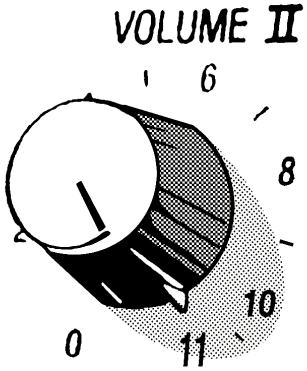
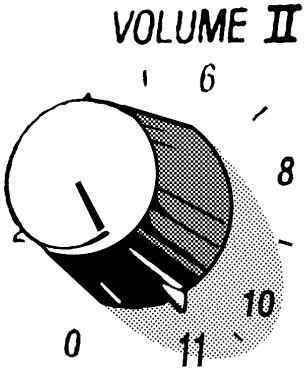
Turning

Rails STI

up to eleven



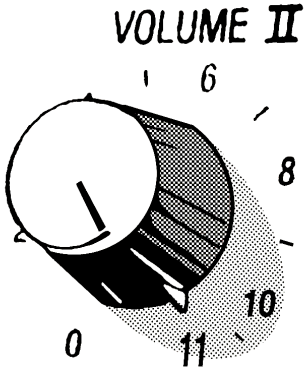
Damian Kampik
u2i



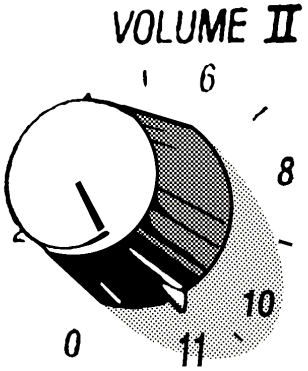
Turning

Rails STI

up to eleven



(polymorphic associations too)



QR codes

Spinal Tap

Spinal Tap


You're on ten on your guitar. Where can you go from there?
You put it at Eleven. Exactly. One louder.

Posted by u/[deleted] 4 years ago

15

"Alexa, volume 11" sets the volume to max. Is this a tribute to 'up to 11'

Alexa will tell you that you can only set the volume from 0 to 10, but when you say volume 11 it puts it to the max.

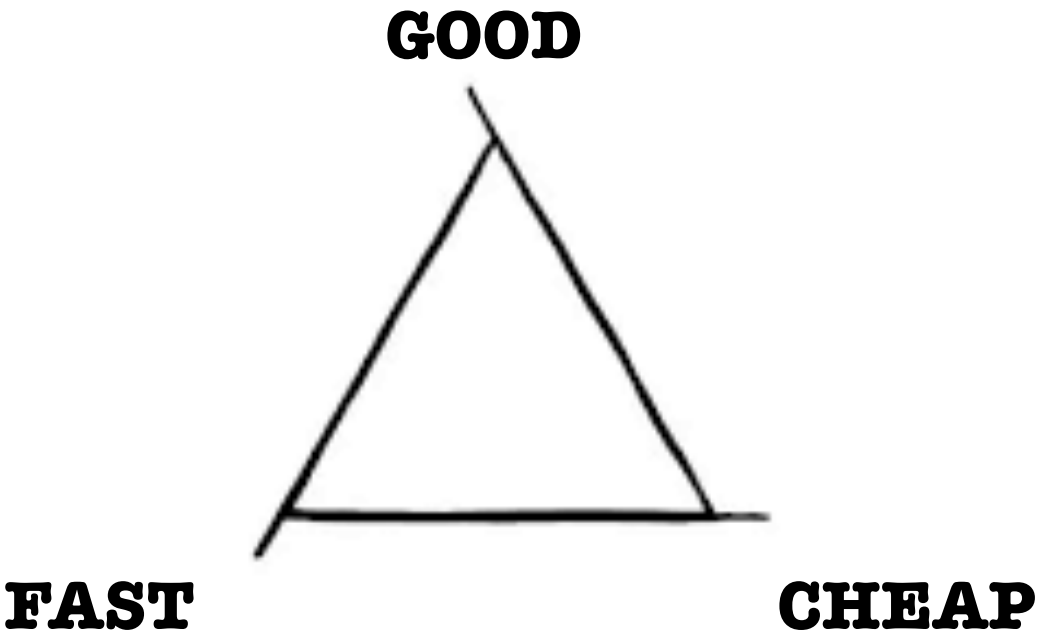
It’s convenient with your framework.
Where can you go from there?

It’s convenient with your framework.
Where can you go from there?
You make it quicker.
Exactly.
One better.

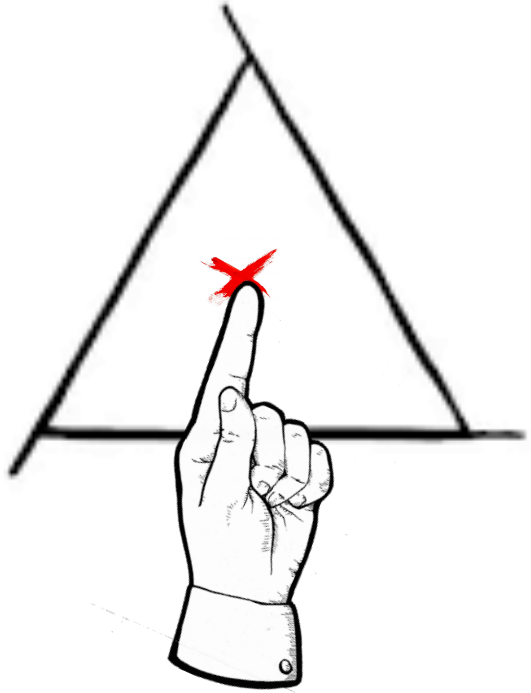
STI

STI

Polymorphic associations


GOOD
FAST
CHEAP

GOOD



FAST

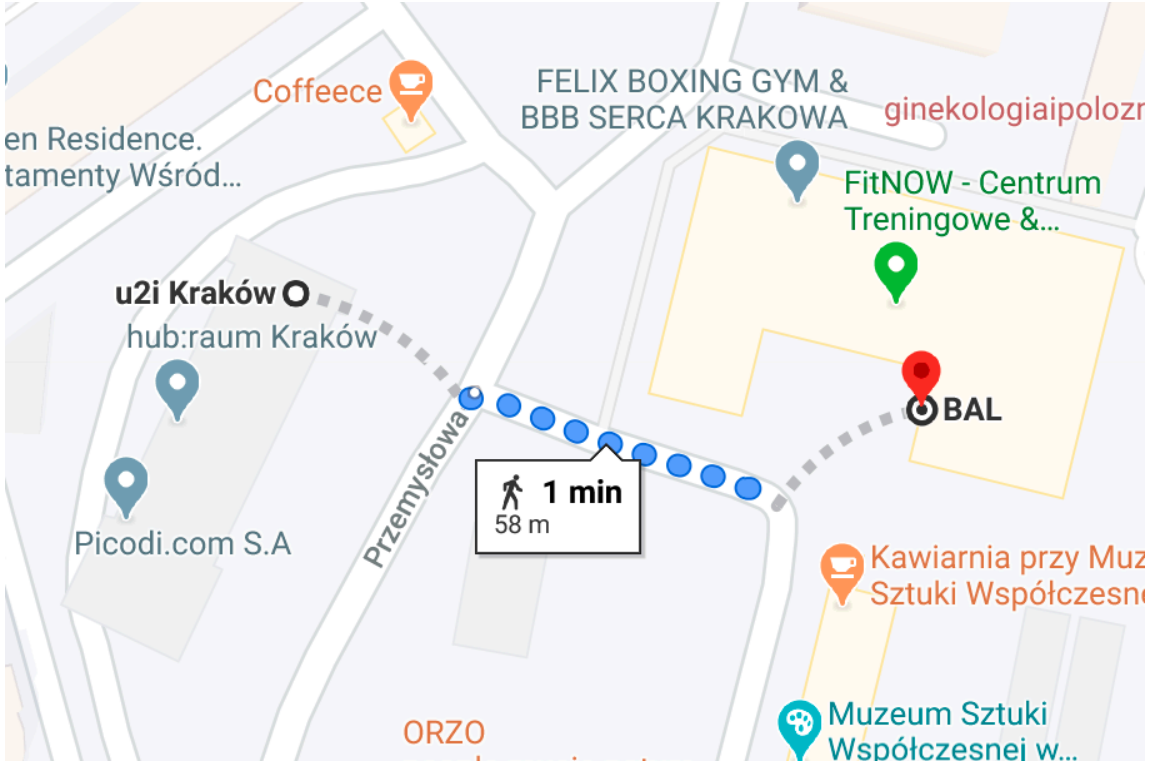
CHEAP

A short intro

- **Damian Kampik**
- 6 yrs of professional experience
- 2 years in u2i
- Come visit us and let's have a coffee together :)

u2i

Come visit us - let’s have a coffee together!



Agenda

Agenda

1. STI & polymorphic associations - what are these?
2. What pushed us to search for a solution?
3. A bit about the idea
4. Solution

Definitions

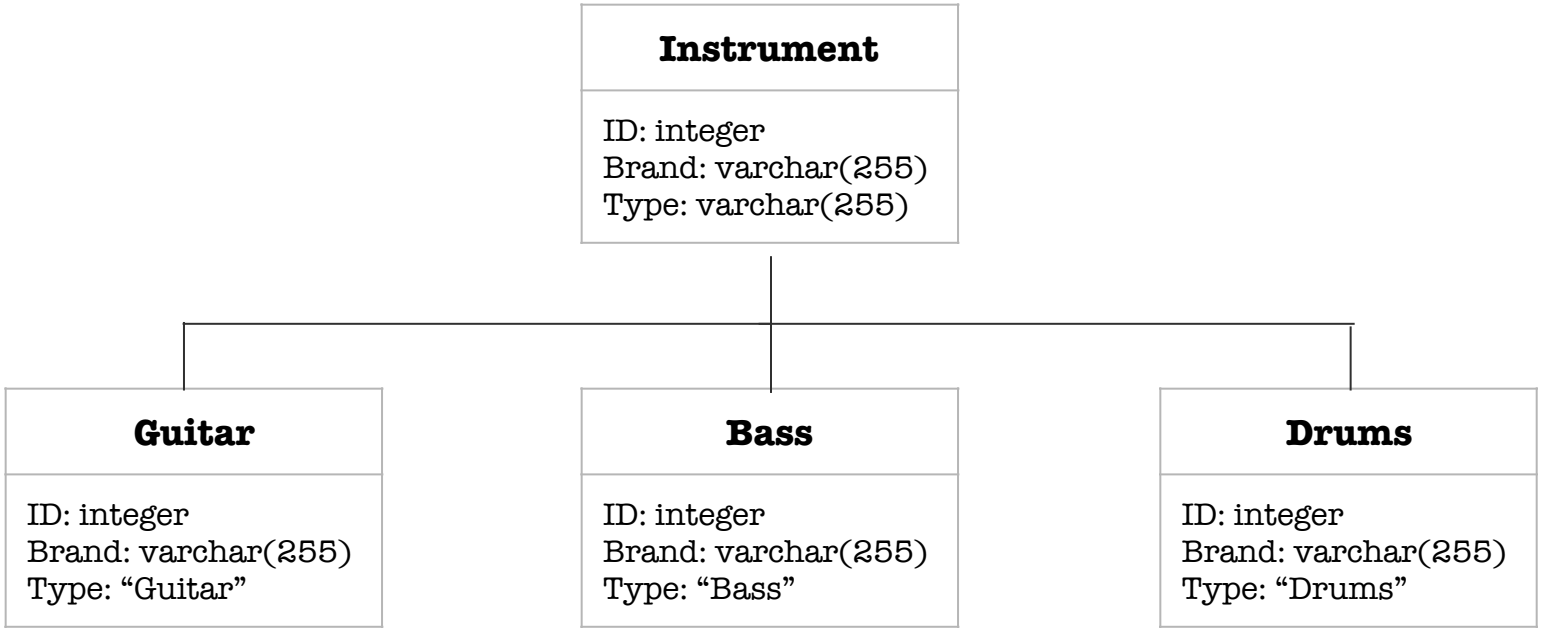
Single Table Inheritance

Guitar
ID: integer Brand: varchar(255)

Bass
ID: integer Brand: varchar(255)

Drums
ID: integer Brand: varchar(255)

Single Table Inheritance



How does it work?

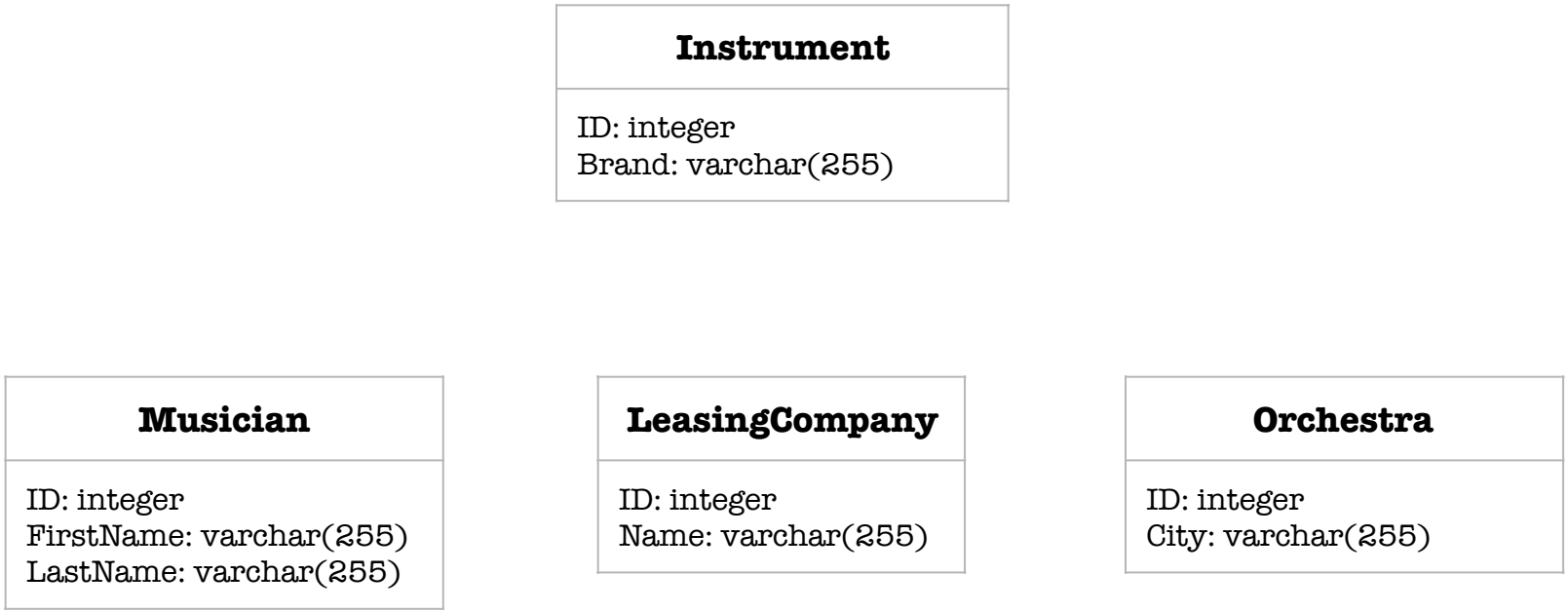
```
class Instrument < ApplicationRecord; end

class Guitar < Instrument; end
class Bass < Instrument; end
class Drums < Instrument; end
```

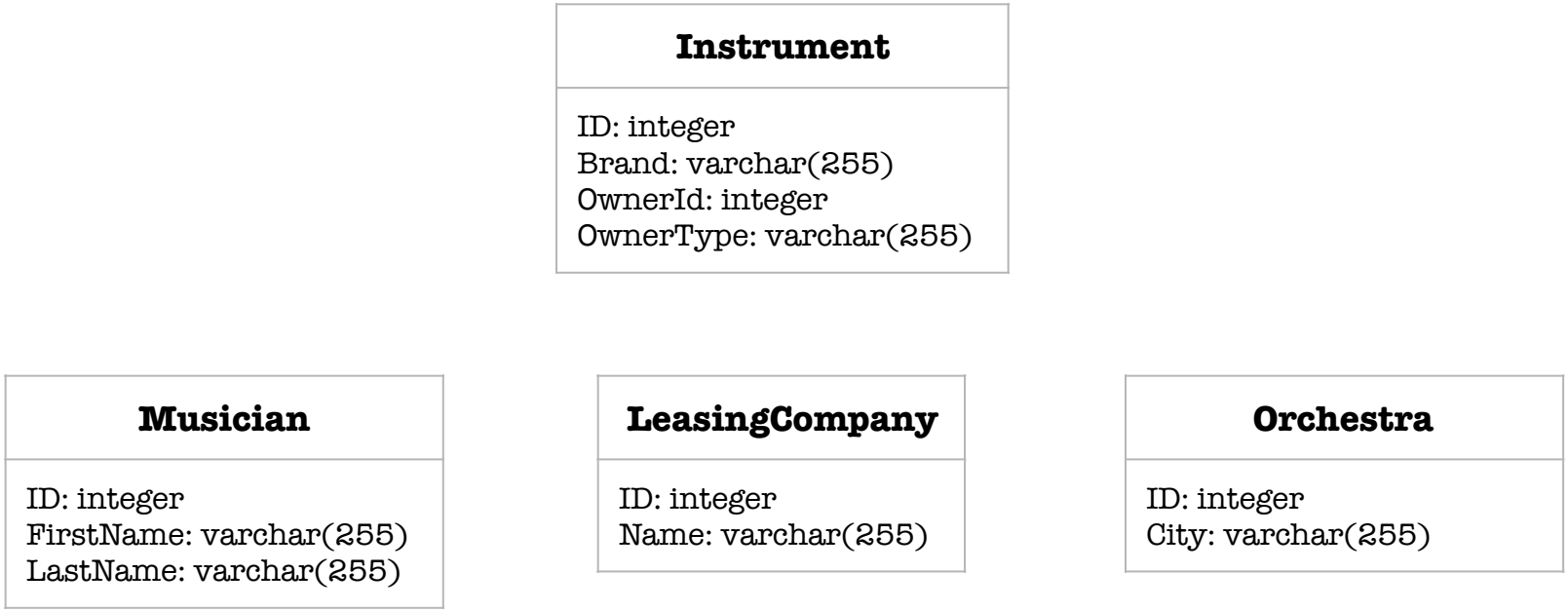
How does it work?

```
2.4.1 :001 > Guitar.first
  Guitar Load (0.6ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."type" IN ('Guitar') ORDER
BY "instruments"."id" ASC LIMIT ?  [["LIMIT", 1]]
 => #<Guitar id: 4, type: "Guitar", created_at: "2019-11-13
23:55:25", updated_at: "2019-11-13 23:55:25">
```

Polymorphic associations



Polymorphic associations



How does it work?

```ruby
class Instrument < ApplicationRecord
  belongs_to :owner, polymorphic: true
end

class LeasingCompany < ApplicationRecord
  has_many :instruments, as: :owner
end
```

How does it work?

```
2.4.1 :001 > Instrument.where(owner: Musician.first)
  Musician Load (0.5ms)  SELECT  "musicians".* FROM "musicians"
ORDER BY "musicians"."id" ASC LIMIT ?  [["LIMIT", 1]]
  Instrument Load (0.7ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."owner_type" = ? AND
"instruments"."owner_id" = ? LIMIT ?  [["owner_type",
"Musician"], ["owner_id", 1], ["LIMIT", 11]]
 => #<ActiveRecord::Relation [#<Instrument id: 1, owner_id: 1,
owner_type: "Musician", created_at: "2019-11-13 23:47:49",
updated_at: "2019-11-13 23:47:49">]>
```

Problem

A bit of math

~100 000 users

A bit of math

~100 000 users

~200 business days

———————————————————

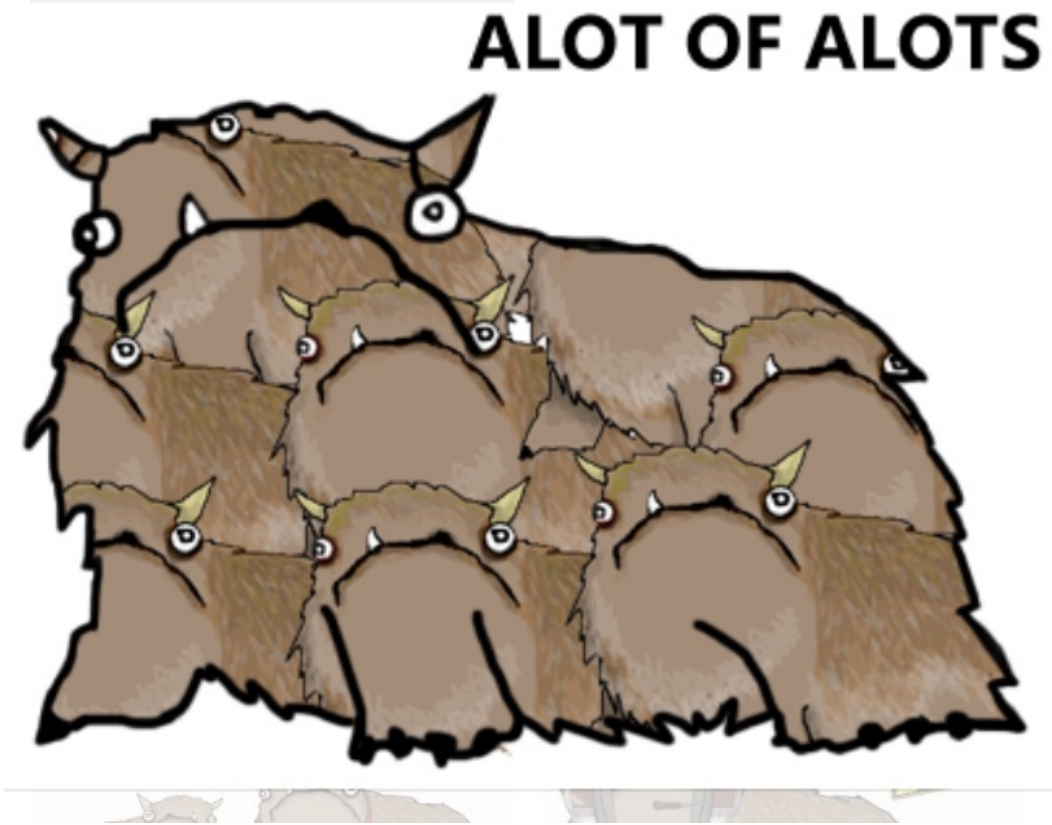
A bit of math

~100 000 users

~200 business days

---

20 000 000 records / year


ALOT OF ALOTS

Idea

Let's take our table...

Column	Type
id	integer
owner_id	integer
owner_type	varchar(255)
type	varchar(255)

Let's take our table...

Column	Type
id	integer
owner_id	integer
owner_type	varchar(255)
type	varchar(255)

Musician
LeasingCompany
Orchestra

Guitar
Bass
Drums

… and map strings to integers

Column	Type
id	integer
owner_id	integer
owner_type	smallint
type	smallint

1 #Musician
2 #LeasingCompany
3 #Orchestra

1 #Guitar
2 #Bass
3 #Drums

Assumptions

1. The table will take up less disk space

Assumptions

1. The table will take up less disk space
2. Table indices will take up less disk space

Assumptions

1. The table will take up less disk space
2. Table indices will take up less disk space
3. Integer-based indices will be quicker than their string-based counterparts

Solution

Solution

pt. 1 - STI

Example data

instruments_sample(id, type); index on „type” column, 10m records

	5-char strings	**10-char strings**
Data	610 MB	623 MB
Indexes	243 MB	287 MB
Data + indexes	853 MB	910 MB
Avg. lookup time	713,76 ms	855,17 ms

Example data

instruments_sample(id, type); index on „type” column, 10m records

	Before optimization	**After optimization**
Data	623 MB	
Indexes	287 MB	
Data + indexes	910 MB	
Avg. lookup time	855,17 ms	

Instrument - before changes

```ruby
class Instrument < ApplicationRecord
end

# as a reminder:
# class Guitar < Instrument; end
# class Bass < Instrument; end
# class Drums < Instrument; end
```

Instrument - after changes

```ruby
class Instrument < ApplicationRecord
 include OptimallyInheritable

 support_sti_for %w[Guitar Bass Drums]
end
```

OptimallyInheritable

```ruby
module OptimallyInheritable
  extend ActiveSupport::Concern

  module ClassMethods
    def support_sti_for(cls_list) ... end
    def sti_cls_list ... end
    def find_sti_class(type_name) ... end
    def sti_name ... end
  end
end
```

OptimallyInheritable

```
def support_sti_for(cls_list)
  @sti_cls_list = []
  @sti_cls_list += cls_list
end
```

OptimallyInheritable

```
def sti_cls_list
  if superclass.respond_to? :sti_cls_list
    superclass.sti_cls_list
  else
    @sti_cls_list
  end
end
```

OptimallyInheritable

```
def sti_name
  idx = sti_cls_list.index(self.name)
  if idx.nil?
    super
  else
    idx + 1
  end
end
```

OptimallyInheritable

```
def find_sti_class(type_name)
  idx = type_name.to_i
  super if idx.zero?

  sti_cls_list[type_name.to_i - 1].constantize
rescue NameError, TypeError
  super
end
```

Example query - before changes

```
2.4.1 :001 > Guitar.first
  Guitar Load (0.6ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."type" IN ('Guitar') ORDER
BY "instruments"."id" ASC LIMIT ?  [["LIMIT", 1]]
 => #<Guitar id: 4, type: "Guitar", created_at: "2019-11-13
23:55:25", updated_at: "2019-11-13 23:55:25">
```

Example query - after changes

```
2.4.1 :001 > Guitar.first
  Guitar Load (0.5ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."type" IN (1) ORDER BY
"instruments"."id" ASC LIMIT ?  [["LIMIT", 1]]
 => #<Guitar id: 4, type: 1, created_at: "2019-11-13
23:55:25", updated_at: "2019-11-13 23:55:25">
```

Result

instruments_sample(id, type); index on „type” column, 10m records

	Before optimization	**After optimization**
Data	623 MB	
Indexes	287 MB	
Data + indexes	910 MB	
Avg. lookup time	855,17 ms	

Result

instruments_sample(id, type); index on „type” column, 10m records

	Before optimization	**After optimization**
Data	623 MB	560 MB
Indexes	287 MB	214 MB
Data + indexes	910 MB	774 MB
Avg. lookup time	855,17 ms	714,85 ms

Result

instruments_sample(id, type); index on „type” column, 10m records

	Before optimization	**After optimization**
Data	610 MB	560 MB
Indexes	243 MB	214 MB
Data + indexes	853 MB	774 MB
Avg. lookup time	713,76 ms	714,85 ms

Conclusions

1. For short strings we're mostly saving disk space
2. The longer the string is, the better is to migrate to integers
3. We become resilient against variance of table size caused by long values stored in „type" column

Instrument:: HyperbassFlute

1

Why not enum?!

Why not enum?!

Enum (PSQL)	**Tinyint**
4 bytes	1 byte
Heavier indexes	Lighter indexes
Requires migrations to add new values	You can simply add a value to array

Solution

pt. 2 - polymorphic associations

Example data

instruments_sample(id, resource_type, resource_id);
index on „resource_type” and „resource_id”,
10m records

	Before optimization	**After optimization**
Data	687 MB	
Indexes	344 MB	
Data + indexes	1031 MB	
Avg. lookup time	1418,48 ms	

doliveirakn / **polymorphic_integer_type**

Gemfile

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/
#{repo}.git" }

ruby '2.4.1'

gem 'rails', '~> 5.2.3'

...
gem 'polymorphic_integer_type', ‘2.2.1’
```

Instrument - before changes

```ruby
class Instrument < ApplicationRecord
  belongs_to :owner, polymorphic: true
end

# as a reminder:
# owners: [Musician, LeasingCompany, Orchestra]
```

Instrument - after changes

```ruby
class Instrument < ApplicationRecord
  include PolymorphicIntegerType::Extensions
  INSTRUMENT_POLYMORPHISM = {
    1 => 'Musician',
    2 => 'LeasingCompany',
    3 => 'Orchestra'
  }

  belongs_to :owner, polymorphic: INSTRUMENT_POLYMORPHISM
end
```

LeasingCompany - before changes

```ruby
class LeasingCompany < ApplicationRecord
  has_many :instruments, as: :owner
end
```

LeasingCompany - after changes

```ruby
class LeasingCompany < ApplicationRecord
  include PolymorphicIntegerType::Extensions
  has_many :instruments, as: :owner
end
```

Instrument - before changes

```
2.4.1 :001 > Instrument.where(owner: Musician.first)
  Musician Load (0.5ms)  SELECT  "musicians".* FROM "musicians"
ORDER BY "musicians"."id" ASC LIMIT ?  [["LIMIT", 1]]
  Instrument Load (0.7ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."owner_type" = ? AND
"instruments"."owner_id" = ? LIMIT ?  [["owner_type",
"Musician"], ["owner_id", 1], ["LIMIT", 11]]
 => #<ActiveRecord::Relation [#<Instrument id: 1, owner_id: 1,
owner_type: "Musician", created_at: "2019-11-13 23:47:49",
updated_at: "2019-11-13 23:47:49">]>
```

Instrument - after changes

```
2.4.1 :001 > Instrument.where(owner: Musician.first)
  Musician Load (0.5ms)  SELECT  "musicians".* FROM "musicians"
ORDER BY "musicians"."id" ASC LIMIT ?  [["LIMIT", 1]]
  Instrument Load (0.6ms)  SELECT  "instruments".* FROM
"instruments" WHERE "instruments"."owner_type" = ? AND
"instruments"."owner_id" = ? LIMIT ?  [["owner_type", 1],
["owner_id", 1], ["LIMIT", 11]]
 => #<ActiveRecord::Relation [#<Instrument id: 1, owner_id: 1,
owner_type: 1, created_at: "2019-11-13 23:47:49", updated_at:
"2019-11-13 23:47:49">]>
```

What does this gem do?

```
Model.#{resource_type}_mapping
Instrument.owner_type_mapping
```

What does this gem do?

```
Model.#{resource_type}_mapping
Instrument.owner_type_mapping

ActiveRecord::PredicateBuilder::
PolymorphicArrayValue.type_to_ids_mapping
```

What does this gem do?

```
Model.#{resource_type}_mapping
Instrument.owner_type_mapping

ActiveRecord::PredicateBuilder::PolymorphicArrayValue.
type_to_ids_mapping

INSTRUMENT_POLYMORPHISM = {
  1 => 'Musician',
  2 => 'LeasingCompany',
  3 => 'Orchestra'
}
```

Example data

instruments_sample(id, resource_type, resource_id);
index on „resource_type” and „resource_id”,
10m records

	Before optimization	**After optimization**
Data	687 MB	
Indexes	344 MB	
Data + indexes	1031 MB	
Avg. lookup time	1418,48 ms	

Example data

instruments_sample(id, resource_type, resource_id);
index on „resource_type” and „resource_id”,
10m records

	Before optimization	**After optimization**
Data	687 MB	623 MB
Indexes	344 MB	229 MB
Data + indexes	1031 MB	852 MB
Avg. lookup time	1418,48 ms	670,49 ms

Conclusions

1. We're saving space mostly on indexes
2. Lookup time can be heavily reduced due to index size and performance

Proof that it works!

Production results

After doing these, the table size went down from:
24GB to 7.5GB

Indexes:
1) Primary key (id) index is the same (as expected)
2) `idx_attendance_tiers_on_date_resource_student_year_type` goes down from 4888 MB to 3362 MB
3) `idx_attendance_tiers_resource_and_student` goes down from 4117 MB to 2611 MB.

Any questions?

Thank you!