ActiveRecord::Relation and Arel

Rob Mack, Spiceworks

Theo Mills and I built data access layer for a large Rails app. We built it on top of Active Record and Arel. ActiveRecord a bit of a leaky abstraction. You think in associations some places, but table names other places.

What's Arel?

- Query generation for Rails 3
- Relational Algebra (algebra of sets)
- gem github://rails/arel

- closure - not that closure, just means returns a set that you can operate on
- creates syntax tree
  - leaves are relations
  - internal nodes are operators

Arel History

- 2007 - Nick Kallen (Pivotal/Twitter) creates sql_algebra
- 2008 - Renamed to Active Relation, then Arel
- 2009 - Bryan Helmkamp (@brynary), takes over as maintainer
- 2009 - Emilo Tagua - integrates it with ActiveRecord
- 2010 - Aaron Patterson (@tenderlove) - starts

Source: https://speakerdeck.com/u/tomstuart/p/relational-algebra-and-arel

What does this have to do with Rails? Rails 2 queries were generated by string concatenation. Sql string was built up, then executed. Rails 3 changes this with Arel, but when 3.0 launched, Arel was on of the causes for the Rails 3 slowness. The original Arel stores methods calls in a link list, to generation the sql, it has to walk backwards through the list. Tenderlove changes design to abstract syntax tree with a manager object. Once you have your AST, you don't necessarily need to generate SQL, hence visitors.

Arel vs. ActiveRecord::Relation

There's a lot of confusion about Arel vs. ActiveRecord::Relation. AR::Rel is the AR layer on top of Arel.

ActiveRecord::Relation

- where
- select
- group
- order
- reorder
- reverse_order
- limit
- offset
- joins
- includes
- lock
- readonly
- having
- uniq

Active Record query interface defined in ActiveRecord::QueryMethods. All return relations.

What's in it for me?

- lazy evaluation
- chaining
- portability

Lazy Evaluation

```
# app/controller/user_controller.rb

def index
  @recent_users = User.where("created_at > ?", 1.day.ago)
 end

 # app/controller/index.html.erb

<% cache("recent_users") do %>
  <% @recent.users.each do |u| %>
  ...
```


more useful on a large legacy app

Chaining

```
> User.where(:id => 1).object_id
=> 43078

> User.where(:id => 1).object_id
=> 43080

User.where(:status => 'active').
    where("created_at > ?", 1.day.ago).
    order(:created_at).limit(10)
```


Chaining works by cloning the relation, building complex search example (app not big enough for full search solution)

Useful Methods

> User.order(:created_at).**to_sql**
=> "SELECT users.* FROM users ORDER BY created_at"

> User.**unscoped**.class
=> ActiveRecord::Relation

> User.unscoped.to_sql
=> "SELECT users.* FROM users"

Note: Arel has a to_sql method too, unscoped remove default scope, also a scoped method

Merge Relations

```
inactive = User.where(:status => 'inactive')
recent = User.where("created_at > ?", 1.week.ago)

recent.merge(inactive).to_sql
=> "SELECT users.* FROM users WHERE users.status
= 'active' AND (users.created_at > '2012-08-21');"
```

where_values

recent.merge(inactive).where_values
=> [<Arel::Node>, "created_at > '2012-08-21'"]

joins_values
select_values
limit_values

uniq_values
order_values
etc..

A lot like named scopes

```
scope :active, where(:status => 'active')
```

```
def self.active
  where(:status => 'active')
end
```


not much difference

Console Gotcha

```
> User.where(:status => 'active')
=> doh!

# File activerecord/lib/active_record/relation.rb
def inspect
  to_a.inspect
end
```


executing this in the rails console is generally bad, use .to_sql or ;nil
Inspect method calls to_a, so if you are playing with Relations on console, the output spew can be overwhelming

Arel

Arel Operators

- eq
- not_eq
- lt
- lteq
- gt
- gteq
- matches # like
- not_matches
- in
- not_in
- take # limit
- skip # offset

ActiveRecord::Base

```
arel_table
arel_engine

u = User.arel_table
u.class
=> Arel::Table
```

Mix and Match

```
u = User.arel_table
User.where(u[:created_at].gt(1.day.ago)).limit(1)

User.where(:id => 10).arel.class
=> Arel::SelectManager
```


drop arel into relations, use arel method to get from a relation back to the underlying arel object

Why do I need Arel?

- complex queries
- complex dynamic joins, specifying table alias
- compatibility

Maybe you don't

Visitors

Arel::Visitors::PostgreSQL
Arel::Visitors::MySQL
Arel::Visitors::MSSQL
Arel::Visitors::Oracle
Arel::Visitors::SQLite
Arel::Visitors::IBM_DB
Arel::Visitors::Informix

Visitor design pattern, separate algorithm from object structure that it operates on. Visitors traverse the AST creating SQL.

Portability with Visitors

```
u = User.arel_table
User.where(u[:name].matches("bob")).to_sql

"users.name ILIKE 'bob'" # postgres
"users.name LIKE 'bob'" #mysql
```

Table Join Gotcha

```
def self.cheaper_than(price)
  where("price < ?", price)
end
```

```
def self.cheaper_than(price)
  where(arel_table[:price].lt(price))
end
```


second one will work if you are joining in multiple tables with price column, aliased table or otherwise

Ernie Miller

- github://ernie/squeel
- railscasts.com/episodes/354-squeel
- erniemiller.org

Ernie Miller – living social, knows a ton about arel, contributor, interesting blog
before you go diving to far into Arel, take a look at the squeel gem

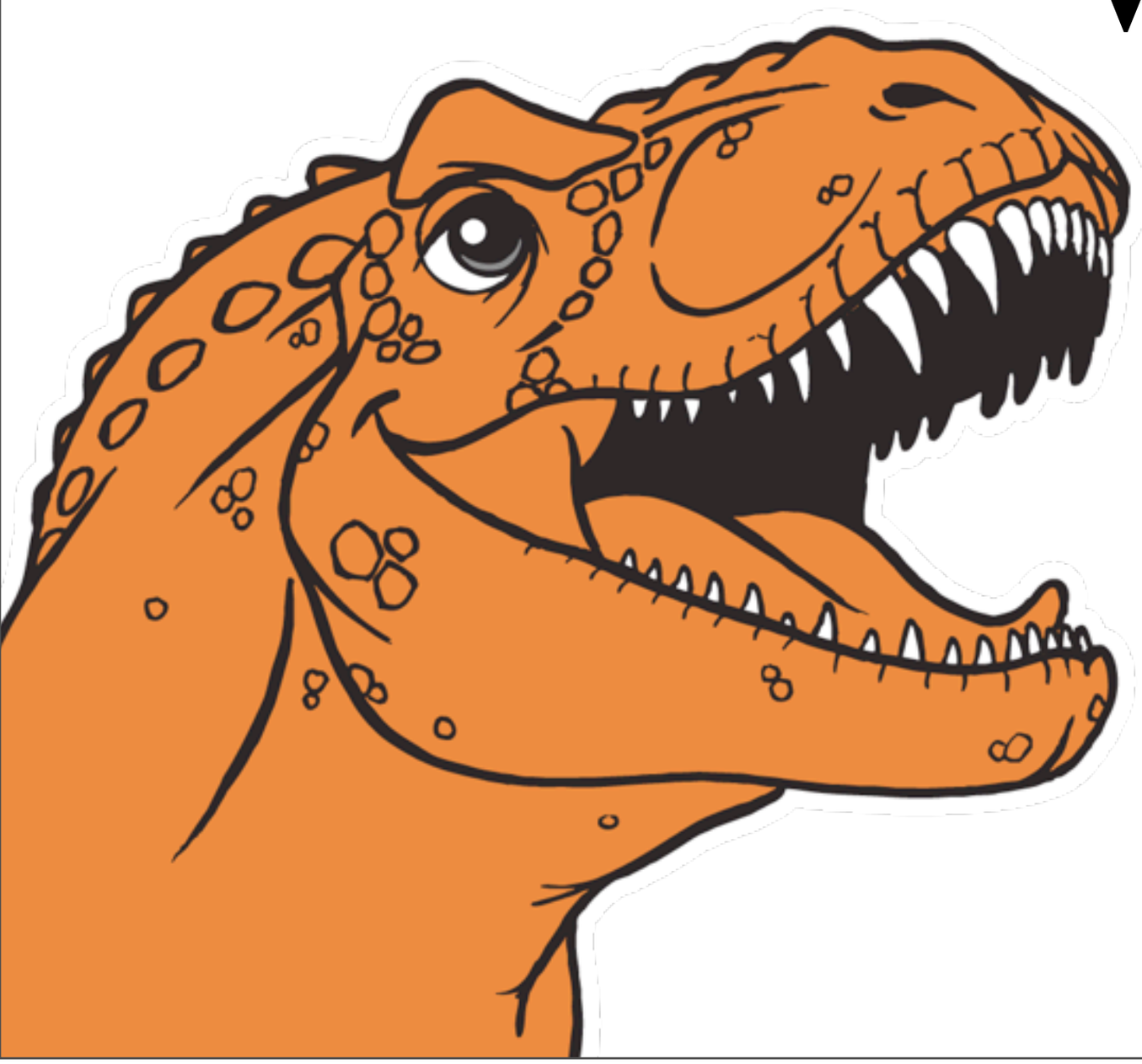
Links

- Aaron Patterson on Arel 2.0 - http://web.archive.org/web/20101113093529/http://engineering.attinteractive.com/2010/10/arel-two-point-ohhhhh-yaaaaaa/
- History of Arel - https://speakerdeck.com/u/tomstuart/p/relational-algebra-and-arel

Thanks!

Rob Mack, Spiceworks
@robmack
rob@robmack.com

Obligatory "We're Hiring!" Slide



SPICEWORKS™
IT'S EVERYTHING IT