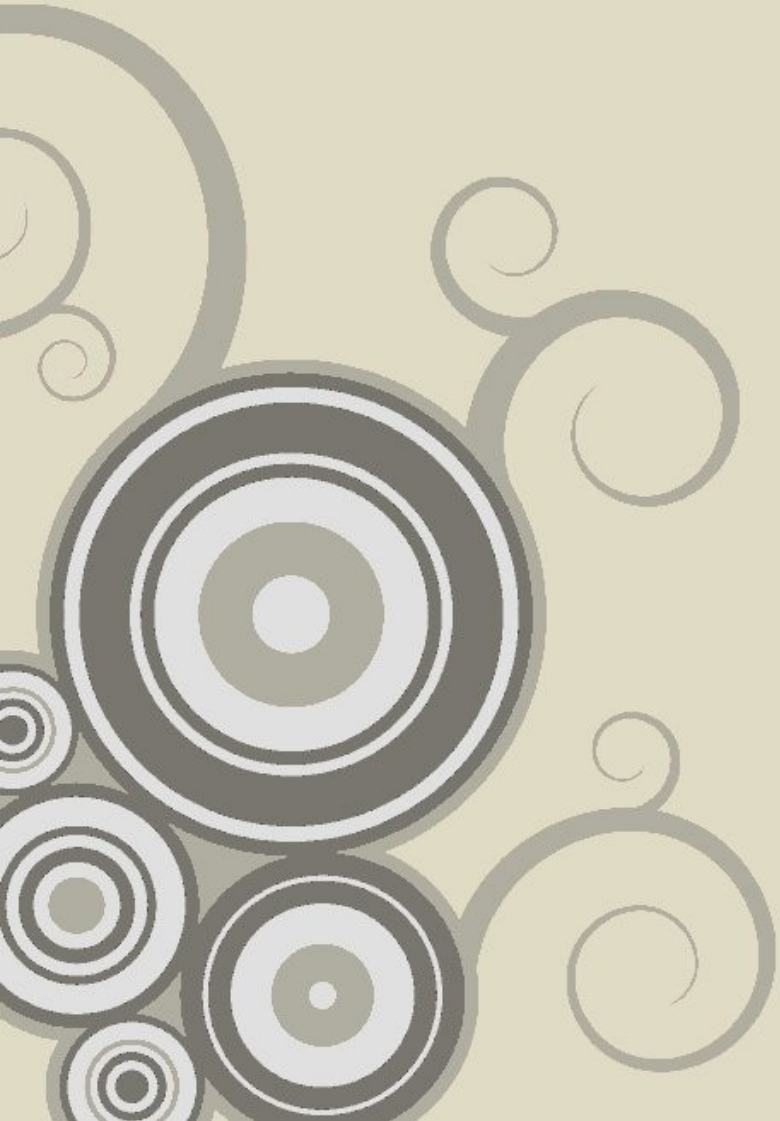
Laboratorio de Computación IV



Clase 23

Andrés Fortier

Repaso

- Relaciones anidadas 1:N
  - ActiveRecord
    - `has_many XYZ, :dependent => :destroy`
    - `accepts_nested_attributes_for XYZ, :allow_destroy => true`
  - Controller
    - params
    - .require(...).permit(..., XYZ_attributes: [..., _destroy])

Repaso

- Relaciones anidadas 1:N
  - Formulario (Mostrar)
    - `f.fields_for XYZ do |form_builder|`
  - Formulario (Eliminar)
    - `<%= f.hidden_field :_destroy %>`

      `<a href="#" onclick="removeXYZ(this)">Remove</a>`
    - Usamos la función `hide()` de jQuery.

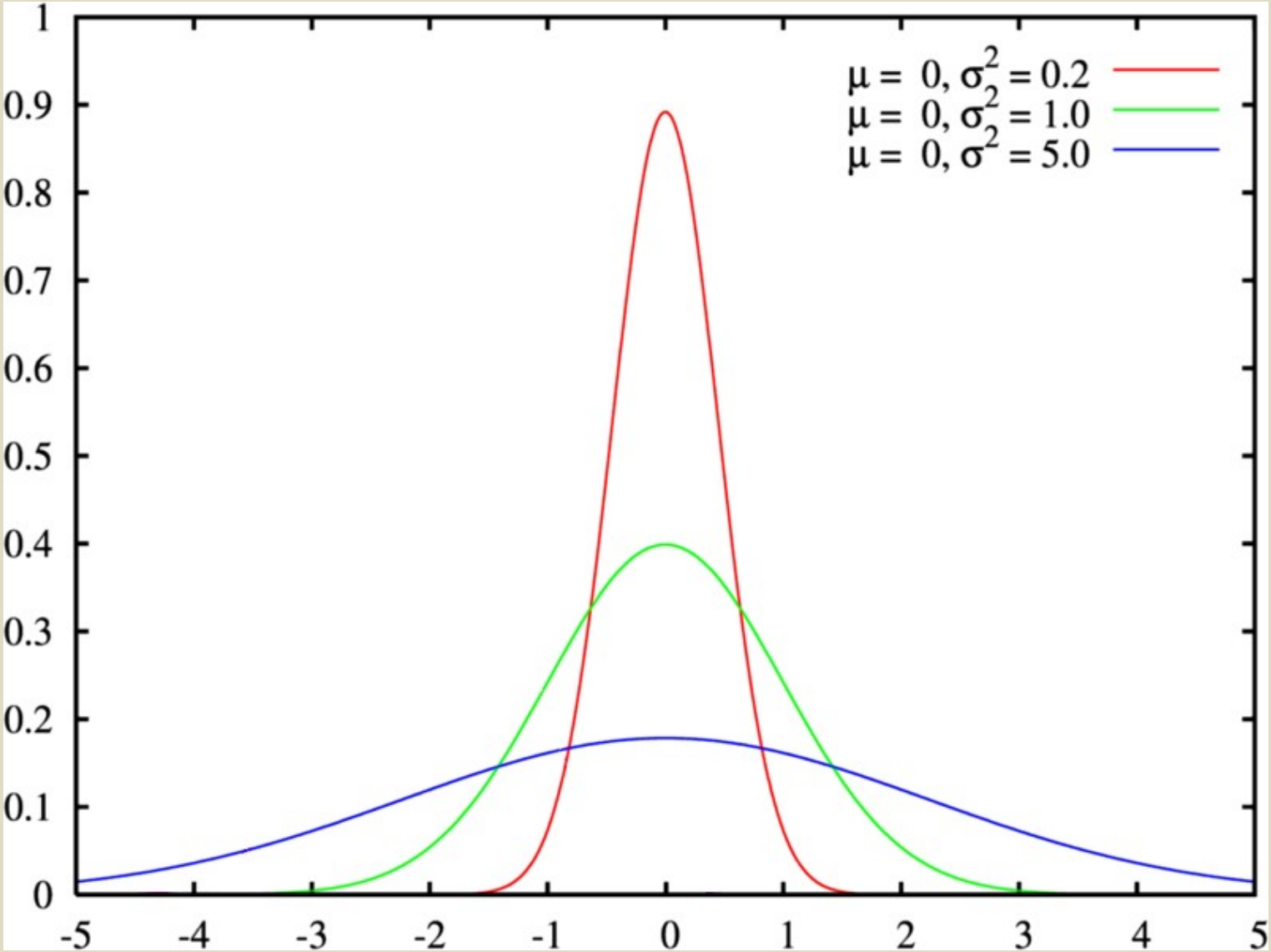
Repaso

- Relaciones anidadas 1:N
  - Formulario (Agregar)
    - (Server) Renderizamos el template para un objeto nuevo.
    - (Server) Lo convertimos en string y lo pasamos como parámetro de una función Javascript.
    - (Client) Reemplazamos, en el string del formulario, un token especial por un id nuevo.
    - (Client) Usamos la función `append` de jQuery.

Benchmarking / profiling

- Como siempre, hay un mundo detrás. Esto es simplemente una introducción al tema.
- Algunos puntos importantes
  - N evaluaciones.
  - En general, descartamos la primera.
  - El promedio no (siempre) sirve.
    - Distribución normal; varianza, mediana y outliers.
    - Distribución multimodal.

Distribución normal



Distribución bi/multi-modal

- No es el caso común en benchmarking de código, pero a veces sucede.

Benchmarking

- Hagamos una clase sencilla de benchmarking.
  - `services` no es el mejor lugar

/app/services/simple_benchmark.rb

```
class SimpleBenchmark

  def initialize(repetitions)
    @repetitions = repetitions
  end

  ...
```

Benchmarking

/app/services/simple_benchmark.rb

```
  def measure
    yield # Discard first run
    values = (1..@repetitions).map do
      Benchmark.realtime do
        yield
      end
    end
    avg = values.sum.to_f / values.size
    result = values
      .map do |value|
        "#{(value * 1000).round(2)} ms"
      end
      .join(", ")
    result + " - Average: #{(avg * 1000).round(2)} ms"
  end
end
```

Benchmarking

- Vayan a una consola de Rails

```
$ bin/rails console
> sb = SimpleBenchmark.new(5)
> sb.measure do "a"*1_000_000 end
 => "0.9 ms, 0.85 ms, 53.16 ms, 0.48 ms, 0.34 ms - Average:
11.15 ms"
> sb.measure do "a"*1_000_000 end
 => "0.96 ms, 0.86 ms, 0.83 ms, 0.88 ms, 0.92 ms - Average:
0.89 ms"
> sb.measure do "a"*1_000_000 end
 => "0.32 ms, 0.32 ms, 0.32 ms, 0.41 ms, 0.41 ms - Average:
0.36 ms"
```

Benchmarking

```
> sb = SimpleBenchmark.new(50)
> sb.measure do "a"*1_000_000 end
 => "0.84 ms, 0.83 ms, 45.23 ms, 0.32 ms, 0.32 ms, 0.34 ms,
0.33 ms, 0.34 ms, 0.35 ms, 0.35 ms, 0.35 ms, 37.69 ms, 0.32
ms, 0.49 ms, 0.33 ms, 0.33 ms, 0.34 ms, 0.35 ms, 0.36 ms,
0.34 ms, 38.16 ms, 0.31 ms, 0.31 ms, 0.32 ms, 0.32 ms, 0.4
ms, 0.35 ms, 0.35 ms, 0.35 ms, 41.23 ms, 0.36 ms, 0.35 ms,
0.36 ms, 0.37 ms, 0.38 ms, 0.44 ms, 0.39 ms, 0.42 ms, 38.59
ms, 0.32 ms, 0.32 ms, 0.33 ms, 0.33 ms, 0.42 ms, 0.37 ms,
0.36 ms, 0.37 ms, 37.81 ms, 0.31 ms, 0.31 ms - Average: 5.1
ms"
```

Benchmarking

/app/services/simple_benchmark.rb

```
  def measure
    yield # Discard first run
    values = (1..@repetitions).map do
      GC.start
      Benchmark.realtime do
        yield
      end
    end
    avg = values.sum.to_f / values.size
    result = values
      .map do |value|
        "#{(value * 1000).round(2)} ms"
      end
      .join(", ")
    result + " - Average: #{(avg * 1000).round(2)} ms"
  end
end
```

Benchmarking

```
$ bin/rails c
Loading development environment (Rails 4.1.8)
> sb = SimpleBenchmark.new(50)
> sb.measure do "a"*1_000_000 end
 => "0.46 ms, 0.33 ms, 0.29 ms, 0.13 ms, 0.17 ms, 0.14 ms,
0.13 ms, 0.13 ms, 0.18 ms, 0.13 ms, 0.13 ms, 0.17 ms, 0.13
ms, 0.13 ms, 0.13 ms, 0.18 ms, 0.14 ms, 0.13 ms, 0.17 ms,
0.13 ms, 0.13 ms, 0.17 ms, 0.13 ms, 0.13 ms, 0.15 ms, 0.16
ms, 0.15 ms, 0.15 ms, 0.16 ms, 0.15 ms, 0.17 ms, 0.17 ms,
0.15 ms, 0.15 ms, 0.15 ms, 0.16 ms, 0.14 ms, 0.15 ms, 0.14
ms, 0.14 ms, 0.14 ms, 0.14 ms, 0.14 ms, 0.15 ms, 0.17 ms,
0.14 ms, 0.13 ms, 0.13 ms, 0.14 ms, 0.13 ms - Average: 0.16
ms"
```

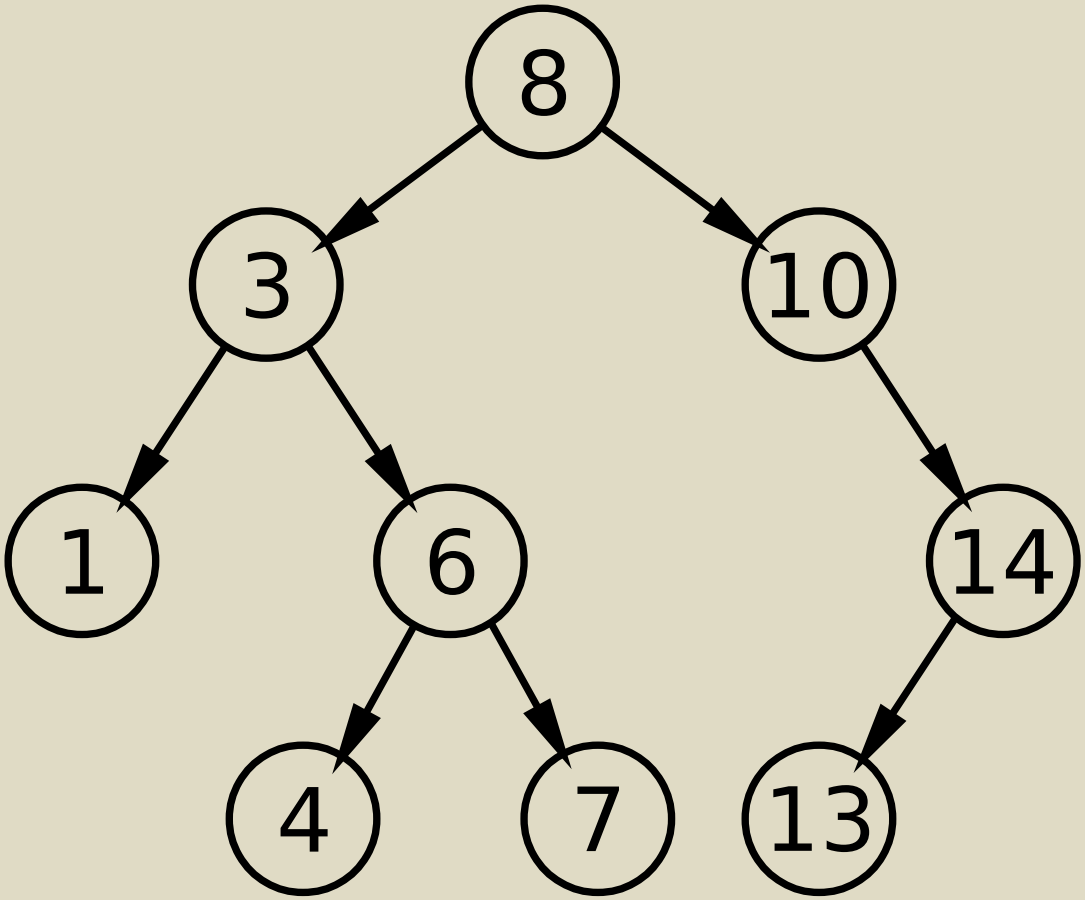
- Dejemos esto picando por un momento.

Indices en BD

- Se usan para
  - Mejorar la velocidad de búsqueda.
  - Definir restricciones (ej. unicidad).
- Tiempos de búsqueda
  - Lineal
  - Logarítmico
  - Constante

Arboles binarios de búsqueda

- Ejemplo de búsqueda logarítmica



Indices en BD

- Algunos índices usan árboles.
- Hay diversos tipos e implementaciones (ej. b-tree, r-tree, etc).
- Son árboles que se balancean al insertar/eliminar registros.
- ¿Cuándo vale la pena tener un índice en la BD?
  - En general, cuando la cantidad de búsquedas es mucho mayor a la cantidad de inserciones/borrados.

Indices en BD

- Definamos una clase mas para tener un set de prueba

/app/services/benchmark_seeds.rb

```
class BenchmarkSeeds
  def seed_posts
    Article.destroy_all
    user = User.last
    10.times do
      ActiveRecord::Base.transaction do
        5000.times do |i|
          Article.create!(title: "Seed post #{i}", text:
"This is the post number", author: user);
        end
      end
    end
  end
end
```

Indices en BD

```
$ bin/rails console
> BenchmarkSeeds.new.seed_posts
> Article.count
   (1.4ms)  SELECT COUNT(*) FROM "articles"
 => 50000
> sb = SimpleBenchmark.new(5)
```

Indices en BD

```
> sb.measure do p Article.where({title: "Seed post
999"}).to_a.length end

 Article Load (22.6ms)  SELECT "articles".* FROM "articles"
WHERE "articles"."title" = 'Seed post 999'
10
  Article Load (29.4ms)  SELECT "articles".* ...
10
  Article Load (16.5ms)  SELECT "articles".* ...
10
  Article Load (17.4ms)  SELECT "articles".* ...
10
  Article Load (15.9ms)  SELECT "articles".* ...
10
  Article Load (17.4ms)  SELECT "articles".* ...
10
 => "31.14 ms, 17.62 ms, 18.29 ms, 16.7 ms, 18.29 ms -
Average: 20.41 ms"
```

Indices en BD

```
> Article.where({title: "Seed post 999"}).explain
  Article Load (23.2ms)  SELECT "articles".* FROM "articles"
WHERE "articles"."title" = 'Seed post 999'
 => EXPLAIN for: SELECT "articles".* FROM "articles"  WHERE
"articles"."title" = 'Seed post 999'
0|0|0|SCAN TABLE articles
```

Indices en BD

```
> sb.measure do p Article.where('title LIKE ?', "%Seed post
999%").to_a.length end

  Article Load (35.1ms)  SELECT "articles".* FROM "articles"
WHERE (title LIKE '%Seed post 999%')
10
  Article Load (29.2ms)  SELECT "articles".* ...
10
  Article Load (25.5ms)  SELECT "articles".* ...
10
  Article Load (25.4ms)  SELECT "articles".* ...
10
  Article Load (26.0ms)  SELECT "articles".* ...
10
  Article Load (25.7ms)  SELECT "articles".* ...
10
 => "30.11 ms, 26.31 ms, 26.16 ms, 26.85 ms, 26.53 ms -
Average: 27.19 ms"
```

Indices en BD

```
> Article.where('title LIKE ?', "%Seed post 999%").explain
  Article Load (38.2ms)  SELECT "articles".* FROM "articles"
WHERE (title LIKE '%Seed post 999%')
 => EXPLAIN for: SELECT "articles".* FROM "articles"  WHERE
(title LIKE '%Seed post 999%')
0|0|0|SCAN TABLE articles
```

Indices en BD

- Agreguemos un índice

```
> Article.connection.add_index "articles", ["title"], name:
"index_articles_on_title"
   (0.2ms)  select sqlite_version(*)
   (0.3ms)                SELECT sql
            FROM sqlite_master
            WHERE name='index_articles_on_author_id' AND
type='index'
            UNION ALL
            SELECT sql
            FROM sqlite_temp_master
            WHERE name='index_articles_on_author_id' AND
type='index'

   (365.2ms)  CREATE  INDEX "index_articles_on_title" ON
"articles" ("title")
 => []
```

Indices en BD

```
> sb.measure do p Article.where({title: "Seed post
999"}).to_a.length end
  Article Load (0.6ms)  SELECT "articles".* FROM "articles"
WHERE "articles"."title" = 'Seed post 999'
10
  Article Load (0.4ms)  SELECT "articles".* ...
10
  Article Load (0.3ms)  SELECT "articles".* ...
10
  Article Load (0.3ms)  SELECT "articles".* ...
10
  Article Load (0.3ms)  SELECT "articles".* ...
10
  Article Load (0.3ms)  SELECT "articles".* ...
10
 => "1.17 ms, 1.26 ms, 1.58 ms, 1.06 ms, 1.05 ms - Average:
1.23 ms"
```

20.41 vs 1,23

Indices en BD

```
> Article.where({title: "Seed post 999"}).explain
  Article Load (0.7ms)  SELECT "articles".* FROM "articles"
WHERE "articles"."title" = 'Seed post 999'
 => EXPLAIN for: SELECT "articles".* FROM "articles"  WHERE
"articles"."title" = 'Seed post 999'
0|0|0|SEARCH TABLE articles USING INDEX
index_articles_on_title (title=?)
```

Indices en BD

```
> sb.measure do p Article.where('title LIKE ?', "%Seed post
999%").to_a.length end
  Article Load (35.9ms)  SELECT "articles".* FROM "articles"
WHERE (title LIKE '%Seed post 999%')
10
  Article Load (26.4ms)  SELECT "articles".* ...
10
  Article Load (25.4ms)  SELECT "articles".* ...
10
  Article Load (24.5ms)  SELECT "articles".* ...
10
  Article Load (26.1ms)  SELECT "articles".* ...
10
  Article Load (25.1ms)  SELECT "articles".* ...
10
 => "27.27 ms, 26.29 ms, 25.27 ms, 26.9 ms, 25.96 ms -
Average: 26.34 ms"
```

27.19 vs 26.34 (?!)

Indices en BD

```
> Article.where('title LIKE ?', "%Seed post 999%").explain
  Article Load (38.9ms)  SELECT "articles".* FROM "articles"
WHERE (title LIKE '%Seed post 999%')
 => EXPLAIN for: SELECT "articles".* FROM "articles"  WHERE
(title LIKE '%Seed post 999%')
0|0|0|SCAN TABLE articles
```

Acceso por medio de relaciones

- Para acceder al email del autor de un post debemos hacer `article.author.email`

```
> Article.last.author.email
  Article Load (0.2ms)  SELECT  "articles".* FROM "articles"
 ORDER BY "articles"."id" DESC LIMIT 1
  User Load (0.1ms)  SELECT  "users".* FROM "users"  WHERE
"users"."id" = ? LIMIT 1  [["id", 2]]
 => "jane@example.com"
```

- ¿Que sucede si intentamos esto sobre una lista de artículos (ej. 500)?

Acceso por medio de relaciones

```
> sb.measure do p (Article.limit(500).inject("") {|str,
article| article.author.email + str}).length end

  Article Load (6.5ms)  SELECT  "articles".* FROM "articles"
LIMIT 500
  User Load (0.1ms)  SELECT  "users".* FROM "users"  WHERE
"users"."id" = ? LIMIT 1  [["id", 2]]
  ...
  User Load (0.1ms)  SELECT  "users".* FROM "users"  WHERE
"users"."id" = ? LIMIT 1  [["id", 2]]
  User Load (0.1ms)  SELECT  "users".* FROM "users"  WHERE
"users"."id" = ? LIMIT 1  [["id", 2]]
8000
 => "318.49 ms, 328.4 ms, 352.44 ms, 369.35 ms, 323.2 ms -
Average: 338.37 ms"
```

Acceso por medio de relaciones

- Problema: tenemos un query por autor (500).
- Solución: indicarle a AR que pre-cargue los registros asociados.
- Dos formas de hacerlo
  - Como queries independientes.
  - Como left-joins.

Acceso por medio de relaciones

```
> sb.measure do p
(Article.includes(:author).limit(500).inject("") {|str,
article| article.author.email + str}).length end

  Article Load (6.3ms)  SELECT  "articles".* FROM "articles"
LIMIT 500
  User Load (0.2ms)  SELECT "users".* FROM "users"  WHERE
"users"."id" IN (2)
8000
  ...
8000
  Article Load (2.2ms)  SELECT  "articles".* FROM "articles"
LIMIT 500
  User Load (0.2ms)  SELECT "users".* FROM "users"  WHERE
"users"."id" IN (2)
8000
 => "27.24 ms, 25.74 ms, 23.2 ms, 24.18 ms, 23.5 ms -
Average: 24.77 ms"
```

338.87 vs 24.77

Acceso por medio de relaciones

```
> sb.measure do p
(Article.eager_load(:author).limit(500).inject("") {|str,
article| article.author.email + str}).length end
SQL (9.1ms)  SELECT  "articles"."id" AS t0_r0,
"articles"."title" AS t0_r1, "articles"."text" AS t0_r2,
"articles"."created_at" AS t0_r3, "articles"."updated_at" AS
t0_r4, "articles"."author_id" AS t0_r5, "users"."id" AS
t1_r0, "users"."email" AS t1_r1, "users"."encrypted_password"
AS t1_r2, "users"."reset_password_token" AS t1_r3,
"users"."reset_password_sent_at" AS t1_r4,
"users"."remember_created_at" AS t1_r5,
"users"."sign_in_count" AS t1_r6,
"users"."current_sign_in_at" AS t1_r7,
"users"."last_sign_in_at" AS t1_r8,
"users"."current_sign_in_ip" AS t1_r9,
"users"."last_sign_in_ip" AS t1_r10, "users"."created_at" AS
t1_r11, "users"."updated_at" AS t1_r12 FROM "articles" LEFT
OUTER JOIN "users" ON "users"."id" = "articles"."author_id"
LIMIT 500
```

Acceso por medio de relaciones

```
...
8000
 => "43.99 ms, 41.58 ms, 41.52 ms, 40.96 ms, 42.33 ms -
Average: 42.07 ms"
```

338.87 vs 42.07

Cerrando

- Importancia en saber qué medir y cómo medirlo.
- Árboles de búsqueda para mejorar tiempos.
- Indices de BD para mejorar tiempos de búsquedas.
- Pre-cargar modelos relacionados para reducir la cantidad de queries.

Yapa

- Maintenance mode: cuando están trabajando sobre su app.
- https://github.com/biola/turnout

Links

- http://blogs.perl.org/users/steffen_mueller/2010/09/your-benchmarks-suck.html
- https://www.ruby-toolbox.com/categories/Benchmarking
- http://blog.arkency.com/2013/12/rails4-preloading/
- http://www.slideshare.net/aysylu/benchmarking-youre-doing-it-wrong-strangeloop-2014
- http://stackoverflow.com/questions/1108/how-does-database-indexing-work