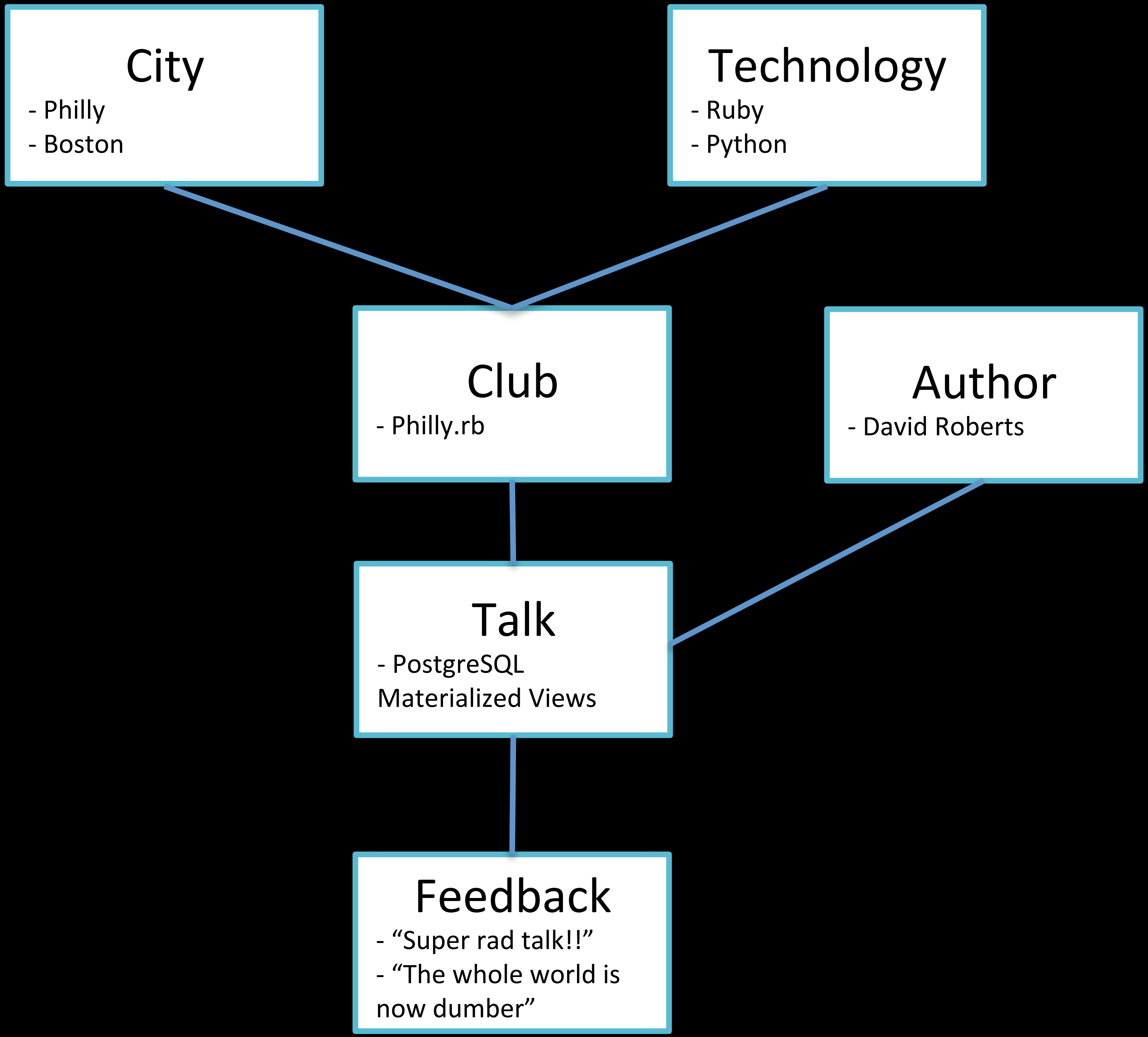
PostgreSQL Materialized Views

And Active Record

The Problem

How do you quickly report on data represented by multiple ActiveRecord associations?

Data Model


City
- Philly
- Boston
Technology
- Ruby
- Python
Club
- Philly.rb
Author
- David Roberts
Talk
- PostgreSQL Materialized Views
Feedback
- "Super rad talk!!"
- "The whole world is now dumber"

View all Comments

```
class Feedback < ActiveRecord::Base
  belongs_to :talk
  INVALID_COMMENTS = ['', 'NA', 'N/A', 'not
applicable']

  scope :filled_out,
    -> { where.not(comment: INVALID_COMMENTS) }
end

Feedback.filled_out
```

```
Feedback Load (2409.6ms)  SELECT "feedbacks".*
FROM "feedbacks"  WHERE ("feedbacks"."comment"
NOT IN ('', 'NA', 'N/A', 'not applicable'))
```

Filter Comments by Author

```
Feedback.filled_out.joins(talk: :author) \
  .where("authors.name = 'Rhiannon Parker'")
```

```
Feedback Load (442.1ms)  SELECT "feedbacks".*
FROM "feedbacks" INNER JOIN "talks" ON
"talks"."id" = "feedbacks"."talk_id" INNER
JOIN "authors" ON "authors"."id" =
"talks"."author_id" WHERE
("feedbacks"."comment" NOT IN ('', 'NA', 'N/
A', 'not applicable')) AND (authors.name =
'Rhiannon Parker')
```

Filter Comments by City

```
Feedback.filled_out \
  .joins(talk: { club: :city } ) \
  .where("cities.name = ‘Philadelphia'")
```

```
Feedback Load (711.1ms)  SELECT "feedbacks".*
FROM "feedbacks" INNER JOIN "talks" ON
"talks"."id" = "feedbacks"."talk_id" INNER JOIN
"clubs" ON "clubs"."id" = "talks"."club_id"
INNER JOIN "cities" ON "cities"."id" =
"clubs"."city_id" WHERE ("feedbacks"."comment"
NOT IN ('', 'NA', 'N/A', 'not applicable')) AND
(cities.name = 'Philadelphia')
```

Find comments containing “ipsum” from Ruby clubs for authors named “Parker”

```ruby
Feedback.filled_out \
  .joins(talk: [:author, { club: :city }] ) \
  .where("cities.name = 'Philadelphia'") \
  .where("feedbacks.comment LIKE '%ipsum%'") \
  .where("authors.name LIKE '%Parker%'")
```

```
Feedback Load (410.7ms)  SELECT "feedbacks".* FROM "feedbacks"
INNER JOIN "talks" ON "talks"."id" = "feedbacks"."talk_id"
INNER JOIN "authors" ON "authors"."id" = "talks"."author_id"
INNER JOIN "clubs" ON "clubs"."id" = "talks"."club_id" INNER
JOIN "cities" ON "cities"."id" = "clubs"."city_id" WHERE
("feedbacks"."comment" NOT IN ('', 'NA', 'N/A', 'not
applicable')) AND (cities.state_abbr = 'PA') AND
(feedbacks.comment LIKE '%ipsum%') AND (authors.name LIKE
'%Parker%')
```

Slow Queries do not get along with Web Applications

A Solution:
Materialized Views

Materialized Views

- Act similar to a Database View, but persists results for future queries
- Must be refreshed to be updated with most recent data

Creating a Materialized View in PostgreSQL 9.3

```
CREATE MATERIALIZED VIEW mv_feedback_report AS
        SELECT  cities.id as city_id,
                cities.name as city_name,
                cities.state_abbr as state_abbr,
                technologies.id as technology_id,
                clubs.id as club_id,
                clubs.name as club_name,
                talks.id as talk_id,
                talks.name as talk_name,
                authors.id as author_id,
                authors.name as author_name,
                feedbacks.id as feedback_id,
                feedbacks.score as score,
                feedbacks.comment as comment
        FROM feedbacks
        INNER JOIN talks ON feedbacks.talk_id = talks.id
        INNER JOIN authors ON talks.author_id = authors.id
        INNER JOIN clubs ON talks.club_id = talks.id
        INNER JOIN cities ON clubs.city_id = cities.id
        INNER JOIN technologies ON clubs.technology_id = technologies.id
        WHERE feedbacks.comment NOT IN ('', 'NA', 'N/A', 'not applicable')
```

Filter Comments by Author

50% reduction in runtime

```
# no materialized view - 520ms
  SELECT "feedbacks".* FROM "feedbacks"
  INNER JOIN "talks" ON "talks"."id" = "feedbacks"."talk_id"
  INNER JOIN "authors" ON "authors"."id" = "talks"."author_id"
  WHERE ("feedbacks"."comment" NOT IN ('', 'NA', 'N/A', 'not
applicable'))
  AND (authors.name = 'Rhiannon Parker');

  # materialized view - 265ms
  SELECT * FROM mv_feedback_report where author_name = 'Rhiannon
Parker';
```

Filter Comments by City

66% reduction in runtime

```
# no materialized view - 600ms
  SELECT "feedbacks".* FROM "feedbacks"
  INNER JOIN "talks" ON "talks"."id" = "feedbacks"."talk_id"
  INNER JOIN "clubs" ON "clubs"."id" = "talks"."club_id"
  INNER JOIN "cities" ON "cities"."id" = "clubs"."city_id"
  WHERE ("feedbacks"."comment" NOT IN ('', 'NA', 'N/A', 'not
applicable'))
  AND (cities.name = 'Philadelphia');

  # materialized view - 200ms
  SELECT * FROM mv_feedback_report WHERE city_name = 'Philadelphia';
```

But this is a Ruby talk!

ActiveRecord Migration

```ruby
class CreateFeedbackReportMv < ActiveRecord::Migration
  def up
    connection.execute <<-SQL
      CREATE MATERIALIZED VIEW mv_feedback_report AS
        SELECT  cities.id as city_id,
                cities.name as city_name,
                cities.state_abbr as state_abbr,
                technologies.id as technology_id,
                clubs.id as club_id,
                clubs.name as club_name,
                talks.id as talk_id,
                talks.name as talk_name,
                authors.id as author_id,
                authors.name as author_name,
                feedbacks.id as feedback_id,
                feedbacks.score as score,
                feedbacks.comment as comment
        FROM feedbacks
        INNER JOIN talks ON feedbacks.talk_id = talks.id
        INNER JOIN authors ON talks.author_id = authors.id
        INNER JOIN clubs ON talks.club_id = clubs.id
        INNER JOIN cities ON clubs.city_id = cities.id
        INNER JOIN technologies ON clubs.technology_id = technologies.id
        WHERE feedbacks.comment NOT IN ('', 'NA', 'N/A', 'not applicable')
    SQL
  end

  def down
    connection.execute 'DROP MATERIALIZED VIEW IF EXISTS mv_feedback_report'
  end
end
```

Create a Model

Just like any other model!

```ruby
# Used for reporting only
class FeedbackReport < ActiveRecord::Base
  # Use associations just like any other ActiveRecord object
  belongs_to :feedback
  belongs_to :author
  belongs_to :talk
  belongs_to :club
  belongs_to :city
  belongs_to :technology

  self.table_name = 'mv_feedback_report'

  def self.repopulate
    connection.execute("REFRESH MATERIALIZED VIEW #{table_name}")
  end

  # materialized views cannot be changed
  def readonly
    true
  end
end
```

Downsides

- Requires PostgreSQL 9.3
- Entire Materialized View must be refreshed to update
    - Bad when Live Data is required
    - For this use case, roll your own Materialized View using standard tables

Downsides

- Migrations are painful!
  - Recommend writing in SQL, so no using scopes
  - Entire Materialized View must be dropped and redefined for any changes to the View or referring tables
  - Hard to read and track what changed

Use Materialized Views

- For fast / live queries of complex associations or calculated fields
- When up to the minute data is not critical
- When Performance is more important than Storage
- Create a corresponding ActiveRecord model for easy use in Rails

Resources

- Source Code used in Talk
  - https://github.com/droberts84/materialized-view-demo
- PostgreSQL Documentation
  - https://wiki.postgresql.org/wiki/Materialized_Views