Active Record

LIBRARY & PATTERN

Bob Lail

Bob Lail

Advocate Engineering

Active Record

LIBRARY & PATTERN

GOALS

I. See alternatives to the pattern

II. How to use the library when you want flexibility around the pattern

PARTS

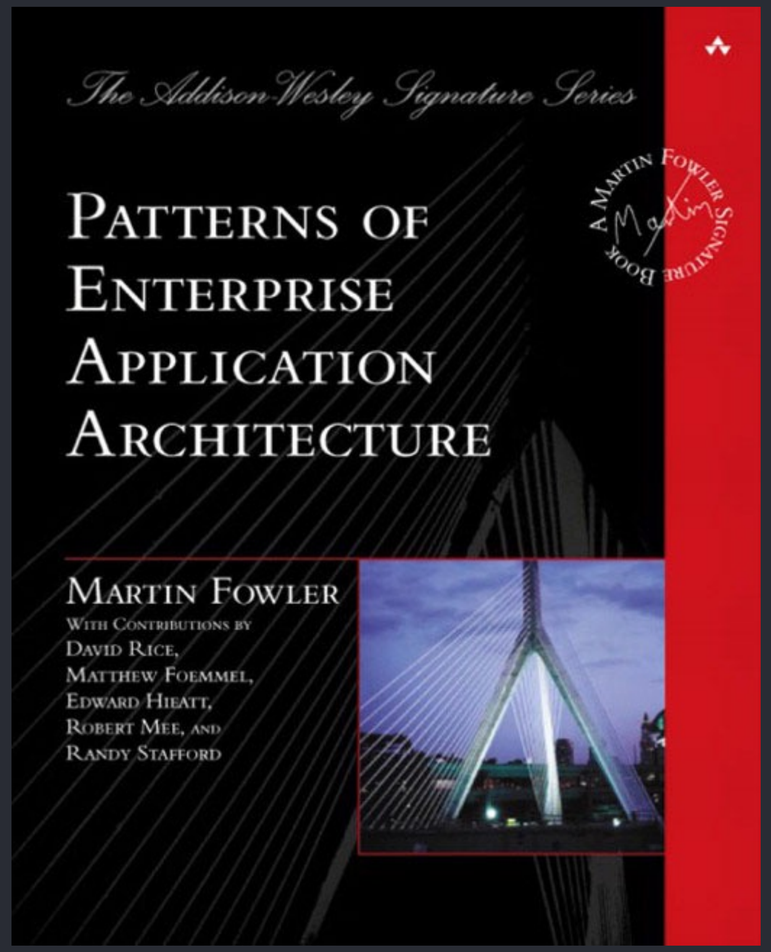
I. Strengths & Weaknesses

II. Resolving the Weaknesses

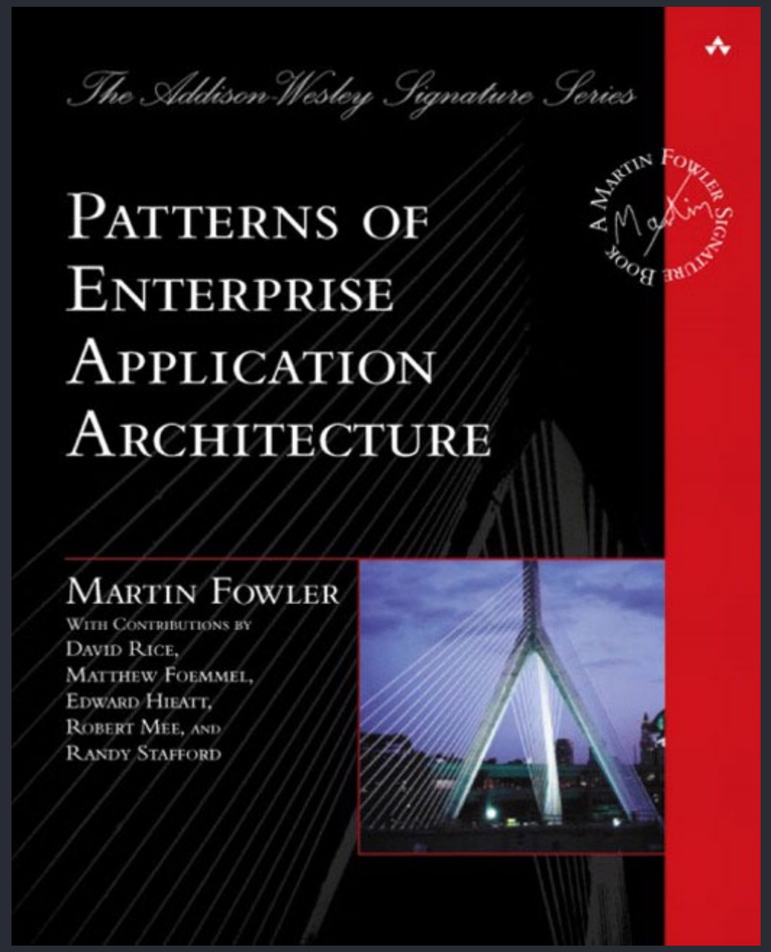
III. Investment Advice

Active Record

STRENGTHS & WEAKNESSES



- Active Record
- Application Controller
- Coarse-Grained Lock
- Data Mapper
- Data Transfer Object
- Database Session State
- Domain Model
- Foreign Key Mapping
- Gateway
- Identity Map
- Implicit Lock
- Lazy Load
- Metadata Mapping
- Model View Controller
- Page Controller
- Query Object
- Record Set
- Registry
- Repository
- Row Data Gateway
- Service Layer
- Service Stub
- Table Data Gateway
- Table Module
- Template View
- Transaction Script
- Unit of Work
- Value Object



Active Record
Application Controller
Coarse-Grained Lock
Data Mapper
Data Transfer Object
Database Session State
Domain Model
Foreign Key Mapping
Gateway
Identity Map
Implicit Lock
Lazy Load
Metadata Mapping
Model View Controller
Page Controller
Query Object
Record Set
Registry
Repository
Row Data Gateway
Service Layer
Service Stub
Table Data Gateway
Table Module
Template View
Transaction Script
Unit of Work
Value Object

RAILS

Blog Guides API Ask for help Contribute on GitHub

Imagine what you could build if you learned Ruby on Rails...

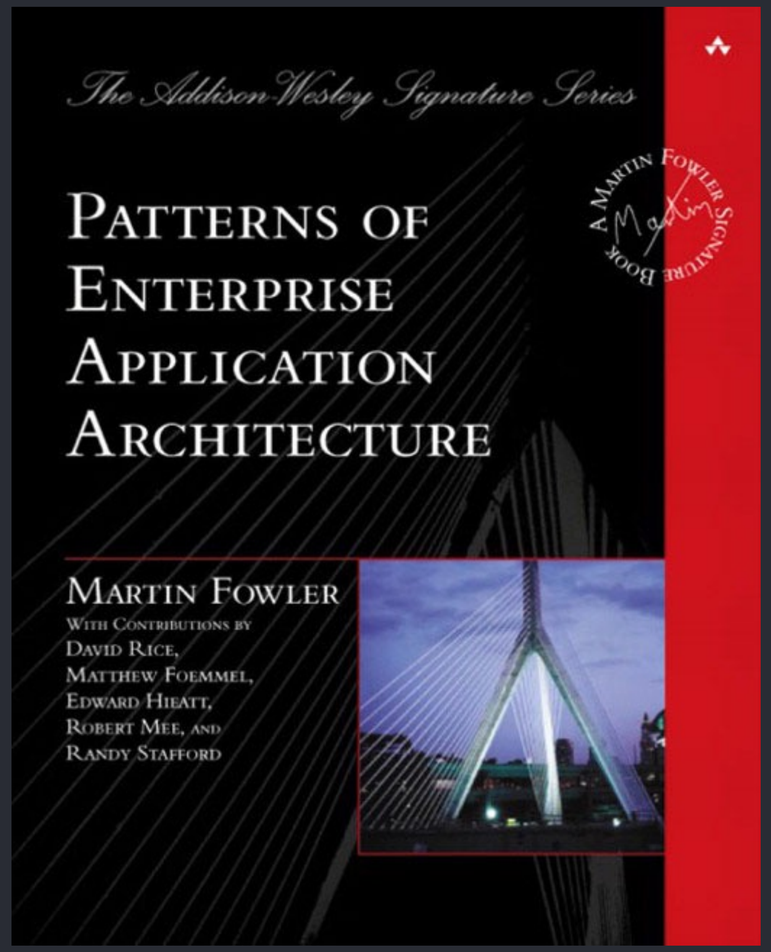
Learning to build a modern web application is daunting. Ruby on Rails makes it much easier and more fun. It includes everything you need to build fantastic applications, and you can learn it with the support of our large, friendly community.

Latest version — Rails 5.2.3 released March 28, 2019

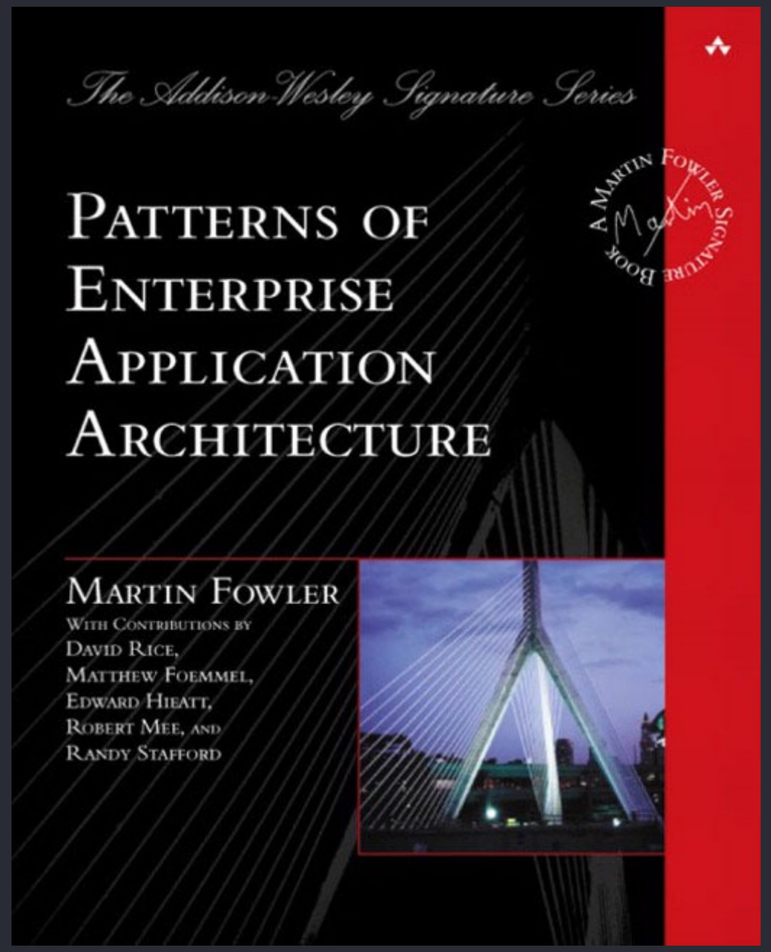
The five programming books that meant most to me

David wrote this on Dec 26 2012 / 28 comments

There are so many programming books out there, but most focus on specific technologies and their half-life is incredibly short. Others focus on process or culture. Very few focus on the timeless principles of writing good code, period. The following five books had the biggest influence on my programming style and development:



Active Record
Application Controller
Coarse-Grained Lock
Data Mapper
Data Transfer Object
Database Session State
Domain Model
Foreign Key Mapping
Gateway
Identity Map
Implicit Lock
Lazy Load
Metadata Mapping
Model View Controller
Page Controller
Query Object
Record Set
Registry
Repository
Row Data Gateway
Service Layer
Service Stub
Table Data Gateway
Table Module
Template View
Transaction Script
Unit of Work
Value Object



An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.

```ruby
class Book < ActiveRecord::Base
end
Book.table_name # => "books"
```

```ruby
class Book < ActiveRecord::Base
end
Book.table_name # => "books"


book = Book.find_by(title: "The Chosen")
# SELECT * FROM books WHERE title = 'The Chosen' LIMIT 1
```

```ruby
class Book < ActiveRecord::Base
end
Book.table_name # => "books"


book = Book.find_by(title: "The Chosen")
# SELECT * FROM books WHERE title = 'The Chosen' LIMIT 1

book = Book.create(title: "My Name is Asher Lev")
# INSERT INTO books (title) VALUES ('My Name is Asher Lev')

book.update(publication_date: Date.new(1967, 4, 28))
# UPDATE books SET publication_date = '1967-04-28' WHERE id = 1

book.destroy
# DELETE FROM books WHERE id = 1
```

```ruby
class Book < ActiveRecord::Base
  belongs_to :author
end

book = Book.find(1)
# SELECT * FROM books WHERE id = 1 LIMIT 1

book.author.name # => "Chiam Potok"
# SELECT * FROM authors WHERE id = 3 LIMIT 1
```

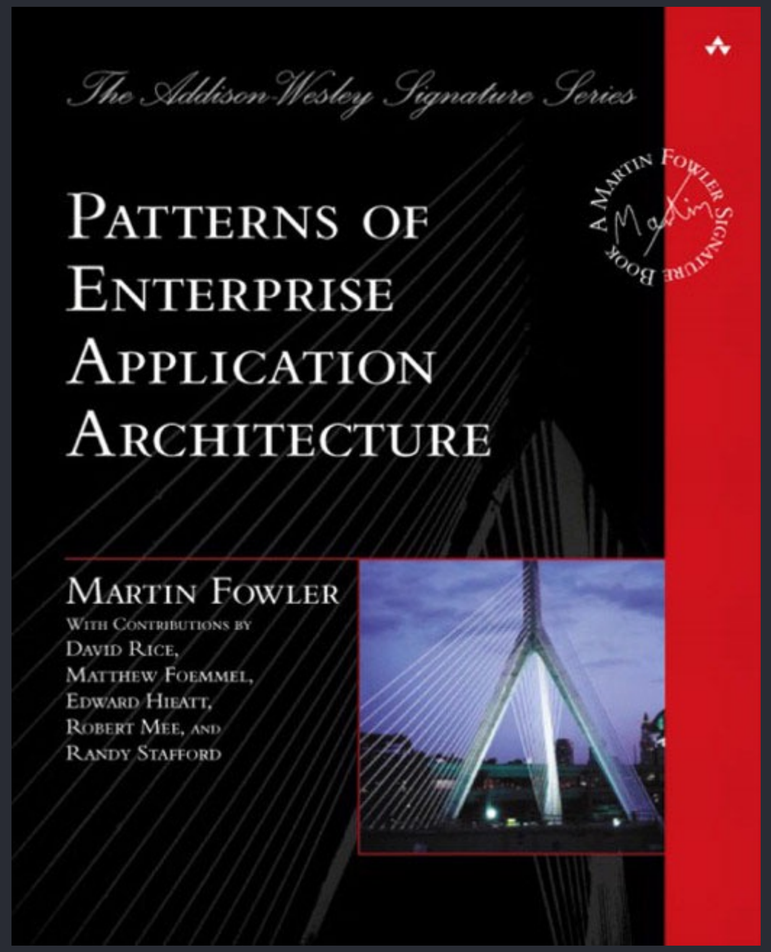
```ruby
class Book < ActiveRecord::Base
  validates :title, presence: true
end

Book.new(title: nil).valid? # => false
Book.new(title: nil).errors.to_h # => { title: "can't be blank" }
```

```ruby
class Author < ActiveRecord::Base
  after_create :ensure_author_has_a_biography!

  def ensure_author_has_a_biography!
    Biography.create!(author: self)
  end
end

class Biography < ActiveRecord::Base
  belongs_to :author
end

Author.create!(name: "Alan Paton")
# BEGIN
# INSERT INTO authors (name) VALUES ('Alan Paton')
# INSERT INTO biographies (author_id) VALUES (2)
# COMMIT
```

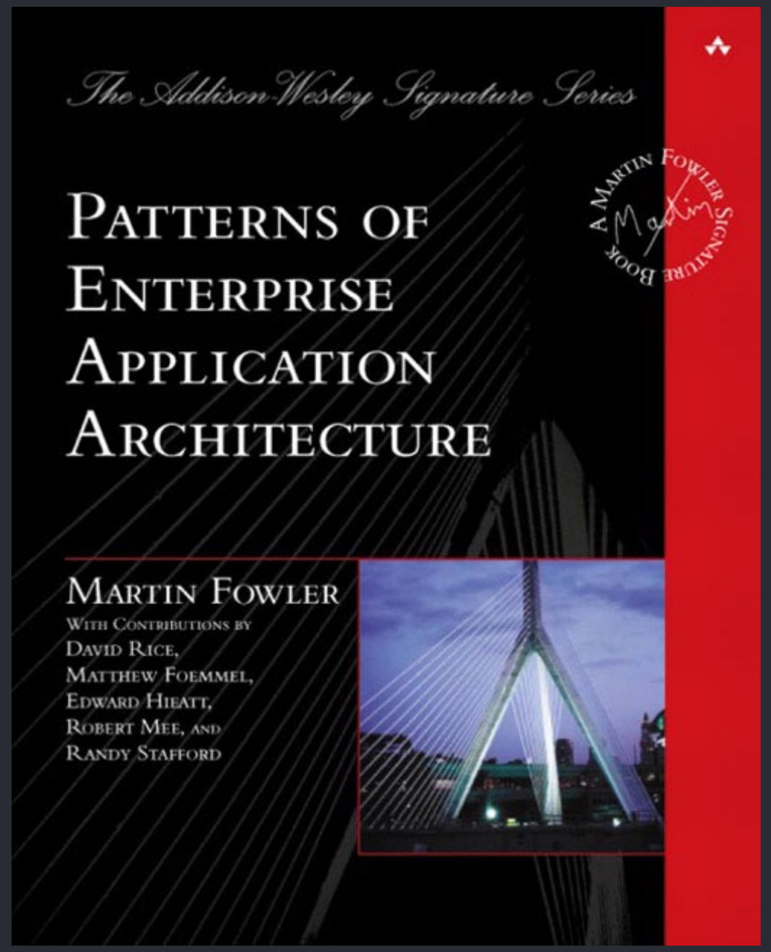
```ruby
Book.where(title: "The Chosen")
# SELECT * FROM books WHERE title = 'The Chosen'

Book.where(title: ["The Chosen", "My Name is Asher Lev"])
# SELECT * FROM books
#   WHERE title IN ('The Chosen', 'My Name is Asher Lev')

Book.where(publication_date: Date.new(1967, 1, 1)..Date.new(1977, 12, 31))
# SELECT * FROM books
#   WHERE publication_date BETWEEN '1967-01-01' AND '1977-12-31'
```

```ruby
Book.where(title: "The Chosen")                                    # => [ #<Book ...> ]
# SELECT * FROM books WHERE title = 'The Chosen'

Book.where(title: ["The Chosen", "My Name is Asher Lev"])          # => [ #<Book ...> ]
# SELECT * FROM books
#   WHERE title IN ('The Chosen', 'My Name is Asher Lev')

Book.where(publication_date: Date.new(1967, 1, 1)..Date.new(1977, 12, 31))  # => [ #<Book ...> ]
# SELECT * FROM books
#   WHERE publication_date BETWEEN '1967-01-01' AND '1977-12-31'
```

```ruby
Book.where(title: "The Chosen")                                    # => ActiveRecord::Relation
# SELECT * FROM books WHERE title = 'The Chosen'

Book.where(title: ["The Chosen", "My Name is Asher Lev"])          # => ActiveRecord::Relation
# SELECT * FROM books
#   WHERE title IN ('The Chosen', 'My Name is Asher Lev')

Book.where(publication_date: Date.new(1967, 1, 1)..Date.new(1977, 12, 31))  # => ActiveRecord::Relation
# SELECT * FROM books
#   WHERE publication_date BETWEEN '1967-01-01' AND '1977-12-31'
```

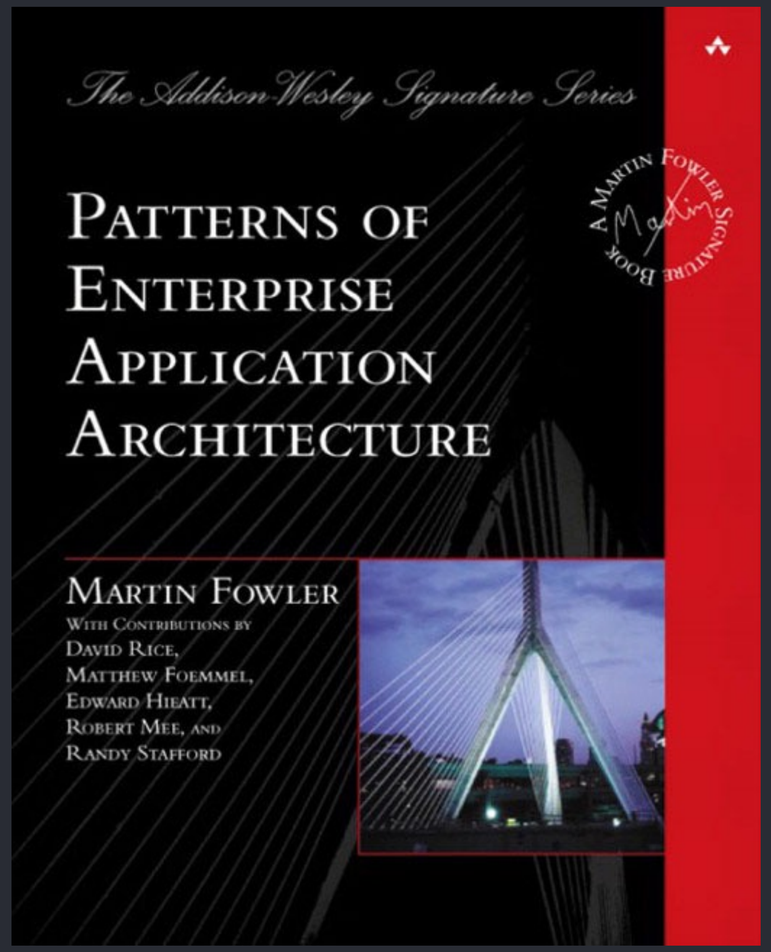


An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.

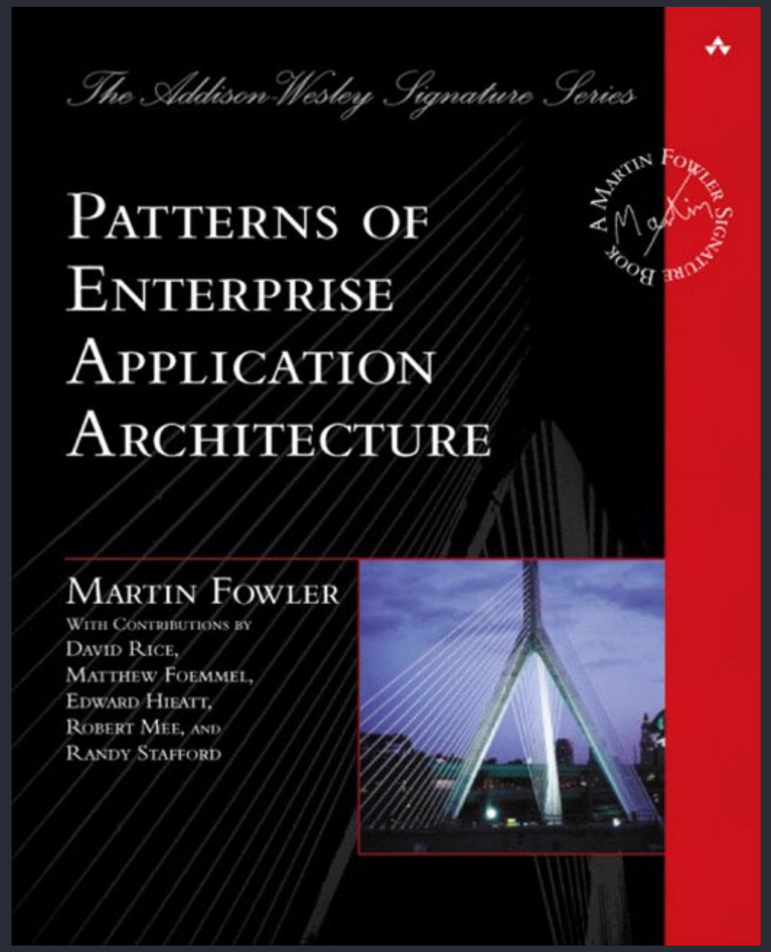


An object that ***wraps a row*** *in a database table or view, encapsulates the database access, and adds domain logic on that data.*

```sql
-- Author, title and most recent purchase price
-- of books that weren't bought in the last 30 days
SELECT
  authors.name,
  books.title,
  orders.price
FROM books
INNER JOIN authors ON books.author_id = authors.id
INNER JOIN orders ON orders.book_id = books.id
INNER JOIN (
    SELECT book_id, MAX(created_at) AS created_at
    FROM orders
    GROUP BY book_id
  ) AS latest_order
  ON orders.book_id = latest_order.book_id
  AND orders.created_at = latest_order.created_at
WHERE orders.created_at < NOW() - INTERVAL 30 DAY;
```



*An object that **wraps a row** in a database table or view, encapsulates the database access, and adds domain logic on that data.*



An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic *on that data.*

	Author Creator	Author
Every author needs a biography (even if it's empty)		
We can't save an author without knowing their name		
What is an author's current age?		
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	
We can't save an author without knowing their name		
What is an author's current age?		
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name		
What is an author's current age?		
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name	✔	
What is an author's current age?		
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name	✔	✔
What is an author's current age?		
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name	✔	✔
What is an author's current age?	✖	
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name	✔	✔
What is an author's current age?	✘	✔
What's an author's address? (for a mailing label)		
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✔	✔
We can't save an author without knowing their name	✔	✔
What is an author's current age?	✘	✔
What's an author's address? (for a mailing label)	✘	
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✓	✓
We can't save an author without knowing their name	✓	✓
What is an author's current age?	✗	✓
What's an author's address? (for a mailing label)	✗	✓
How much has an author been paid in royalties?		

	Author Creator	Author
Every author needs a biography (even if it's empty)	✓	✓
We can't save an author without knowing their name	✓	✓
What is an author's current age?	✗	✓
What's an author's address? (for a mailing label)	✗	✓
How much has an author been paid in royalties?	✗	

	Author Creator	Author
Every author needs a biography (even if it's empty)	✓	✓
We can't save an author without knowing their name	✓	✓
What is an author's current age?	✗	✓
What's an author's address? (for a mailing label)	✗	✓
How much has an author been paid in royalties?	✗	✓

	Author Creator	Author	Author Helper
Every author needs a biography (even if it's empty)	✓	✓	✓
We can't save an author without knowing their name	✓	✓	✓
What is an author's current age?	✗	✓	✓
What's an author's address? (for a mailing label)	✗	✓	✓
How much has an author been paid in royalties?	✗	✓	✓

WEAKNESSES

1. Working with Sets
2. Composite Results
3. Separating Responsibilities

Active Record

RESOLVING THE WEAKNESSES

WORKING WITH SETS

```ruby
Book.all.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id = 5 LIMIT 1
# SELECT * FROM authors WHERE id = 13 LIMIT 1
# SELECT * FROM authors WHERE id = 11 LIMIT 1
# SELECT * FROM authors WHERE id = 7 LIMIT 1
# SELECT * FROM authors WHERE id = 35 LIMIT 1
# SELECT * FROM authors WHERE id = 14 LIMIT 1
# SELECT * FROM authors WHERE id = 2 LIMIT 1
# SELECT * FROM authors WHERE id = 22 LIMIT 1
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```ruby
books = Book.all
author_ids = books.map(&:author_id)
authors_by_id = Author.where(id: author_ids).index_by(&:id)
books.each do |book|
  book.author = authors_by_id[book.author_id]
end

books.each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)


Book.all.preload(:author).each do |book|
  puts "#{book.title} by #{book.author.name}"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)
```

```
Book.preload(:author)
Book.eager_load(:author)
Book.includes(:author)
```

```ruby
Book.preload(:author)
Book.eager_load(:author)
Book.includes(:author)

Book.preload(:author).find_each do |book|
  puts "#{book.title} (#{book.author.name})"
end
# SELECT * FROM books
# SELECT * FROM authors WHERE id IN (5, 13, 11, 7, 35, 14, 2, 22)

Book.eager_load(:author).each do |book|
  puts "#{book.title} (#{book.author.name})"
end
# SELECT
#   books.id AS t0_r0,
#   books.title AS t0_r1,
#   books.author_id AS t0_r2,
#   authors.id AS t1_r0,
#   authors.name AS t1_r1
# FROM books
# LEFT OUTER JOIN authors ON authors.id = books.id
```

```
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.preload(:author)
      .map { |book|
        { id: book.id,
          title: book.title,
          publicationDate: book.publication_date,
          author: {
            id: book.author.id,
            name: book.author.name } } })
  end
end

Book.benchmark "BookPresenter.1" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.preload(:author)
      .map { |book|
        { id: book.id,
          title: book.title,
          publicationDate: book.publication_date,
          author: {
            id: book.author.id,
            name: book.author.name } } })
  end
end

Book.benchmark "BookPresenter.1" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.preload(:author)
      .map { |book|
        { id: book.id,
          title: book.title,
          publicationDate: book.publication_date,
          author: {
            id: book.author.id,
            name: book.author.name } } })
  end
end

Book.benchmark "BookPresenter.1" do
  BookPresenter.new(Book.all).to_json
end
```

```
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.preload(:author)
      .map { |book|
        { id: book.id,
          title: book.title,
          publicationDate: book.publication_date,
          author: {
            id: book.author.id,
            name: book.author.name } } })
  end
end

Book.benchmark "BookPresenter.1" do
  BookPresenter.new(Book.all).to_json
end
```

```
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.preload(:author)
      .map { |book|
        { id: book.id,
          title: book.title,
          publicationDate: book.publication_date,
          author: {
            id: book.author.id,
            name: book.author.name } } })
  end
end

Book.benchmark "BookPresenter.1" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.joins(:author)
      .pluck(:id, :title, :publication_date, "authors.id", "authors.name")
      .map { |id, title, publication_date, author_id, author_name|
        { id: id,
          title: title,
          publicationDate: publication_date,
          author: {
            id: author_id,
            name: author_name } } })
  end
end

Book.benchmark "BookPresenter.2" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.joins(:author)
      .pluck(:id, :title, :publication_date, "authors.id", "authors.name")
      .map { |id, title, publication_date, author_id, author_name|
        { id: id,
          title: title,
          publicationDate: publication_date,
          author: {
            id: author_id,
            name: author_name } } })
  end
end

Book.benchmark "BookPresenter.2" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.joins(:author)
      .pluck(:id, :title, :publication_date, "authors.id", "authors.name")
      .map { |id, title, publication_date, author_id, author_name|
        { id: id,
          title: title,
          publicationDate: publication_date,
          author: {
            id: author_id,
            name: author_name } } })
  end
end

Book.benchmark "BookPresenter.2" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    JSON.dump(books.joins(:author)
      .pluck(:id, :title, :publication_date, "authors.id", "authors.name")
      .map { |id, title, publication_date, author_id, author_name|
        { id: id,
          title: title,
          publicationDate: publication_date,
          author: {
            id: author_id,
            name: author_name } } })
  end
end

Book.benchmark "BookPresenter.2" do
  BookPresenter.new(Book.all).to_json
end
```

```ruby
BookPresenter = Struct.new(:books) do
  def to_json
    query = books.joins(:author).select(<<~SQL).to_sql
      json_object(
        'id', books.id,
        'title', books.title,
        'publicationDate', books.publication_date,
        'author', json_object(
          'id', authors.id,
          'name', authors.name) ) AS object
    SQL

    Book.connection.select_value(<<~SQL)
      SELECT json_arrayagg(d.object) FROM (#{query}) AS d
    SQL
  end
end

Book.benchmark "BookPresenter.3" do
  BookPresenter.new(Book.all).to_json
end
```

```
BookPresenter = PluckMap[Book].define do
  id
  title
  publicationDate select: :publication_date
  has_one :author do
    id
    name
  end
end

Book.benchmark "BookPresenter.4" do
  BookPresenter.new(Book.all).to_json
end
```

github.com/boblail/pluck_map

```
Comparison:
          Pluck Json:     1970.0 i/s
  to_json__optimized:     1359.3 i/s - 1.45x  slower
   Pluck/Map Pattern:     1237.3 i/s - 1.59x  slower
             PluckMap:     1069.8 i/s - 1.84x  slower
        ActiveRecord:      433.2 i/s - 4.55x  slower

Comparison:
          Pluck Json:      10225 allocated
  to_json__optimized:      20434 allocated - 2.00x more
   Pluck/Map Pattern:     116533 allocated - 11.40x more
             PluckMap:     120781 allocated - 11.81x more
        ActiveRecord:     242973 allocated - 23.76x more
```

github.com/boblail/pluck_map/blob/master/test/benchmarks.rb

```ruby
Book.where(author_id: 3).each do |book|
  book.delete
end
# SELECT * FROM books WHERE author_id = 3
# DELETE FROM books WHERE id = 19
# DELETE FROM books WHERE id = 23
# DELETE FROM books WHERE id = 28
```

```ruby
Book.where(author_id: 3).each do |book|
  book.delete
end
# SELECT * FROM books WHERE author_id = 3
# DELETE FROM books WHERE id = 19
# DELETE FROM books WHERE id = 23
# DELETE FROM books WHERE id = 28


Book.where(author_id: 3).delete_all
# DELETE FROM books WHERE author_id = 3
```

```ruby
Book.where(author_id: 3).each do |book|
  book.delete
end
# SELECT * FROM books WHERE author_id = 3
# DELETE FROM books WHERE id = 19
# DELETE FROM books WHERE id = 23
# DELETE FROM books WHERE id = 28


Book.where(author_id: 3).delete_all
# DELETE FROM books WHERE author_id = 3


Book.where(author_id: 3).destroy_all
# SELECT * FROM books WHERE author_id = 3
# DELETE FROM books WHERE id = 19
# DELETE FROM books WHERE id = 23
# DELETE FROM books WHERE id = 28
```

```ruby
Book.where(author_id: 3).each do |book|
  book.update(author_id: 21)
end
# SELECT * FROM books WHERE author_id = 3
# UPDATE books SET author_id = 21 WHERE id = 19
# UPDATE books SET author_id = 21 WHERE id = 23
# UPDATE books SET author_id = 21 WHERE id = 28
```

```ruby
Book.where(author_id: 3).each do |book|
  book.update(author_id: 21)
end
# SELECT * FROM books WHERE author_id = 3
# UPDATE books SET author_id = 21 WHERE id = 19
# UPDATE books SET author_id = 21 WHERE id = 23
# UPDATE books SET author_id = 21 WHERE id = 28


Book.where(author_id: 3).update_all(author_id: 21)
# UPDATE books SET author_id = 21 WHERE author_id = 3
```

```ruby
ordered_ids = params[:sequence] # => [6, 3, 17]

Book.transaction do
  ordered_ids.each_with_index do |id, i|
    book = Book.find(id)
    book.update_column :position, i
  end
end
# BEGIN
# SELECT * FROM books WHERE id = 6
# UPDATE books SET position = 0 WHERE id = 6
# SELECT * FROM books WHERE id = 3
# UPDATE books SET position = 1 WHERE id = 3
# SELECT * FROM books WHERE id = 17
# UPDATE books SET position = 2 WHERE id = 17
# COMMIT
```

```ruby
ordered_ids = params[:sequence] # => [6, 3, 17]

Book.transaction do
  ordered_ids.each_with_index do |id, i|
    Book.where(id: id).update_all(position: i)
  end
end
# BEGIN
# UPDATE books SET position = 0 WHERE id = 6
# UPDATE books SET position = 1 WHERE id = 3
# UPDATE books SET position = 2 WHERE id = 17
# COMMIT
```

```ruby
ordered_ids = params[:sequence] # => [6, 3, 17]

Book.where(id: ordered_ids).update_all([
  "position = ARRAY_POSITION(ARRAY[?], id)",
  ordered_ids
])
# UPDATE books
#   SET position = ARRAY_POSITION(ARRAY[6, 3, 17], id)
#   WHERE id IN (6, 3, 17)
```

```ruby
ordered_ids = params[:sequence] # => [6, 3, 17]

Book.where(id: ordered_ids).update_all([
  "position = ARRAY_POSITION(ARRAY[?], id)",
  ordered_ids
])
# UPDATE books
#   SET position = ARRAY_POSITION(ARRAY[6, 3, 17], id)
#   WHERE id IN (6, 3, 17)


Book.where(id: ordered_ids).update_all([
  "position = JSON_EXTRACT(?, CONCAT('$.', id))",
  ordered_ids.each_with_index.to_h.to_json
])
# UPDATE books
#   SET position = JSON_EXTRACT('{"6":0,"3":1,"17":2}', CONCAT('$.', id))
#   WHERE id IN (6, 3, 17)
```

```ruby
book = Book.create(title: "The Chosen")
# INSERT INTO books (title) VALUES ('The Chosen')
```

```ruby
book = Book.create(title: "The Chosen")
# INSERT INTO books (title) VALUES ('The Chosen')


books = Book.create([
  { title: "The Chosen" },
  { title: "The Promise" },
  { title: "My Name is Asher Lev" },
])
# INSERT INTO books (title) VALUES ('The Chosen')
# INSERT INTO books (title) VALUES ('The Promise')
# INSERT INTO books (title) VALUES ('My Name is Asher Lev')
```

```ruby
book = Book.create(title: "The Chosen")
# INSERT INTO books (title) VALUES ('The Chosen')


books = Book.create([
  { title: "The Chosen" },
  { title: "The Promise" },
  { title: "My Name is Asher Lev" },
])
# INSERT INTO books (title) VALUES ('The Chosen')
# INSERT INTO books (title) VALUES ('The Promise')
# INSERT INTO books (title) VALUES ('My Name is Asher Lev')


Book.insert_all([
  { title: "The Chosen" },
  { title: "The Promise" },
  { title: "My Name is Asher Lev" }
])
# INSERT INTO books (title) VALUES ('The Chosen'), ('The Promise'), ('My Name is Asher Lev')
```

```ruby
books_attributes.each do |attributes|
  book = Book.find_by(isbn: attributes[:isbn])
  if book
    book.update(attributes)
  else
    Book.create(attributes)
  end
end
# SELECT * FROM books WHERE isbn = '978-0449213445' LIMIT 1
# UPDATE books SET title = 'The Chosen' WHERE id = 1
# SELECT * FROM books WHERE isbn = '978-1400095414' LIMIT 1
# UPDATE books SET title = 'The Promise' WHERE id = 2
# SELECT * FROM books WHERE isbn = '978-1400031047' LIMIT 1
# INSERT INTO books (title, isbn) VALUES ('My Name is Asher Lev', '978-1400031047')
```

```ruby
books_attributes.each do |attributes|
  book = Book.find_or_initialize_by(isbn: attributes[:isbn])
  book.attributes = attributes
  book.save
end
# SELECT * FROM books WHERE isbn = '978-0449213445'
# UPDATE books SET title = 'The Chosen' WHERE id = 1
# SELECT * FROM books WHERE isbn = '978-1400095414'
# UPDATE books SET title = 'The Promise' WHERE id = 2
# SELECT * FROM books WHERE isbn = '978-1400031047'
# INSERT INTO books (title, isbn) VALUES ('My Name is Asher Lev', '978-1400031047')
```

```ruby
class AddUniqueIndexOnBooksIsbn < ActiveRecord::Migration[5.2]
  def change
    add_index :books, :isbn, unique: true
  end
end

Book.upsert_all(books_attributes)

# INSERT INTO books (title, isbn) VALUES
#     ('The Chosen', '978-0449213445'),
#     ('The Promise', '978-1400095414'),
#     ('My Name is Asher Lev', '978-1400031047')
#   ON DUPLICATE KEY UPDATE title = title, isbn = isbn
```

COMPOSITE RESULTS

```sql
-- Author, title and most recent purchase price
-- of books that weren't bought in the last 30 days
SELECT
  authors.name,
  books.title,
  orders.price
FROM books
INNER JOIN authors ON books.author_id = authors.id
INNER JOIN orders ON orders.book_id = books.id
INNER JOIN (
    SELECT book_id, MAX(created_at) AS created_at
    FROM orders
    GROUP BY book_id
  ) AS latest_order
  ON orders.book_id = latest_order.book_id
  AND orders.created_at = latest_order.created_at
WHERE orders.created_at < NOW() - INTERVAL 30 DAY;
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
class Book < ActiveRecord::Base
  belongs_to :author
  has_many :orders

  def latest_order
    @latest_order ||= orders.order(created_at: :desc).first
  end
end

Book.find_all { |book| book.latest_order.created_at < 30.days.ago }
  .map { |book| [book.author.name, book.title, book.latest_order.price] }
# SELECT * FROM books
# SELECT * FROM orders WHERE book_id = 1 ORDER BY created_at DESC LIMIT 1
# SELECT * FROM orders WHERE book_id = 2 ORDER BY created_at DESC LIMIT 1
# SELECT * FROM orders WHERE book_id = 3 ORDER BY created_at DESC LIMIT 1
# SELECT * FROM authors WHERE author_id = 1 LIMIT 1
# SELECT * FROM authors WHERE author_id = 2 LIMIT 1
# SELECT * FROM authors WHERE author_id = 3 LIMIT 1
```

```sql
-- Author, title and most recent purchase price
-- of books that weren't bought in the last 30 days
SELECT
  authors.name,
  books.title,
  orders.price
FROM books
INNER JOIN authors ON books.author_id = authors.id
INNER JOIN orders ON orders.book_id = books.id
INNER JOIN (
    SELECT book_id, MAX(created_at) AS created_at
    FROM orders
    GROUP BY book_id
  ) AS latest_order
  ON orders.book_id = latest_order.book_id
  AND orders.created_at = latest_order.created_at
WHERE orders.created_at < NOW() - INTERVAL 30 DAY;
```

```ruby
class Order < ActiveRecord::Base
  def self.latest
    joins(<<~SQL)
      INNER JOIN (SELECT book_id, MAX(created_at) created_at FROM orders GROUP BY book_id) latest_order
        ON orders.book_id = latest_order.book_id
        AND orders.created_at = latest_order.created_at
    SQL
  end

  def self.created_before(timestamp)
    where(arel_table[:created_at].lt(timestamp))
  end
end

class Book < ActiveRecord::Base
  has_many :orders

  def self.last_ordered_before(timestamp)
    joins(:orders).merge(Order.latest.created_before(timestamp))
  end
end
```

```ruby
class Order < ActiveRecord::Base
  def self.latest
    joins(<<~SQL)
      INNER JOIN (SELECT book_id, MAX(created_at) created_at FROM orders GROUP BY book_id) latest_order
        ON orders.book_id = latest_order.book_id
        AND orders.created_at = latest_order.created_at
    SQL
  end

  def self.created_before(timestamp)
    where(arel_table[:created_at].lt(timestamp))
  end
end

class Book < ActiveRecord::Base
  has_many :orders

  def self.last_ordered_before(timestamp)
    joins(:orders).merge(Order.latest.created_before(timestamp))
  end
end
```

```ruby
class Order < ActiveRecord::Base
  def self.latest
    joins(<<~SQL)
      INNER JOIN (SELECT book_id, MAX(created_at) created_at FROM orders GROUP BY book_id) latest_order
        ON orders.book_id = latest_order.book_id
        AND orders.created_at = latest_order.created_at
    SQL
  end

  def self.created_before(timestamp)
    where(arel_table[:created_at].lt(timestamp))
  end
end

class Book < ActiveRecord::Base
  has_many :orders

  def self.last_ordered_before(timestamp)
    joins(:orders).merge(Order.latest.created_before(timestamp))
  end
end
```

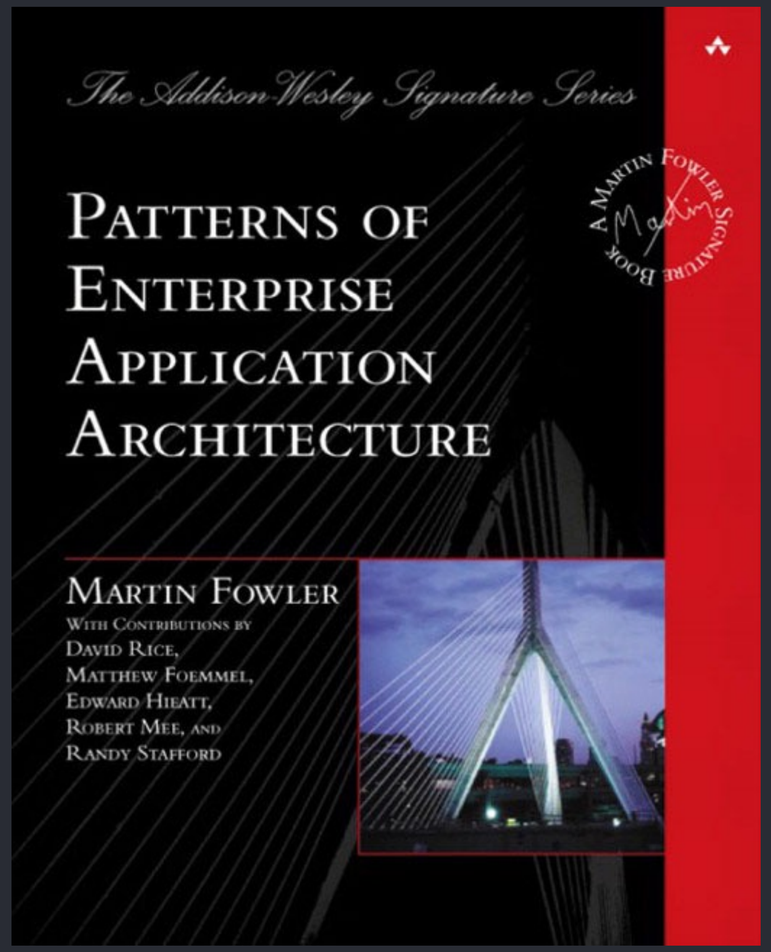
```ruby
class Order < ActiveRecord::Base
  def self.latest
    joins(<<~SQL)
      INNER JOIN (SELECT book_id, MAX(created_at) created_at FROM orders GROUP BY book_id) latest_order
        ON orders.book_id = latest_order.book_id
        AND orders.created_at = latest_order.created_at
    SQL
  end

  def self.created_before(timestamp)
    where(arel_table[:created_at].lt(timestamp))
  end
end

class Book < ActiveRecord::Base
  has_many :orders

  def self.last_ordered_before(timestamp)
    joins(:orders).merge(Order.latest.created_before(timestamp))
  end
end
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
# SELECT authors.name, books.title, orders.price
# FROM books
# INNER JOIN authors ON books.author_id = authors.id
# INNER JOIN orders ON orders.book_id = books.id
# INNER JOIN (SELECT book_id, MAX(created_at) created_at FROM orders GROUP BY book_id) latest_order
#   ON orders.book_id = latest_order.book_id
#   AND orders.created_at = latest_order.created_at
# WHERE orders.created_at < NOW() - INTERVAL 30 DAY;
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReport = Struct.new(:author, :title, :purchase_price)

Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
  .map(&SalesReport.method(:new))
```

```
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReport = Struct.new(:author, :title, :purchase_price)

Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
  .map(&SalesReport.method(:new))
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReport = Struct.new(:author, :title, :purchase_price)

Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
  .map(&SalesReport.method(:new))
```



A layer of Mappers that moves data between objects and a database while keeping them independent of each other and of the mapper itself.

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReport = Struct.new(:author, :title, :purchase_price)

Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
  .map(&SalesReport.method(:new))
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReport = Struct.new(:author, :title, :purchase_price)

Book
  .joins(:author)
  .last_ordered_before(30.days.ago)
  .pluck("authors.name", :title, "orders.price")
  .map(&SalesReport.method(:new))
```

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReportMapper = PluckMap[Book].define do
  author select: "authors.name"
  title
  purchase_price select: "orders.price"
end


SalesReport = SalesReportMapper::Struct


SalesReport.load(
  Book.joins(:author).last_ordered_before(30.days.ago)
)
```

github.com/boblail/pluck_map

```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReportMapper = PluckMap[Book].define do
  author select: "authors.name"
  title
  purchase_price select: "orders.price"
end


SalesReport = SalesReportMapper::Struct


SalesReport.load(
  Book.joins(:author).last_ordered_before(30.days.ago)
)
```

github.com/boblail/pluck_map

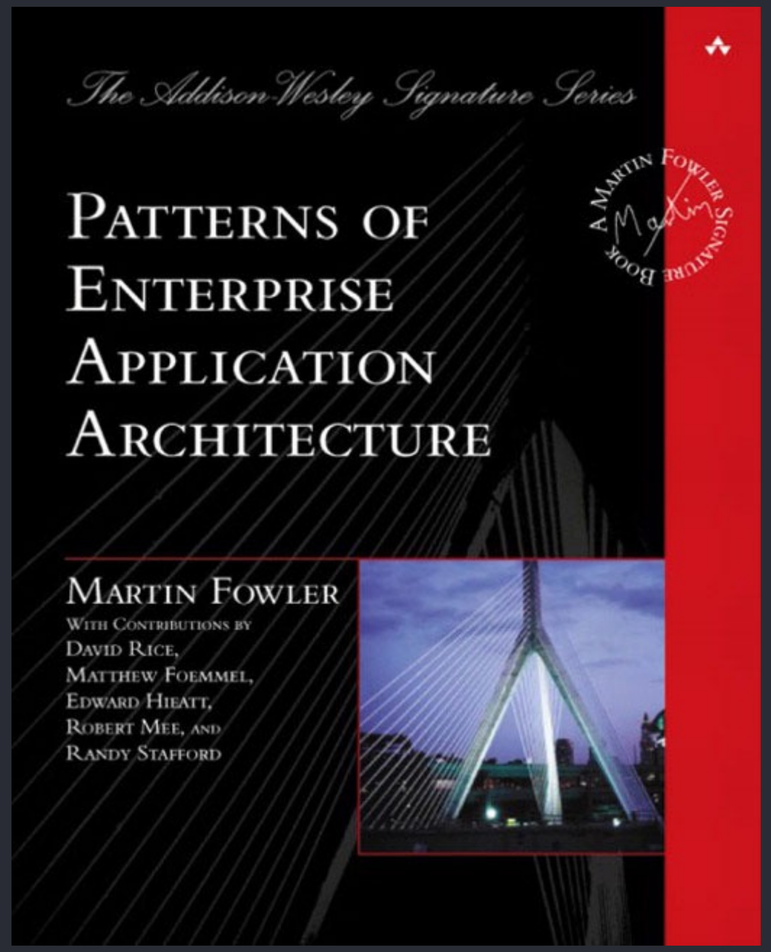
```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReportMapper = PluckMap[Book].define do
  author select: "authors.name"
  title
  purchase_price select: "orders.price"
end


SalesReport = SalesReportMapper::Struct


SalesReport.load(
  Book.joins(:author).last_ordered_before(30.days.ago)
)
```

github.com/boblail/pluck_map

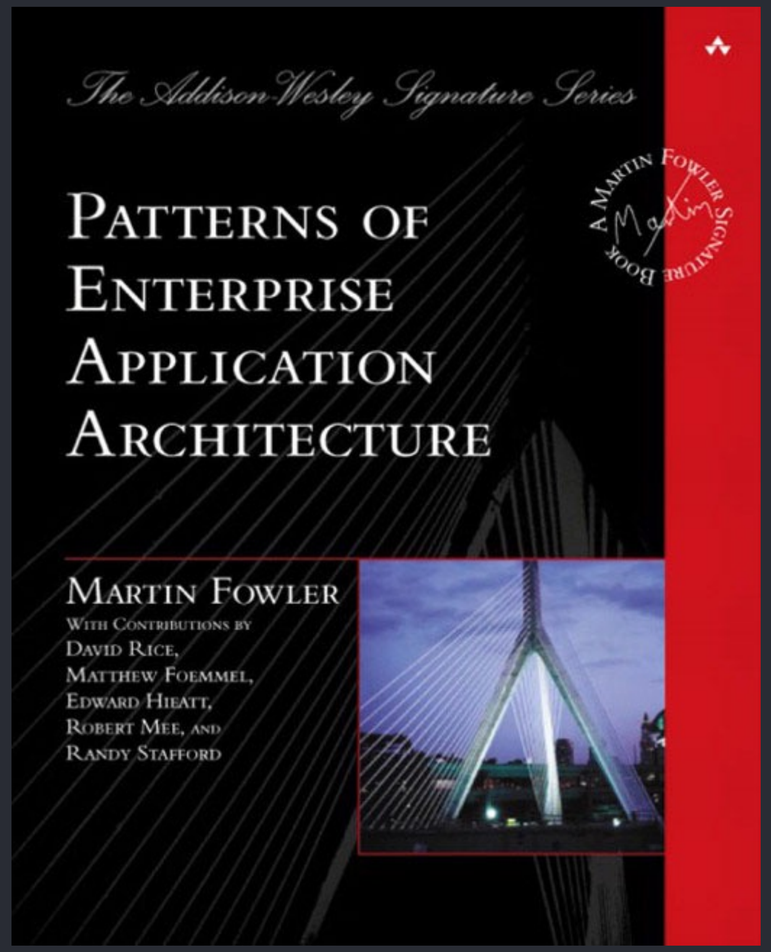
```ruby
# Author, title and most recent purchase price
# of books that weren't bought in the last 30 days
SalesReportMapper = PluckMap[Book].define do
  author select: "authors.name"
  title
  purchase_price select: "orders.price"
end


class SalesReport < SalesReportMapper::Struct
  include ActionView::Helpers::NumberHelper

  def purchase_price
    number_to_currency(super)
  end
end


SalesReport.load(
  Book.joins(:author).last_ordered_before(30.days.ago)
)
```

SEPARATING RESPONSIBILITIES

	Author Creator	Author
Every author needs a biography (even if it's empty)	✓	✓
We can't save an author without knowing their name	✓	✓
What is an author's current age?	✗	✓
What's an author's address? (for a mailing label)	✗	✓
How much has an author been paid in royalties?	✗	✓



An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.



An object that wraps a row in a database table or view and encapsulates the database access~~, and adds domain logic on that data~~.

BENEFITS OF EMPTY ACTIVE RECORDS

I. Isolate and name responsibilities

II. Unit Tests decoupled from the database

III. Freedom to operate on sets

BENEFITS OF EMPTY ACTIVE RECORDS

I. Scalability

II. Maintainability

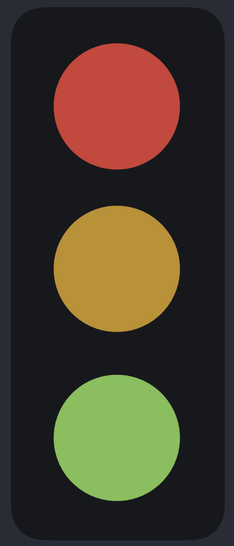
CHECK OUT

I. *7 Patterns to Refactor Fat ActiveRecord Models* (Rezvina, 2012)
bit.ly/rezvina

II. *7 Design Patterns to Refactor MVC Components in Rails* (Kotsurenko, 2016)
bit.ly/kotsurenko

Active Record

INVESTMENT ADVICE



Callbacks

Validations

Query Interface

AVOIDING CALLBACKS

```ruby
class Author < ActiveRecord::Base
  after_create :ensure_author_has_a_biography!

  def ensure_author_has_a_biography!
    Biography.create!(author: self)
  end
end

class Biography < ActiveRecord::Base
  belongs_to :author
end

Author.create!(name: "Alan Paton")
# BEGIN
# INSERT INTO authors (name) VALUES ('Alan Paton')
# INSERT INTO biographies (author_id) VALUES (2)
# COMMIT
```

```ruby
class Author < ActiveRecord::Base; end
class Biography < ActiveRecord::Base; end

class CreateAuthor
  def self.call(params)
    Author.transaction do
      author = Author.create!(params)
      Biography.create!(author: author)
      author
    end
  end
end

CreateAuthor.call(name: "Alan Paton")
# BEGIN
# INSERT INTO authors (name) VALUES ('Alan Paton')
# INSERT INTO biographies (author_id) VALUES (2)
# COMMIT
```

```ruby
class Author < ActiveRecord::Base; end
class Biography < ActiveRecord::Base; end

class EnsureAuthorsHaveBiographies < ActiveRecord::Migration[5.2]
  def up
    execute <<~SQL
    CREATE TRIGGER create_biography_for_author_trigger AFTER INSERT ON authors
      FOR EACH ROW
      BEGIN
        INSERT INTO biographies SET author_id = NEW.id;
      END;
    SQL
  end
end

Author.insert_all [{ name: "Alan Paton" }]
# INSERT INTO authors (name) VALUES ('Alan Paton')
#   -- INSERT INTO biographies (author_id) VALUES (2)
```

VALIDATIONS

```ruby
class Book < ActiveRecord::Base
  validates :publication_date, presence: true
end
```

```ruby
class CreateBook
  include ActiveModel::Model, Virtus

  attribute :title, String
  attribute :publication_date, Date

  validates :title, :publication_date, presence: true

  def self.call(params)
    new(params).save
  end

  def save
    return false unless valid?
    Book.create!(title: title, publication_date: publication_date)
  end
end

CreateBook.call(params)
```

```ruby
class Book < ActiveRecord::Base
  validates :isbn, uniqueness: true
end

Book.create!(isbn: "978-0449213445", title: "The Chosen")
Book.create!(isbn: "978-0449213445", title: "NOPE")
# SELECT 1 FROM books WHERE isbn = '978-0449213445'
# => ActiveRecord::RecordInvalid


class AddUniqueIndexOnBooksIsbn < ActiveRecord::Migration[5.2]
  def change
    add_index :books, :isbn, unique: true
  end
end

Book.create!(isbn: "978-0449213445", title: "The Chosen")
Book.create!(isbn: "978-0449213445", title: "NOPE")
# INSERT INTO books (isbn, title) VALUES ('978-0449213445', 'NOPE')
# => ActiveRecord::RecordNotUnique
```

QUERY INTERFACE

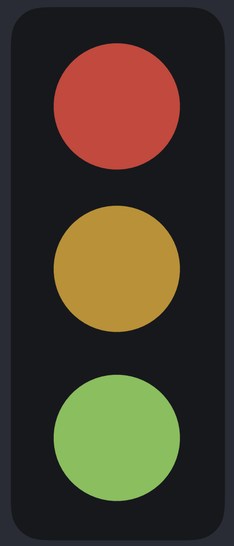
```ruby
class Book < ActiveRecord::Base
  default_scope { order(title: :asc) }

  def self.paperbacks
    where(cover_type: "paperback")
  end

  def self.published(date_range)
    where(publication_date: date_range)
  end
end

Book.paperbacks.published(Date.new(1967, 1, 1)..Date.new(1977, 12, 31))
# SELECT * FROM books
#   WHERE cover_type = 'paperback'
#   AND publication_date BETWEEN '1967-01-01' AND '1977-12-31'
#   ORDER BY title ASC
```

```ruby
class Book < ActiveRecord::Base
  belongs_to :publisher

  def self.published_by(publisher_name)
    joins(:publisher).merge(Publisher.where(name: publisher_name))
  end
end

Book.published_by("Simon & Schuster")
# SELECT * FROM books
# INNER JOIN publishers ON books.publisher_id = publishers.id
# WHERE publishers.name = 'Simon & Schuster'
```

```ruby
class Author < ActiveRecord::Base
  has_many :books

  def self.published_by(publisher_name)
    where(id: Book.published_by(published_by).select(:author_id))
  end
end

Author.published_by("Simon & Schuster")
# SELECT * FROM authors WHERE id IN (
#   SELECT author_id FROM books
#   INNER JOIN publishers ON books.publisher_id = publishers.id
#   WHERE publishers.name = 'Simon & Schuster'
# )
```

```ruby
Book.arel_table # => Arel::Table

class Book < ActiveRecord::Base
  def self.published_after(date)
    where(arel_table[:publication_date].gt(date))
  end
end

Book.published_after(Date.new(2000, 1, 1))
# SELECT * FROM books WHERE publication_date > '2000-01-01'
```

```ruby
class Book < ActiveRecord::Base
  def self.published_after(date)
    where(arel_table[:publication_date].gt(date))
  end

  def self.published_posthumously
    joins(:author).published_after(Author.arel_table[:death_date])
  end
end

Book.published_posthumously
# SELECT * FROM books
# INNER JOIN authors ON books.author_id = authors.id
# WHERE books.publication_date > authors.death_date
```



Callbacks

Validations

Query Interface

Thank You