Arel Intro

OR

Raw is For Sushi, Not SQL

Disclaimer

Nobody’s insulting ActiveRecord

I’m well aware that raw SQL works just fine

Arel won’t fix all the things forever

In short, this isn’t this week’s front-end framework.

Skepticism: a taxonomy

Devs who are already experienced with SQL:

- Raw SQL strings are NBD; you just interpolate as needed
- No performance gain
- It just converts everything to SQL anyway, so why bother

Devs who don’t have much SQL experience/don’t like dealing with the db/have been infected with Front-Endy thinking:

- Arel is weird and gross
- Active Record covers 90% of the use cases
- DON’T TAKE MY BLANKIE

Basic info about Arel (A RElational aLgebra)

AST: Abstract Syntax Tree

What ActiveRecord uses to assemble queries, translate to SQL

Introduced in Rails 3

DB Agnostic

Has now been absorbed fully into ActiveRecord; rumored to be going public with Rails 6

Good Use Cases

- implement query builders to compose queries dynamically
- modules that define frequently-used but complex joins / named functions
- with form objects: controllers that receive requests with variable selectors, ordering clauses, where filters over substantially similar joins:
- defining frequently-referenced values (booleans, sums, max, etc) that require calculation/comparison across tables or cols
    - Example: (use COALESCE)
    - Example: (use 2 subqueries with EXISTS)
    - Example: (from `iRonin.it`): fname, lname, users with nil in one or the other
- time-range scopes: https://github.com/rails/rails/issues/36761#issuecomment-520876203

The Query Builder Pattern

PORO that encapsulates all the querying logic in one place for composition and execution at request time

- More testable
- SRP-friendly
- More explicit
- Reusable: can group related subqueries, select statements, calculations, etc.

```
class BareBonesQueryBuilder
  include ArelFancyStuff

  attr_reader :query, :params

  def initialize(params={})
        @params = params
        @query = base_query
  end

  def base_query
    Album.joins(:songs, artists: :roles).join(review_join)
  end

  def review_join
    query.join(review_events_t, Arel::Nodes::OuterJoin)
      .on(album_t[:id].eq(review_events_t[:reviewable_id])
      .and(review_events_t[:reviewable_type].eq(“album”)
    ).join_sources
  end
```

For controller usage, say in an index action:
--create class method that handles review join, selects for certain review state)

For admin dashboard:
--class method with review joins, add group_by/order for review status

For distributor service:
--add further joins for required data, filters on state of that data, selects to pluck out what’s needed from each table, build metadata)

What Arel Can do: Joins that AR doesn’t support

Before Rails 5, LEFT OUTER JOIN was not supported.

What about polymorphic left outer join?

```
scope :some_poly_outer, -> {
        joins(“LEFT OUTER JOIN poly_target ON id = poly_target.id AND poly_target.targetable_type =
‘this table’s type’”
        }
```

Gross.

Even more gross

```
VehicleInspectionReport.select("defects.id AS defect_id,
                                defects.name AS description,
                                trucks.truck_number AS truck_number,
                                trailers.number AS trailer_number,
                                vehicle_inspection_reports.inspection_time AS inspection_time")
    .joins_with_inspected_items
    .joins("LEFT JOIN defects ON inspected_items.id = defects.associated_object_id
            AND defects.associated_object_type = inspected_items.object_type")
    .joins("LEFT JOIN trucks ON inspected_items.truck_id = trucks.id")
    .joins("LEFT JOIN trailers ON inspected_items.trailer_id = trailers.id")
    .where("inspected_items.id IS NOT NULL")
    .order('truck_number, trailer_number, inspection_time DESC')
```

Though I do apologize to Priyank, from whose blog I grabbed this; it's perfectly cromulent SQL. I just think it's … gross. Here's the original url: https://blog.bigbinary.com/2020/01/07/rails-multiple-polymorphic-joins.html

Arel allows you to use Ruby for all of it:

```
class Source << ApplicationRecord
  class << self
    def targetable_join
        source_table.join(
          target_table, Arel::Nodes::OuterJoin
        ).on(
          target_table[:targetable_id].eq(source_table[:id]).and(target_table[:targetable_type].eq(“source”)
        ).join_sources
    end

    def source_table
      @_source_table ||= self.arel_table
    end

    def target_table
      @_target_table ||= target_table.arel_table
    end
  end
end
```

Where this can become very useful

```
module ArelJunk
    def poly_left_outer_join(source, target, col1, col2)
        source.join(target_table, Arel::Nodes::OuterJoin)
        .on(target[:targetable_id].eq(source[:id]).and(target[:targetable_type].eq("source"))
        .join_sources
    end

    def source_table
       @_source_table ||= self.arel_table
    end
  end
end
```

```
class SourceTable << ApplicationRecord
  extend ArelJunk
        # And bob's yer uncle
end
```

Some Highlights

- Replace SQL strings with Ruby methods
  - More readable: easier to parse than SQL strings, especially for devs who aren't used to reading/writing a lot of SQL
  - Can be unit-tested separately
- Convenient comparison operators: eq, lt, lteq, gt, gteq
- Hash notation access to attributes (within query)
- Composable
- lintable, syntax validation

Greater Precision

- Predictable joins: AR's includes will choose outer, inner, or n+1 based on what you do with the rest of the query (preload and eager_load even worse)
  - Either your performance suffers or you gotta remember all these rules
  - Know exactly what you're getting when you write the query
- No more "select 'em all and let Rails sort 'em out"
  - If you want to get everything, you have to use Arel.star
  - Push more processing down to the db, which has been optimized for it
  - Particularly true for selects that include calculation, casting, formatting
- Avoid ambiguous columns:
  - has_many.belongs_to.where(id: [some ids]) ← if you join, you can get ambiguous column errors
  - Or if you call .select after one of the AR joins, on, say, id, AR chooses last table's id

It’s Not Just Joins

Encapsulate some db functions in methods, include them in select:

def selects

```
[
  ‘this_table.attribute’,
  ‘joined_table.attribute AS whatever’,
  coalesce(this_table[:nullable_attribute], some_joined_table[:nullable_attribute], ← one approach:
source_table.project(source_table[:colum].sum)
        Another: source_table.project(FancyArelStuff)
  jsonify(songs_t[:title], songs_t[:performers]) <--just imagine we did some grouping that lets us roll
this stuff up like this
]
```

end

Arel::Nodes::NamedFunction -- The Good Stuff

```
def coalesce(arg1, arg2)
  Arel::Nodes::NamedFunction.new('COALESCE', [some_table[:nullable_field1], some_table[:nullable_field2]])
end

def jsonify(array1, array2)
  Arel::Nodes::NamedFunction.new('json_build_object', [['a', 'b'], ['c', 'd']
end

def json_aggregate(some_result_set)
  Arel::Nodes::NamedFunction.new("json_agg", [some_result_set]
End

def even_cooler_jsonify(array1, array2)
  Arel::Nodes::NamedFunction.new("json_build_object", [json_agg(a_b_results), json_agg(c_d_results)])
end
```

It’s Not All Beer & Skittles

From the rails team itself: It’s a private API, so they don’t make any promises about behavior

This continues to be contentious: https://github.com/rails/rails/issues/36761#issuecomment-515286636

The Documentation Isn’t Great

https://www.rubydoc.info/github/rails/arel/Arel/

-- seems to raise as many questions as it answers, if not more

-- it’s not crazy to look into source code to explain something; it’s just not ideal

-- Caveat Emptor when it comes to older examples (Arel::Table.new() deprecated, but lots of other parts of older write-ups are still valuable)

There’s a Learning Curve

- What’s the big deal with raw SQL, especially when you have to learn to read a new Frankensyntax that mixes ruby & SQL?
- AR covers 90% of use cases; DON’T MAKE US LEARN DB STUFF
- What the hell even is a QueryBuilder, and why can’t we just add more scopes?

It’s Verbose

scope :ugly_but_short, -> { joins(:something_else).where(“something_else.ugly IS TRUE and something_else.duration <= 5”}

Vs.

self.arel_table.join(SomethingElse.arel_table.on(SomethingElse.arel_table[:fk_id].eq(ThisTable.arel_table[:id])).where(SomethingElse.arel_table[:ugl].eq(true)).and(ThisTable.arel_table[:duration].lteq(5))

It gets longer with subjoins, named functions, sane things like memoizing ThisTable.arel_table.

Additional Resources: Docs & cheat sheets

Official docs: https://www.rubydoc.info/github/rails/arel/Arel/

Rails 6 merge: https://github.com/rails/rails/pull/32097

What's an AST: https://en.wikipedia.org/wiki/Abstract_syntax_tree

Visitor Pattern: https://en.wikipedia.org/wiki/Visitor_pattern

Cheat Sheet: https://github.com/rstacruz/cheatsheets/blob/master/arel.md

Intro: https://www.ironin.it/blog/intro-to-arel-the-database-agnostic-sql.html

Additional Resources: Should-be-Official

Classics:

- https://jpospisil.com/2014/06/16/the-definitive-guide-to-arel-the-sql-manager-for-ruby.html
- http://radar.oreilly.com/2014/05/more-than-enough-arel.html

Additional Resources: Walk-throughs & useful Blogs

Former coworker/tech-lead with an excellent walk-through: Composable Query Builders in Rails with Arel

https://blog.codeship.com/creating-advanced-active-record-db-queries-arel/

https://thoughtbot.com/blog/using-arel-to-compose-sql-queries

https://www.ironin.it/blog/intro-to-arel-the-database-agnostic-sql.html