

Ruby on Rails

Class #3...

- Review Lab 1
- Database Terminology
- Migration
- Unit Testing
- Joins and Relationships

Review Lab 1

- Too hard? Too easy?
- What didn't you understand?



Database Terminology

- **MySQL is a database server**
 - A database server is called an instance

- **MySQL is a database server**
 - A database server is called an instance
- **A database instance has databases**

Database Terminology

- **MySQL is a database server**
 - A database server is called an instance
- **A database instance has databases**
- **A database has tables**

- **MySQL is a database server**
 - A database server is called an instance
- **A database instance has databases**
- **A database has tables**
- **A table has columns and indexes**

- **MySQL is a database server**
 - A database server is called an instance
- **A database instance has databases**
- **A database has tables**
- **A table has columns and indexes**
- **A column has a name and a type**

- create database course_development
- **Creates a database called course_development**
- create database course_test
- **Now we have two databases in our MySQL instance**
 - one for development
 - one for testing

- **Create table...**
 - Specify the columns

- Create table...
 - Specify the columns

```
CREATE TABLE `courses` (  
  `id` int(11) NOT NULL auto_increment,  
  `course_number` varchar(255) NOT NULL,  
  `description` text,  
  `professor` varchar(255) NULL,  
  `credits` int(11) NULL,  
  PRIMARY KEY (`id`)  
);
```

- Create table...
 - Specify the columns

```
CREATE TABLE `courses` (  
  `id` int(11) NOT NULL auto_increment,  
  `course_number` varchar(255) NOT NULL,  
  `description` text,  
  `professor` varchar(255) NULL,  
  `credits` int(11) NULL,  
  PRIMARY KEY (`id`)  
);
```

Creates a table
named courses

- Create table...
 - Specify the columns

```
CREATE TABLE `courses` (  
  `id` int(11) NOT NULL auto_increment,  
  `course_number` varchar(255) NOT NULL,  
  `description` text,  
  `professor` varchar(255) NULL,  
  `credits` int(11) NULL,  
  PRIMARY KEY (`id`)  
);
```

Creates an
id column that sets its
value

- Create table...
 - Specify the columns

```
CREATE TABLE `courses` (  
  `id` int(11) NOT NULL auto_increment,  
  `course_number` varchar(255) NOT NULL,  
  `description` text,  
  `professor` varchar(255) NULL,  
  `credits` int(11) NULL,  
  PRIMARY KEY (`id`)  
);
```

Creates a column of type varchar

- Create table...
 - Specify the columns

```
CREATE TABLE `courses` (  
  `id` int(11) NOT NULL auto_increment,  
  `course_number` varchar(255) NOT NULL,  
  `description` text,  
  `professor` varchar(255) NULL,  
  `credits` int(11) NULL,  
  PRIMARY KEY (`id`)  
);
```

- **SQL types**
 - int
 - decimal
 - char
 - varchar
 - text
 - boolean
 - date
 - datetime
 - Read about the rest...

- Creating a table using the rails migration

```
class Courses < ActiveRecord::Migration
  def self.up
    create_table :courses do |t|
      t.column :course_number, :string
      t.column :description, :text
      t.column :professor, :string
      t.column :credits, :integer
    end
  end

  def self.down
    drop_table :courses
  end
end
```

- **Three Databases and Environments**
 - Development
 - Testing
 - Production

- Defaults to `<project>_development`
 - The database used while developing the application
- Note: When running with the development environment, the model, controller, routes, and views are reloaded every time the page loads
- Do NOT use development environment when you deploy in production

- Defaults to `<project>_production`
 - The database used in the production environment
- Optimized caching and only loads once

- Defaults to `<project>_test`
 - Special database
 - Gets recreated every time rake is used to run the tests
 - Generates the schema from the development db
 - Creates the test database from the generated schema
 - Data is removed and created in the various tests
 - Only use for testing

- **Unit tests validate the model is working as expected**

- Scaffolding created a testing stub for the Course model

```
require File.dirname(__FILE__) + '/../test_helper'

class CourseTest < Test::Unit::TestCase
  fixtures :courses

  # Replace this with your real tests.
  def test_truth
    assert true
  end
end
```

- A fixture has sample data used for testing
 - The fixture statement in the test clears the table

```
# Read about fixtures at http://ar.rubyonrails.org/  
classes/Fixtures.html
```

```
first:
```

```
  id: 1
```

```
  course_number: cs198
```

```
  description: Ruby on rails
```

```
  professor: Armando Fox
```

```
  credits: 1
```

```
second:
```

```
  id: 2
```

```
  description: Invalid record
```

- Let's make sure we can't save if the course number is not specified

```
require File.dirname(__FILE__) + '/../test_helper'  
  
class CourseTest < Test::Unit::TestCase  
  fixtures :courses  
  
  def test_courses_with_number  
    assert courses(:first).clone.save  
  end  
  
  def test_courses_without_number  
    assert !courses(:second).clone.save  
  end  
end
```

- Let's make this work

code

```
class Course < ActiveRecord::Base
  validates_presence_of :course_number
end
```

Functional Tests

- **Functional tests validate the controller is working as expected**
- **Only one controller can be tested at a time**

Integration Tests

- Integration tests span multiple controllers and test workflows
- Selenium does a much better job
- In browser workflow testing with DSL
- We'll revisit when after we discuss views

- Now the functional test breaks

```
...
def test_create
  num_courses = Course.count

  post :create, :course => { :course_number => 'cs198' }

  assert_response :redirect
  assert_redirected_to :action => 'list'

  assert_equal num_courses + 1, Course.count
end
...
```



Fixed it

- Better testing framework than RUnit
- Provides a natural DSL language
- Does a great job on the *Model* and *Controller*
- Use Selenium when you want to test the *view*

- Check out the RSpec gem and plugin

```
require File.dirname(__FILE__) + '/../spec_helper'

describe Course, "with fixtures loaded" do
  fixtures :courses

  it "should not be valid without course number" do
    course = Course.new
    course.should_not be_valid
  end

  it "should be valid with course number" do
    course = Course.new(:course_number => 'cs198')
    course.should be_valid
  end
end
```

- **Introduction to Associations**
 - Belongs to
 - Has one
 - Has many

Belongs To/Has One

- Things that belong to other things

Belongs To/Has One

- Things that belong to other things



name: Santa's Little Helper

Belongs To/Has One

- Things that belong to other things



name: Santa's Little Helper



name: Bart

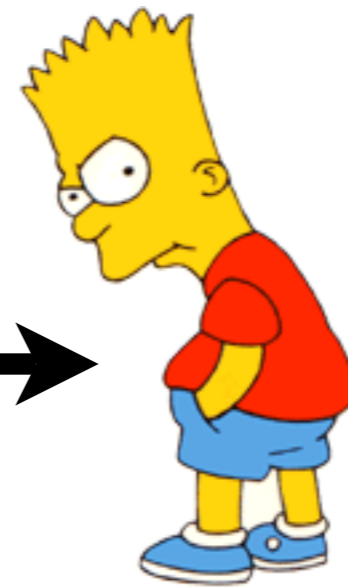
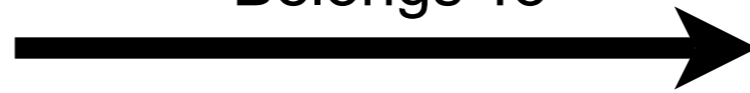
Belongs To/Has One

- Things that belong to other things



name: Santa's Little Helper

Belongs To



name: Bart

Belongs To/Has One

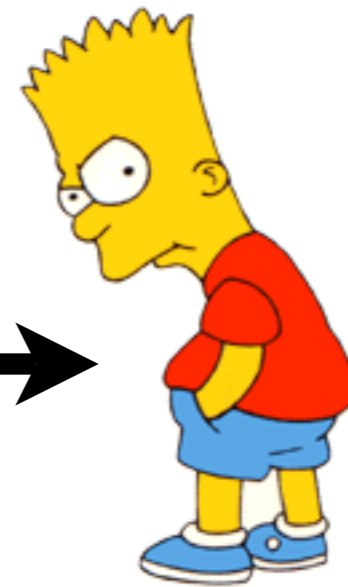
- Things that belong to other things



name: Santa's Little Helper



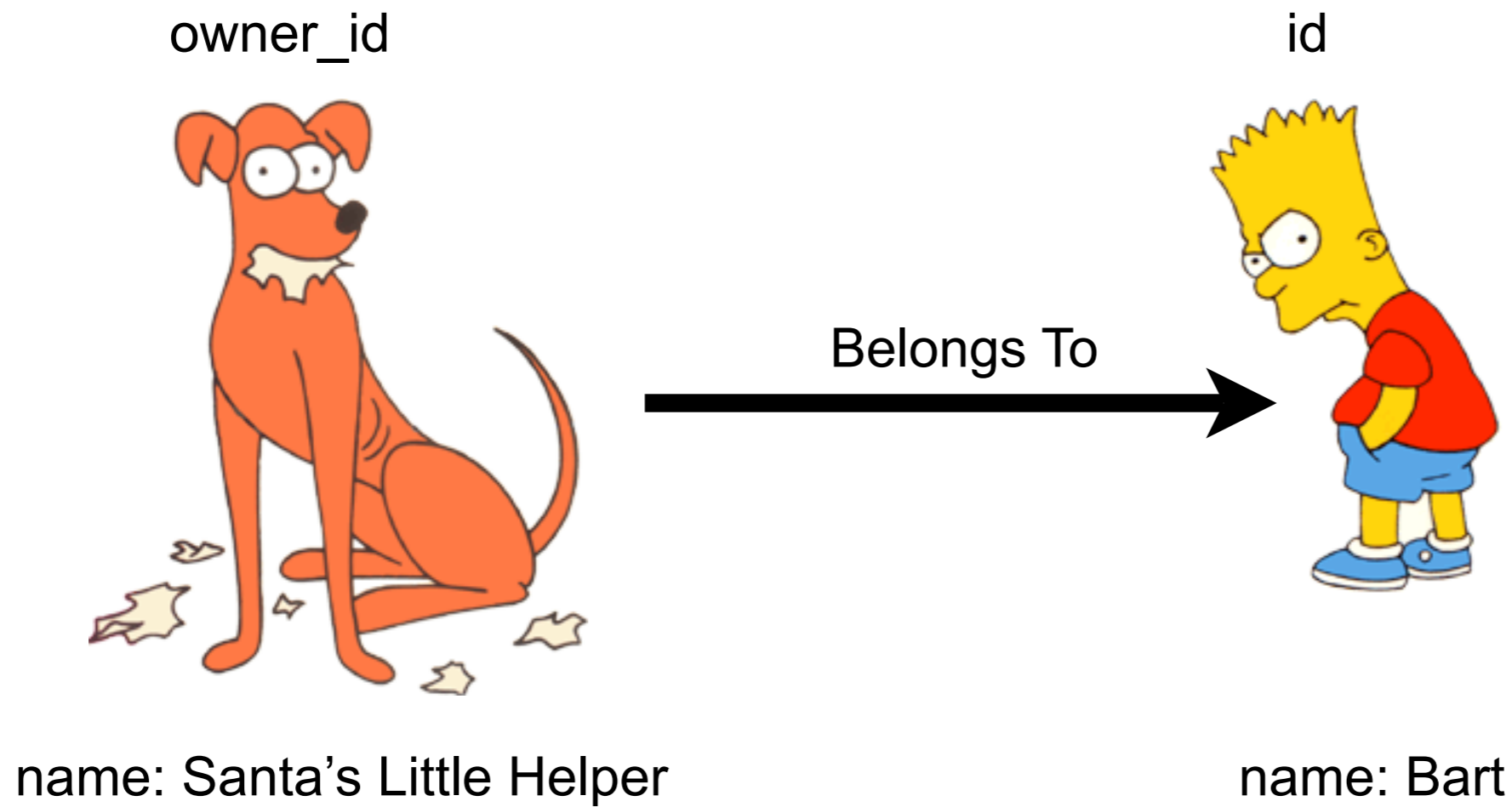
id



name: Bart

Belongs To/Has One

- Things that belong to other things



Belongs To/Has One

- Things that belong to other things

owner_id



name: Santa's Little Helper

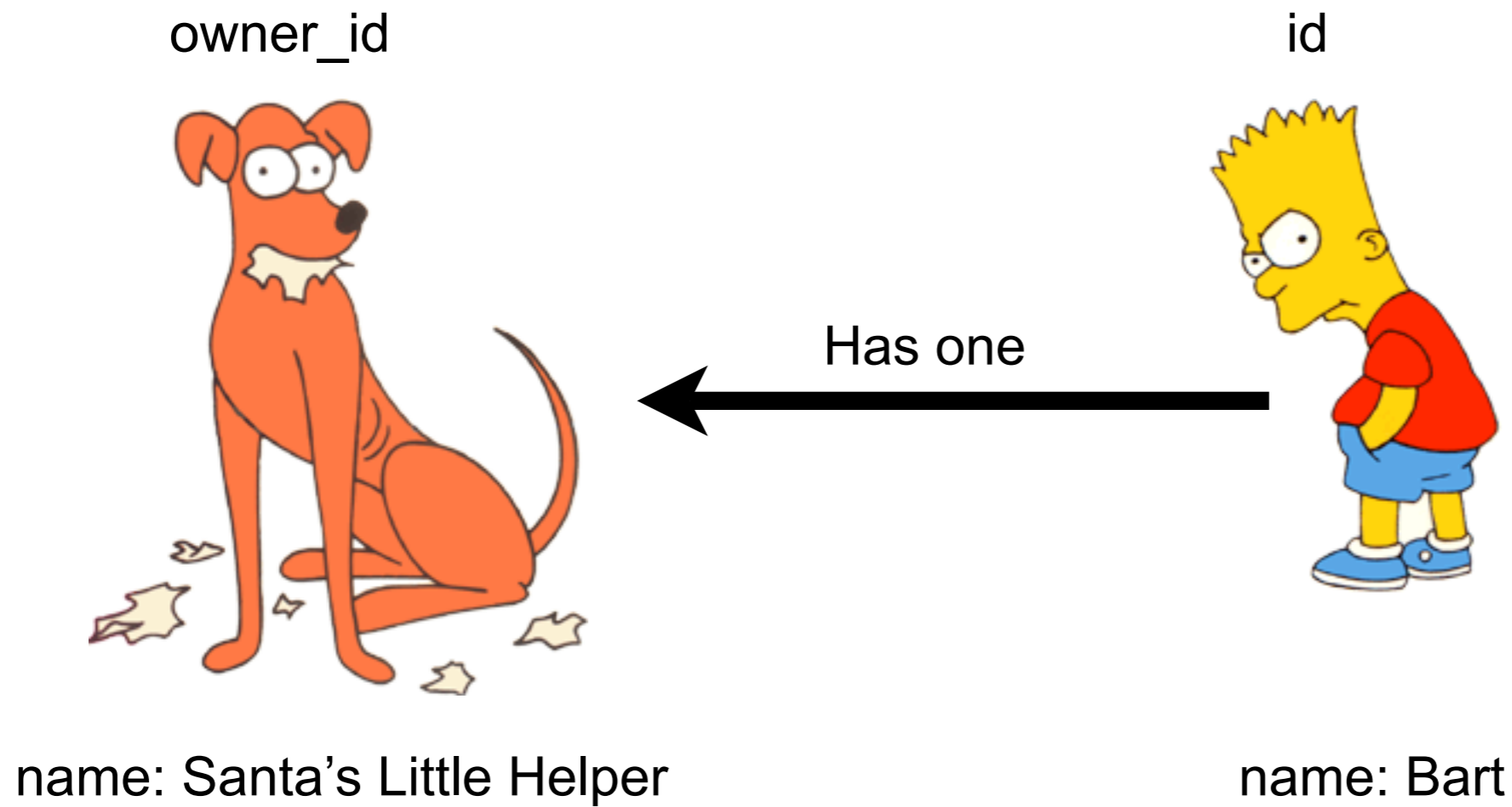
id



name: Bart

Belongs To/Has One

- Things that belong to other things





Belongs To/Has Many

- **Though the miracle of cloning**

Belongs To/Has Many

- Though the miracle of cloning



name: Santa's Little Helper

Belongs To/Has Many

- Though the miracle of cloning



name: Santa's Little Helper



name: Bart

Belongs To/Has Many

- Though the miracle of cloning



name: Santa's Little Helper

id



name: Bart

Belongs To/Has Many

- Though the miracle of cloning

owner_id



name: Santa's Little Helper

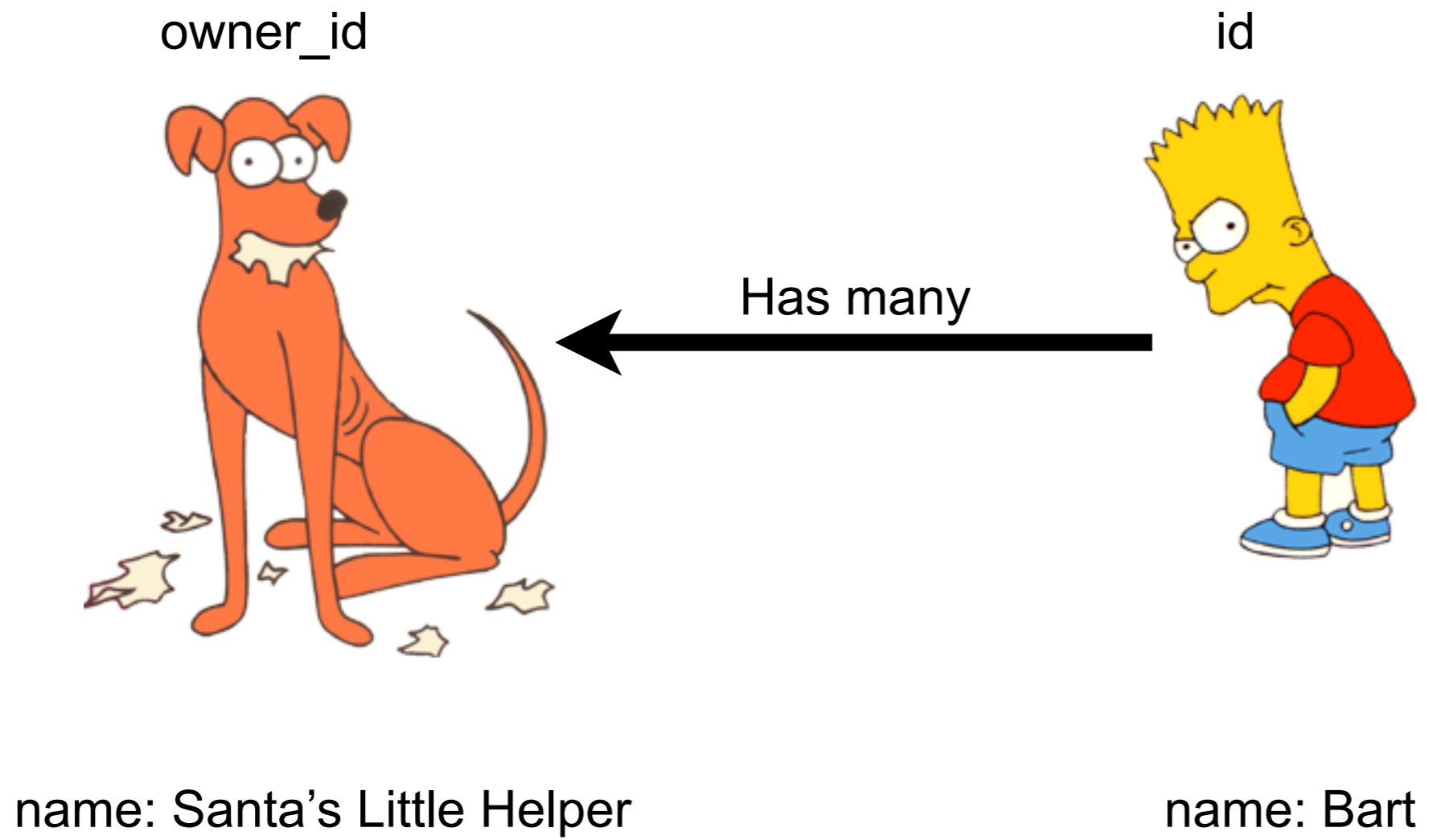
id



name: Bart

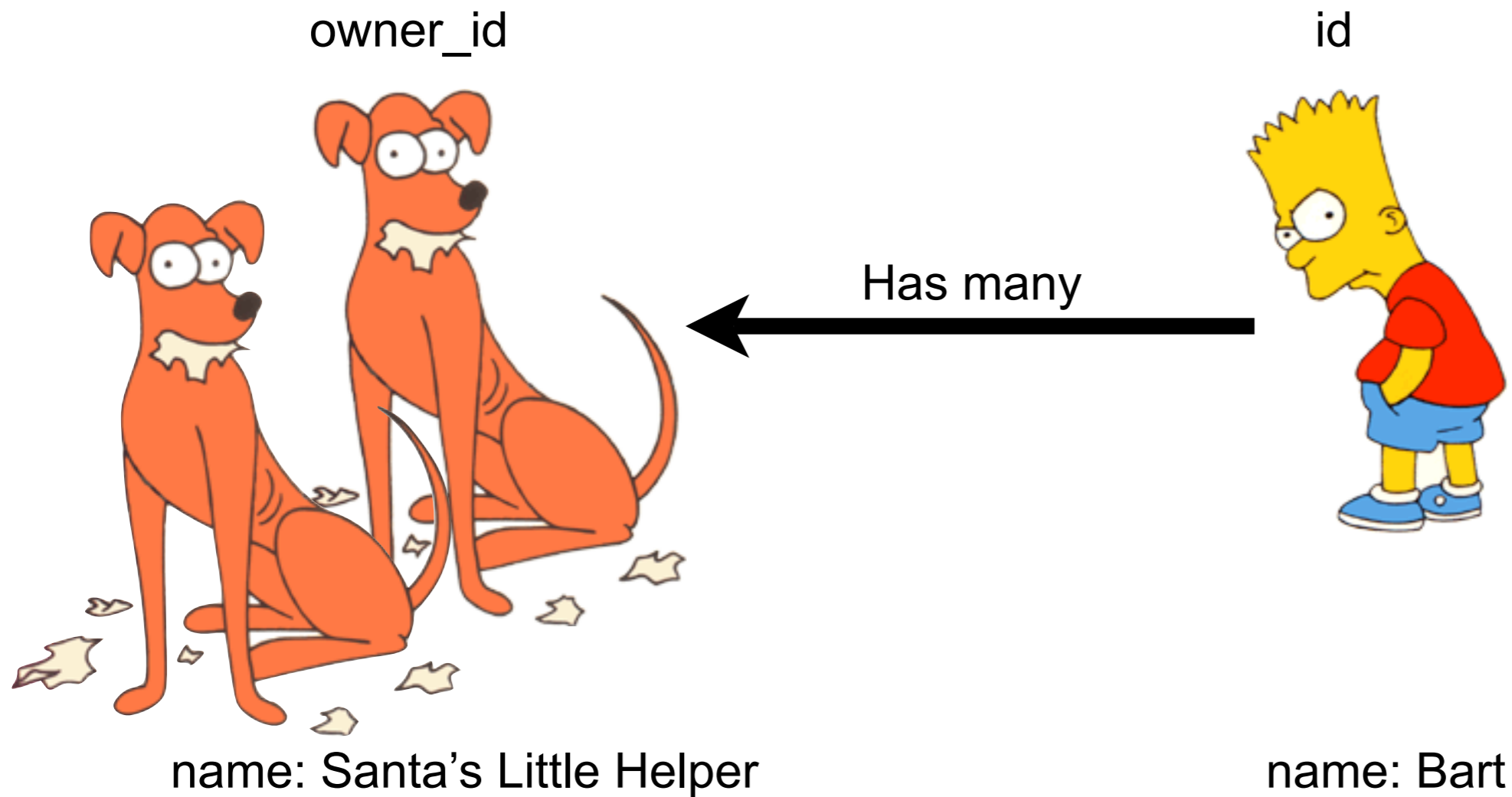
Belongs To/Has Many

- Though the miracle of cloning



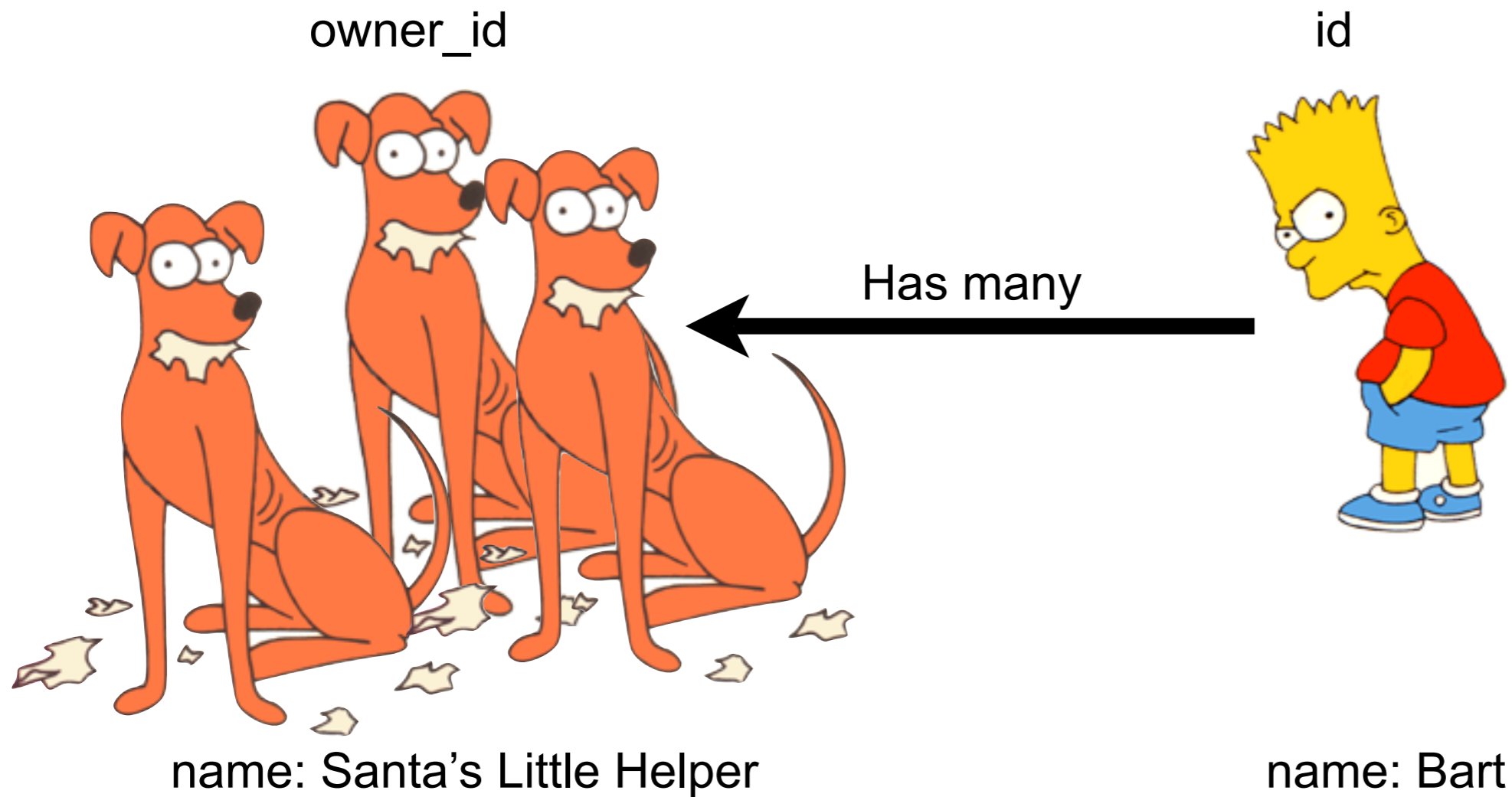
Belongs To/Has Many

- Though the miracle of cloning



Belongs To/Has Many

- Though the miracle of cloning



- Create the tables

```
class Initial < ActiveRecord::Migration
  def self.up
    create_table :owners do |t|
      t.column :name, :string
    end

    create_table :dogs do |t|
      t.column :name, :string
      t.column :owner_id, :integer
    end
  end

  def self.down
    drop_table :dogs
    drop_table :owners
  end
end
```

People and Dogs

```
class Owner
  has_one :dog
end
```

```
class Dog
  belongs_to :owner
end
```

```
class Owner
  has_many :dogs
end
```

An Owner and his Dog

```
class Owner
  has_one :dog
end
```

```
class Dog
  belongs_to :owner
end
```

```
class Owner
  has_many :dogs
end
```

An Owner and his Dog

```
class Owner
  has_one :dog
end
```

A Dog and its Owner

```
class Dog
  belongs_to :owner
end
```

```
class Owner
  has_many :dogs
end
```

An Owner and his Dog

```
class Owner
  has_one :dog
end
```

A Dog and its Owner

```
class Dog
  belongs_to :owner
end
```

Owner with Dogs

```
class Owner
  has_many :dogs
end
```

Using the Associations

- Three ways to create an association
 - Creating a dog and assigning an owner
 - Assigning the dog for the person
 - For has many add the dog to the dogs collection

```

owner = Owner.new(:name => 'Bart')
dog = Dog.new(:name => "Santa's Little Helper", :owner => owner)

# or for has one

owner.dog = dog

# or for has many

owner.dogs << dog
  
```

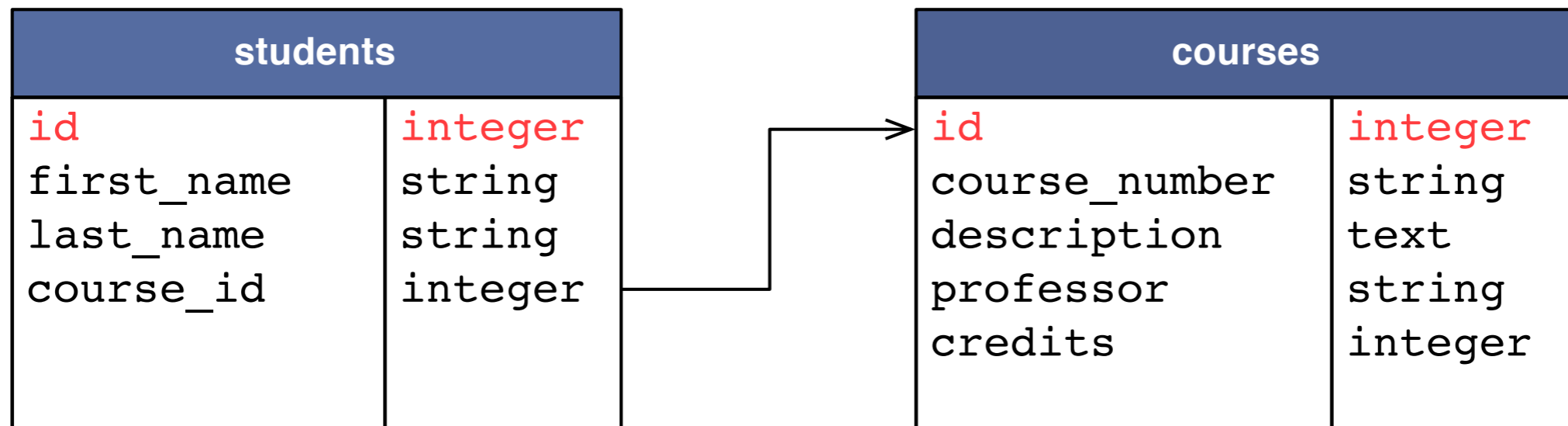
- `person.dogs` is not an array
- Associations are special magical objects
- Not magical... but...
- `owner.dogs.find_by_name("Santa's Little Helper")`
- Will automatically generate the correct SQL to get all dogs for this person with the name "Santa's Little Helper".

has_many Methods

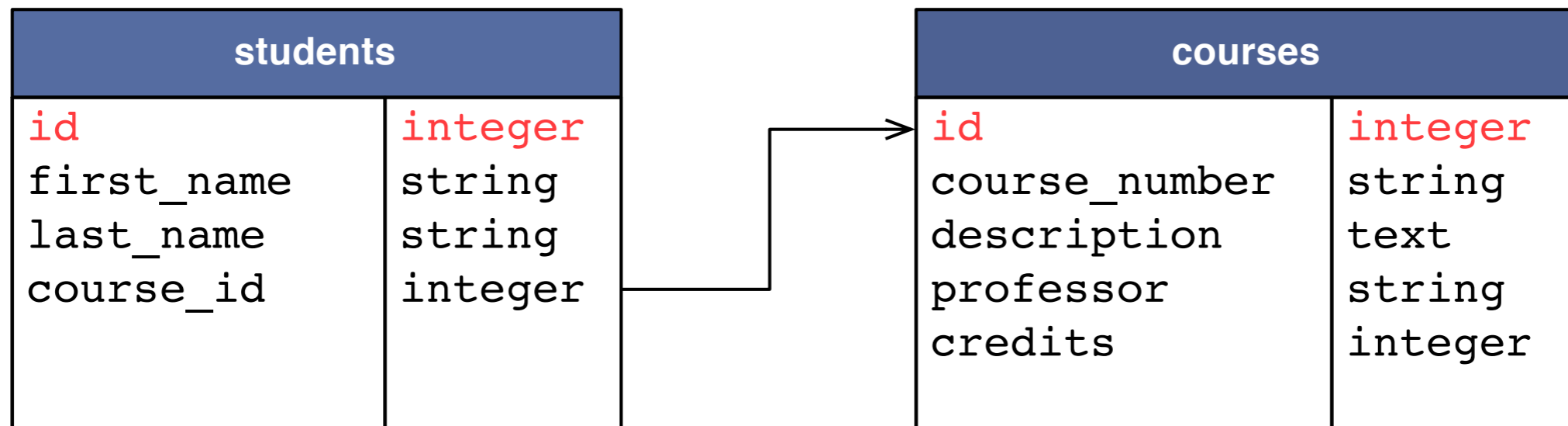
- **Many other methods as well:**
 - **count**
 - **include?**
 - **find**
 - **delete**
 - **clear**
 - **create**
 - **build**
 - **empty?**

- **Add students to the model**

- Add students to the model



- Add students to the model



What's wrong with this model?

Answer

- **A student can take more than one course**

- A student can take more than one course

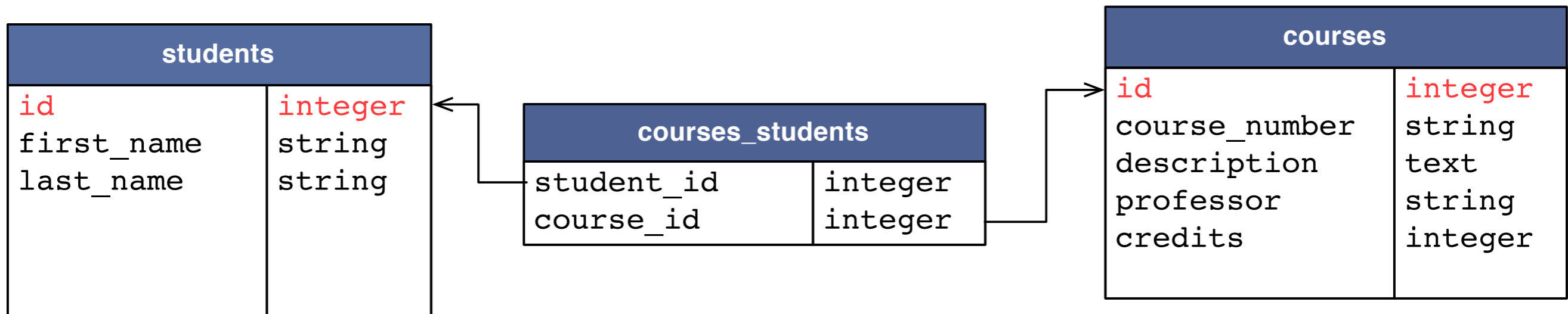
courses	
<code>id</code>	<code>integer</code>
<code>course_number</code>	<code>string</code>
<code>description</code>	<code>text</code>
<code>professor</code>	<code>string</code>
<code>credits</code>	<code>integer</code>

- A student can take more than one course

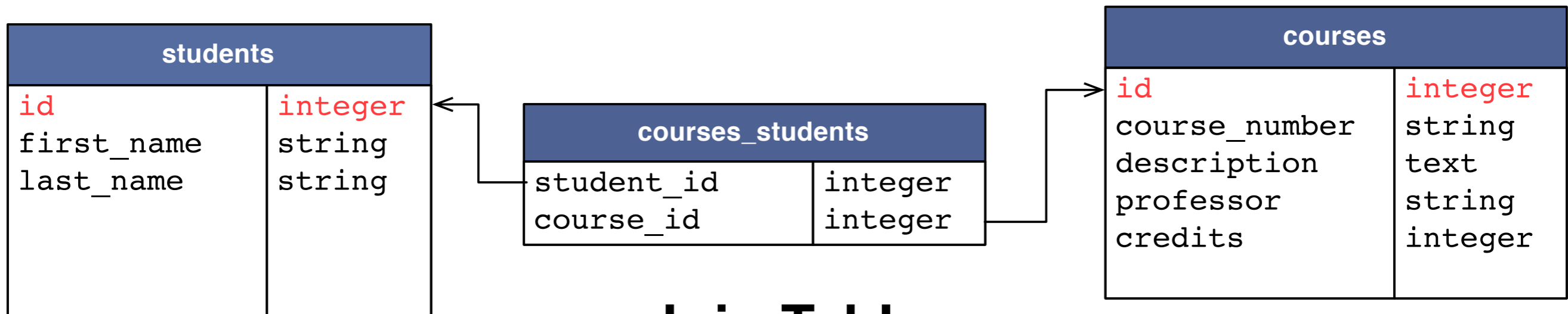
students	
id	integer
first_name	string
last_name	string
course_id	integer

courses	
id	integer
course_number	string
description	text
professor	string
credits	integer

- A student can take more than one course



- A student can take more than one course



Join Table

- Two methods of creating many to many relationships
 - Has and belongs to many
 - Join table
 - Has many through
 - Join model

Application

- **Form a group**
- **Ideal group size is 3-4 people**
- **Discuss and agree upon your application**

- Add a student model to your previous application
- This student is only able to take one course
- Hook up the two models
- Extra: Model this correctly and have a student take multiple courses and have a course have multiple students