-to-overtake-u-s--on-number-of-developers-by-2017.html

  A: In the 2000's they did, but convention over configuration allowed U.S. programmers to be more **productive in a shorter amount of time**, allowing local talent to have a higher Return on Investment (ROI) than offshore developers. Most companies that had spun-off their development overseas brought the projects and work back on-shore, because the cost-per-project was higher for offshore than for on-shore (for now).

- This is no reason for state-side developers to become complacent. Productivity will always be compared, so you should always be learning!

ACTIVE-RECORD
CONVENTIONS

- The object is singular, while the database name is plural (How do I know? Use Code!)

```
require 'active_support/inflector'

"Client".pluralize.underscore
 => “clients"

"LineItem".pluralize.underscore
 => “line_items"

"Address".pluralize.underscore
 => “addresses"

“Octopus".pluralize
 => “Octopi"

"BlueOctopus".pluralize.underscore
 => “blue_octopi"

"Goose".pluralize
 => "Gooses"

"Mouse".pluralize
 => "Mice"
```

Object Name	Table Name
Client	clients
LineItem	line_items
Address	addresses
Octopus	octopi
BlueOctopus	blue_octopi
Goose	gooses ?!? (What about this?)
Mouse	mice

ACTIVE-RECORD

CONVENTIONS

- There are exceptions!

```
require 'active_support/inflector'

"Goose".pluralize
 => "Gooses"

"Bonus".pluralize
 => "Bonus"
```

Object Name	Ruby Plural	Ideal
Goose	Gooses	Geese
Bonus	Bonus	Bonuses

The way to work around this is to set the plural to what you want it to be…

```
ActiveSupport::Inflector.inflections do |inflect|
  inflect.irregular 'bonus', 'bonuses'
end
```

http://softwareas.com/bitten-by-rails-pluralizationinflection/

http://stackoverflow.com/questions/3517989/ruby-on-rails-how-do-you-explicitly-define-plural-names-and-singular-names-in-r

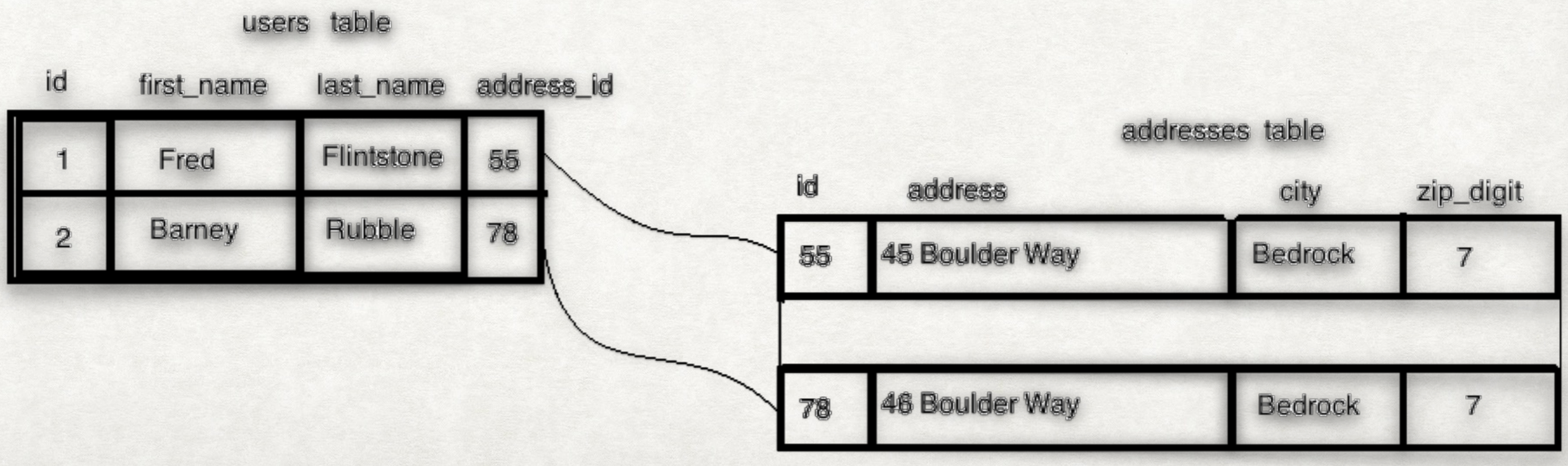
http://strugglingwithruby.blogspot.com/2010/03/rails-singularize-and-pluralize.html

ACTIVE-RECORD

CONVENTIONS

- Records in large tables will have an identity column with a column name of ‘id’

- This is true of all large tables, with a reference column from one table pointing to the other (known as a foreign-key)
  For example, if a Client has an Address, the clients table will have an id column, and an address_id column that has the id value from the addresses table.



ACTIVE-RECORD

CONVENTIONS

Code for User object

```
/app/models/user.rb
class User < ActiveRecord::Base
    has_many :addresses
end
```

```
/db/migrations/20151213135543_create_users.rb
class CreateUsers < ActiveRecord::Migration
    change
        create_table :users do |t|
            t.string    :first_name
            t.string    :last_name
            t.integer   :address_id

            t.timestamps  null: false
        end
    end
end
```

Code for Address object

```
/app/models/address.rb
class Address < ActiveRecord::Base
    belongs_to :user
end
```

```
/db/migrations/20151213140000_create_addresses.rb
class CreateAddresses < ActiveRecord::Migration
    change
        create_table :addresses do |t|
            t.string    :address
            t.string    :city
            t.integer   :zip_digit
            t.belongs_to :user, index: true

            t.timestamps  null: false
        end
    end
end
```

ACTIVE-RECORD

CONVENTIONS

http://guides.rubyonrails.org/association_basics.html

- Rails will model the following relations by default:
  - belongs_to

```
class Order < ActiveRecord::Base
  belongs_to :customer
end
```

  - has_one

```
class Supplier < ActiveRecord::Base
  has_one :account
end
```

  - has_many

```
class Customer < ActiveRecord::Base
  has_many :orders
end
```

ACTIVE-RECORD

CONVENTIONS

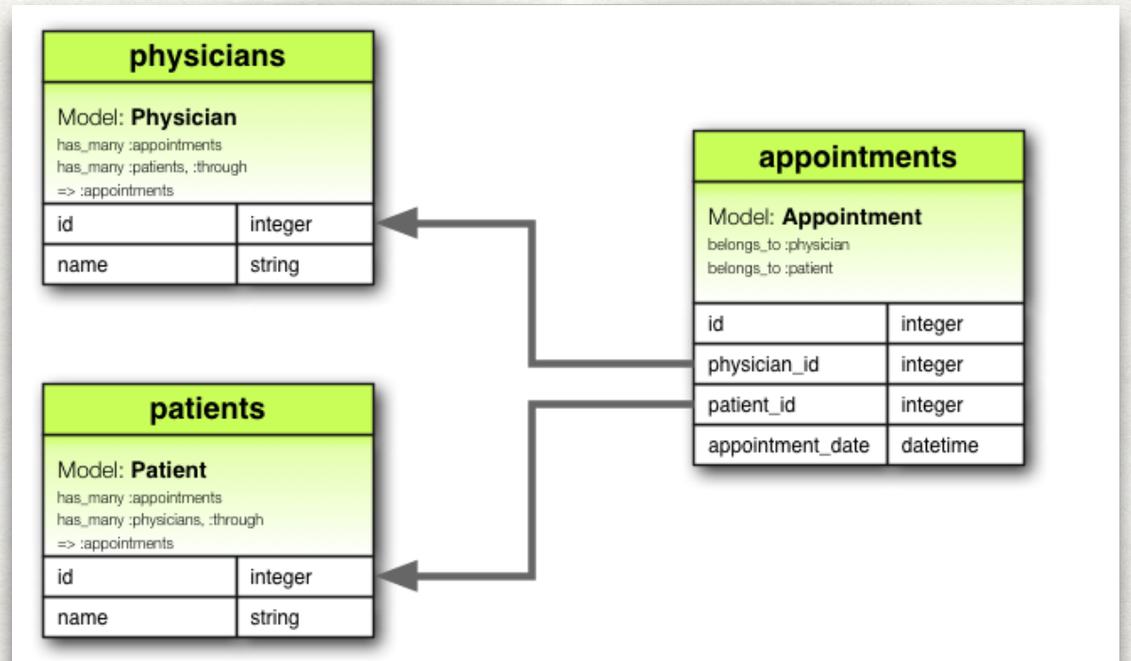
http://guides.rubyonrails.org/association_basics.html

- Rails will model the following relations by default:
  - has_many :through

```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, through: :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, through: :appointments
end
```




```
class Physician < ActiveRecord::Base
  has_many :appointments
  has_many :patients, :through => :appointments
end

class Appointment < ActiveRecord::Base
  belongs_to :physician
  belongs_to :patient
end

class Patient < ActiveRecord::Base
  has_many :appointments
  has_many :physicians, :through => :appointments
end
```

ACTIVE-RECORD

CONVENTIONS

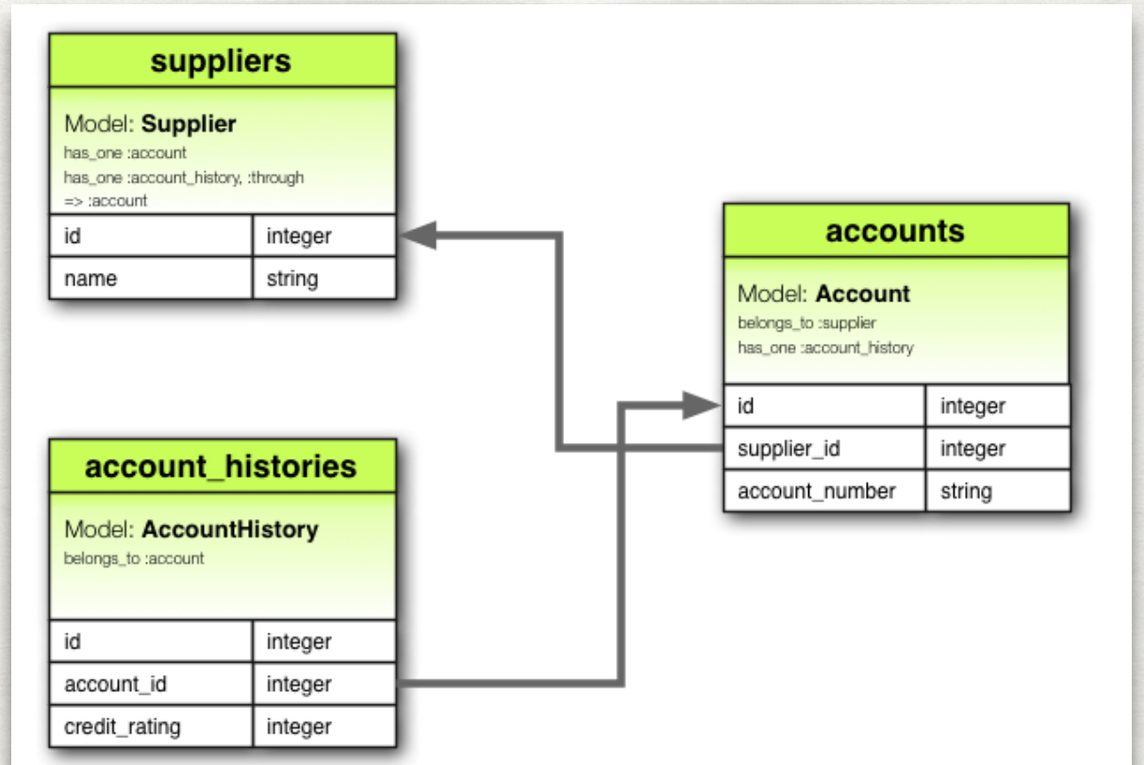
http://guides.rubyonrails.org/association_basics.html

- Rails will model the following relations by default:
  - has_one :through

```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, through: :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```




```
class Supplier < ActiveRecord::Base
  has_one :account
  has_one :account_history, :through => :account
end

class Account < ActiveRecord::Base
  belongs_to :supplier
  has_one :account_history
end

class AccountHistory < ActiveRecord::Base
  belongs_to :account
end
```

ACTIVE-RECORD

CONVENTIONS

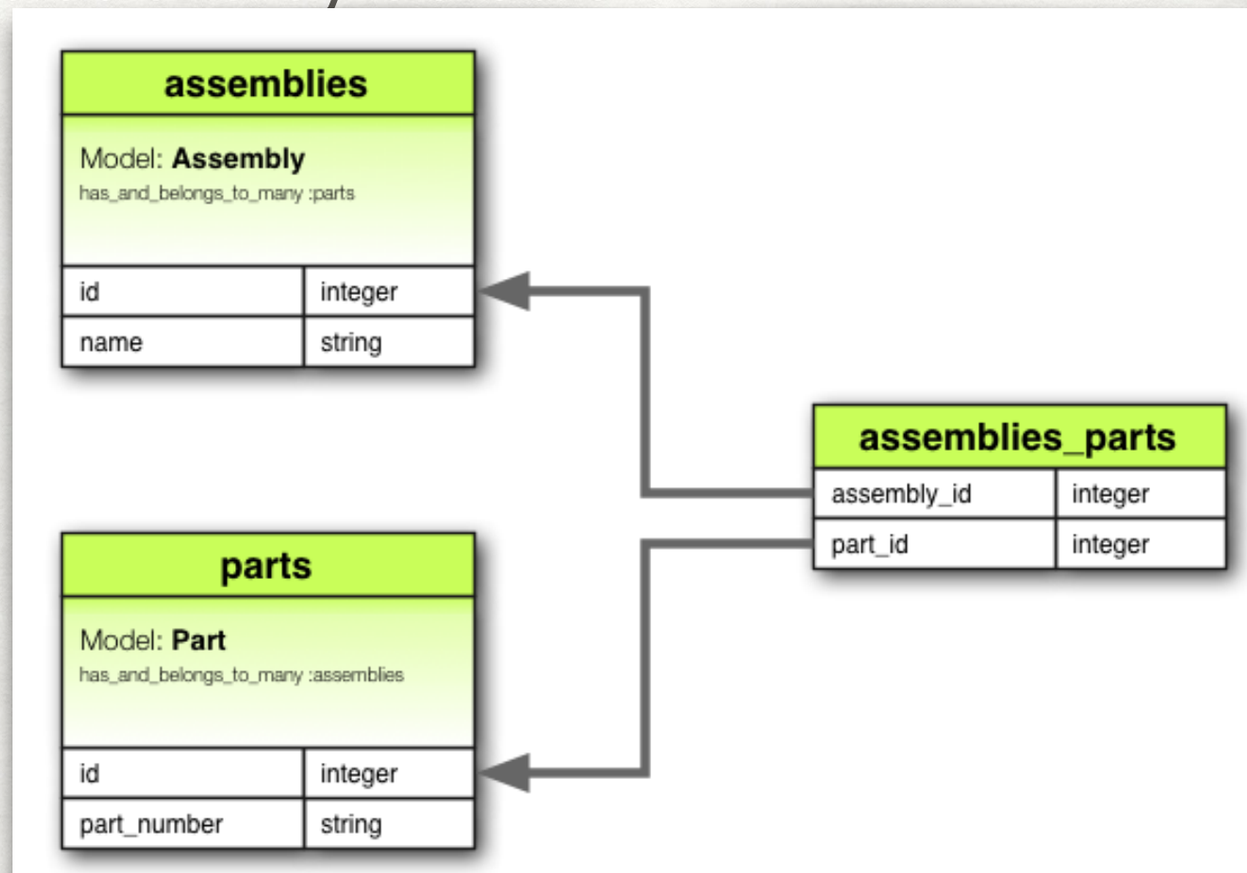
http://guides.rubyonrails.org/association_basics.html

- Rails will model the following relations by default:
  - has_and_belongs_to_many

```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

NOTE: the link table is in alphabetical order, so use assemblies_parts NOT parts_assemblies




```
class Assembly < ActiveRecord::Base
  has_and_belongs_to_many :parts
end

class Part < ActiveRecord::Base
  has_and_belongs_to_many :assemblies
end
```

ACTIVE-RECORD

CONVENTIONS

http://api.rubyonrails.org/classes/ActiveRecord/Inheritance.html

- Rails will model the following relations by default:
  - Single table inheritance (STI)

```
class Company < ActiveRecord::Base; end
class Firm < Company; end
class Client < Company; end

Firm.create( name: 'Target Corporation')
```

NOTE: this record will be saved in the **companies** table, with a **type** column value of 'Firm'

```
obj = Company.where( name: 'Target Corporation').first
obj.is_a?(Firm)  => true
```

http://eewang.github.io/blog/2013/03/12/how-and-when-to-use-single-table-inheritance-in-rails/

MORE INFO...

might be a little dated, but I find that if I want information on anything in Rails, this website is a good place to start

http://railscasts.com/episodes/202-active-record-queries-in-rails-3

http://guides.rubyonrails.org/active_record_basics.html

http://guides.rubyonrails.org/association_basics.html

http://api.rubyonrails.org/classes/ActiveRecord/Inheritance.html

http://eewang.github.io/blog/2013/03/12/how-and-when-to-use-single-table-inheritance-in-rails/

http://stackoverflow.com/questions/2446156/is-there-a-pluralize-function-in-ruby-not-rails

https://www.codefellows.org/blog/this-is-why-learning-rails-is-hard

Questions?

THANKS !

Ben Leadholm
bclead3@yahoo.com
or bigbenny3@gmail.com
@bgbn3 da twitter