Backend-agnostic document indexing & retrieval

Iasonas Gavriilidis
Kyriakos Kentzoglanakis

pamediakopes.gr engineering

Motivation

- We model cases as sequences of individual tasks
- Users need to search tasks
  - operational
  - reporting
- Tasks “indexed” to a dedicated table in our DB
  - lots of JOINs
  - bad performance
  - hard to maintain and expand
  - WET (as opposed to DRY)

Requirements

- Remove task “indexing” from our main DB
- Solution should be simple to maintain/extend
- Info on what/how gets indexed should be defined in one place
- Solution should work independent of backend storage choice
    - elasticsearch
    - solr
    - noSQL
    - RDBMS (???)
- Adding a new backend should be easy

The solution

Indexable

A Ruby module that offers a DSL to define information relevant to indexing

- Specify **what** to index (field names/values, custom attributes)
    - Supports nested documents
- Specify **when** to index (notifications upon “*change*”)

The solution

Queryable

A Ruby module that offers a DSL to perform queries against indexed documents

- Create complex queries
- Control execution
- Uniform way of getting back the results

Indexable :: What to index

```
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at
    string :type, value: proc { task_category.name }
  end
end
```

Indexable :: Nested Documents

```ruby
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at
    string :type, value: proc { task_category.name }
    struct :case_entity
  end
end
```

```ruby
class CaseEntity < ActiveRecord::Base
  include Indexable
  has_many :tasks, class_name: 'Task'

  define_index_fields(owner: Task) do
    integer :id
    date :created_at
    text_array :notes, value: proc { notes.map(&:body) }
  end
end
```

Indexable :: When to index

```ruby
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at
    string :type, value: proc { task_category.name }
    struct :case_entity
  end

  define_index_notifier { self }
end
```

```ruby
class CaseEntity < ActiveRecord::Base
  include Indexable
  has_many :tasks, class_name: 'Task'

  define_index_fields(owner: Task) do
    integer :id
    date :created_at
    text_array :notes, value: proc { notes.map(&:body) }
  end

  define_index_notifier(target: Task) { tasks }
end
```

Indexable :: Schema

```ruby
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at
    string :type, value: proc { task_category.name }
    struct :case_entity
  end

  define_index_notifier { self }
end
```

```ruby
class CaseEntity < ActiveRecord::Base
  include Indexable
  has_many :tasks, class_name: 'Task'

  define_index_fields(owner: Task) do
    integer :id
    date :created_at
    text_array :notes, value: proc { notes.map(&:body) }
  end

  define_index_notifier(target: Task) { tasks }
end
```

Indexable :: Schema

```ruby
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at
    string :type, value: proc { task_category.name }
    struct :case_entity, from: CaseEntity
  end

  define_index_notifier { self }
end
```

supports polymorphic relationships

```ruby
class CaseEntity < ActiveRecord::Base
  include Indexable
  has_many :tasks, class_name: 'Task'

  define_index_fields(owner: Task) do
    integer :id
    date :created_at
    text_array :notes, value: proc { notes.map(&:body) }
  end

  define_index_notifier(target: Task) { tasks }
end
```

Indexable :: Schema

```
> Task.schema
{
  "id" => :integer,
  "last_assigned_at" => :date,
  "type" => :string,
  "case_entity" => {
    "id" => :integer,
    "created_at" => :date,
    "notes" => :text_array
  }
}
```

Indexable :: Custom Schema Properties

```ruby
class Task < ActiveRecord::Base
  include Indexable
  belongs_to :case_entity, class_name: 'CaseEntity'

  define_index_fields do
    integer :id
    date :last_assigned_at, value: :assigned_at,
         is_column: true, label: 'Last Assigned At'
    string :type, value: proc { task_category.name },
           is_column: true, label: 'Task Type'
    struct :case_entity, from: CaseEntity
  end

  define_index_notifier { self }
end
```

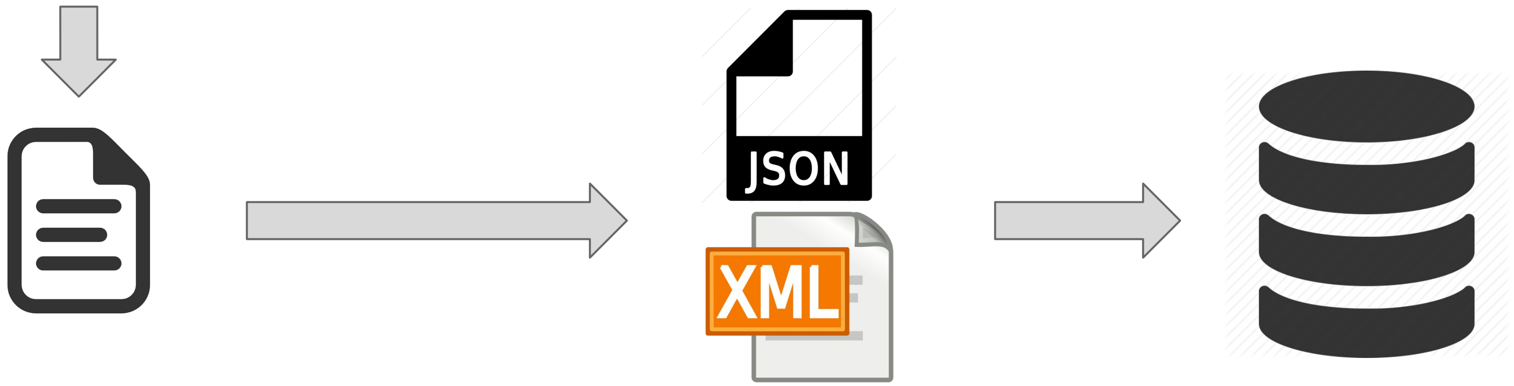
```ruby
class CaseEntity < ActiveRecord::Base
  include Indexable
  has_many :tasks, class_name: 'Task'

  define_index_fields(owner: Task) do
    integer :id
    date :created_at, is_column: true, label: 'Case
Created At'
    text_array :notes, value: proc { notes.map(&:body) }
  end

  define_index_notifier(target: Task) { tasks }
end
```

Indexable :: Schema

```
> Task.schema {|field_type| field_type.get_option(:is_column)}
{
  "id" => nil,
  "last_assigned_at" => true,
  "type" => true,
  "case_entity" => {
    "id" => nil,
    "created_at" => true,
    "notes" => nil
  }
}
```

Indexable :: Document Indexing

```
> task.generate_document
{
  "id" => 12,
  "last_assigned_at" => "2015-10-04T23:11:01Z",
  "type" => "ContactCustomerTask",
  "case_entity" => {
    "id" => 2,
    "created_at" => "2015-10-04T11:11:01Z",
    "notes" => ["this is note A", "Hello there"]
  }
}
```

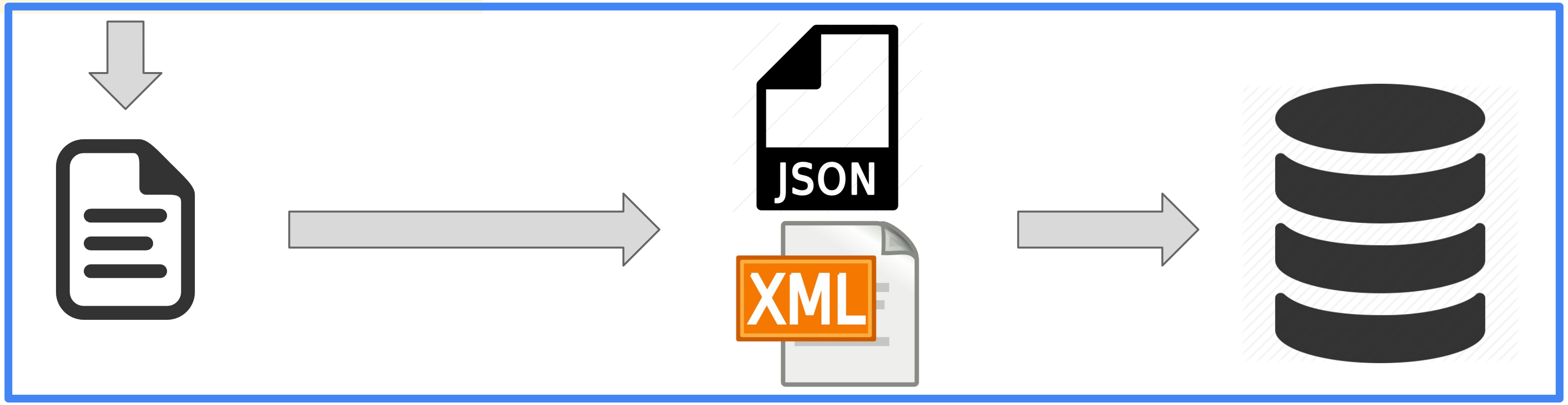


Indexable :: Document Indexing

```
> task.generate_document
{
  "id" => 12,
  "last_assigned_at" => "2015-10-04T23:11:01Z",
  "type" => "ContactCustomerTask",
  "case_entity" => {
    "id" => 2,
    "created_at" => "2015-10-04T11:11:01Z",
    "notes" => ["this is note A", "Hello there"]
  }
}
```

- General transformations
- Backend-specific transformations

Indexer

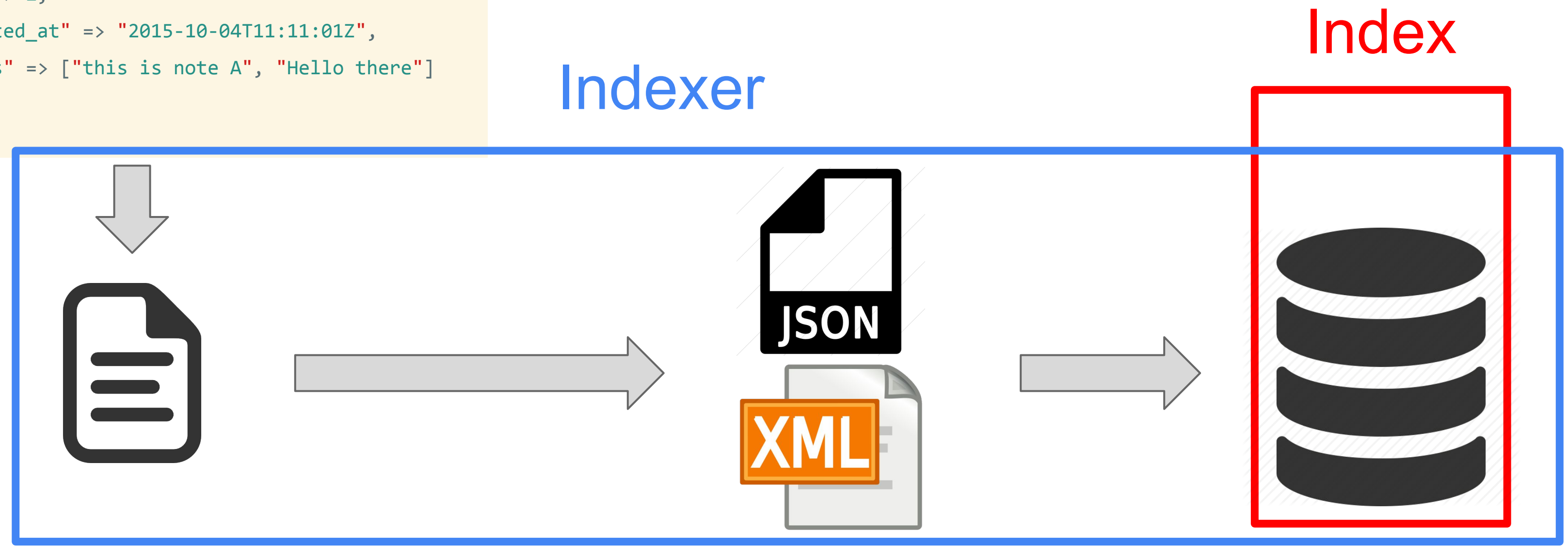


Indexable :: Document Indexing

```
> task.generate_document
{
  "id" => 12,
  "last_assigned_at" => "2015-10-04T23:11:01Z",
  "type" => "ContactCustomerTask",
  "case_entity" => {
    "id" => 2,
    "created_at" => "2015-10-04T11:11:01Z",
    "notes" => ["this is note A", "Hello there"]
  }
}
```

- General transformations
- Backend-specific transformations

- Communication with backend
- Configuration (credentials etc.)



Queryable :: Query Creation :: Part 1

interface

```
q_b = IndexableClass.query_builder
c_b = q_b.criteria_builder

less     = c_b.lt('a_date, '2015-11-11T00:00:00Z')
equal    = c_b.eq('a_nested_model.a_string', 'foo')
criteria = c_b.or(less, equal)

q_b.where(criteria)
q_b.order('an_integer', :asc)
q_b.limit(20)

query = q_b.build
```

- indexing store used
- connection & authentication information
- schema

- Query Builder

creates & internally stores query expressions

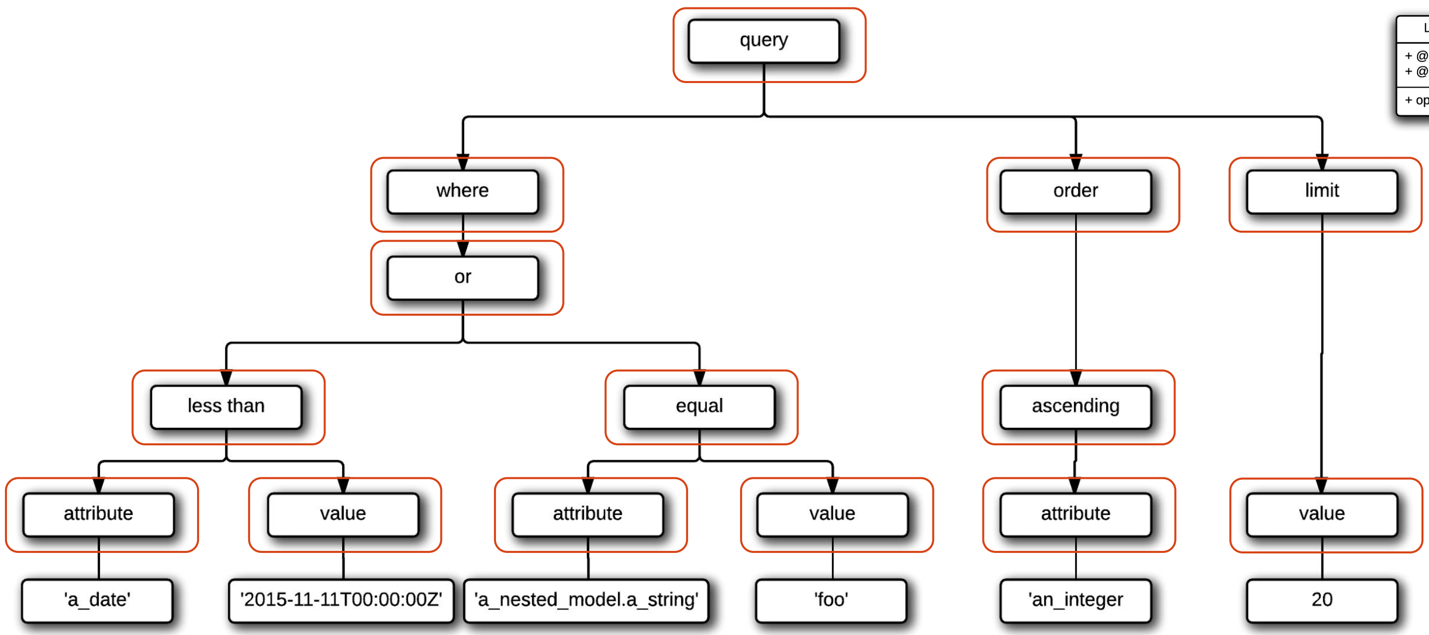
- Criteria Builder

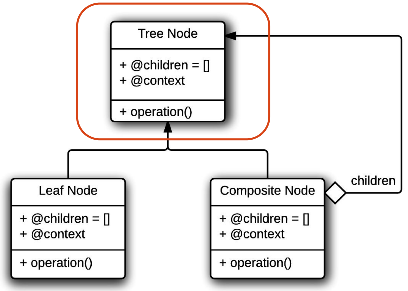
creates & logically combines multiple criteria

Queryable :: Query Creation :: Part 2

internals

```
query = q_b.build
```





Queryable :: Query Evaluation

the visitor pattern

- Related behavior isn't spread over the classes defining the object structure; it's localized in a visitor
- Visitor makes adding new operations easy
- Accumulating state

- Relies on double dispatch. Implementation is chosen based on:
  - Visitor class
  - Element/Node class

```
class TreeNode
  #...#
  def accept(visitor)
    visitor.visit(self)
  end
end
```

```
class Visitor
  def visit(tree_node)
    method_name = "visit_#{tree_node.class.name.lowercase}"
    send(method_name, tree_node)
  end
end
```

Queryable :: Query Evaluation

the visitor pattern in ruby

```ruby
class MyImaginaryStoreEvaluator < Visitor
  def visit_equal(tree_node)
   "#{ tree_node.attribute.accept(self)} = #{subject.value.accept(self) }"
  end
  def visit_and(tree_node)
   "#{ tree_node.children.map { |child| child.accept(self) }.join('and') }"
  end
end
```

```ruby
def visit_value(tree_node)
  attribute_type =
    tree_node.associated_attribute.type
  case attribute_type
    when :integer
      "#{ tree_node.value }"
    when :string
      "'#{ tree_node.value }'"
    #...#
  end
end
```

```ruby
class Equal < TreeNode
```

```ruby
class And < TreeNode
```

```ruby
class Value < TreeNode
```

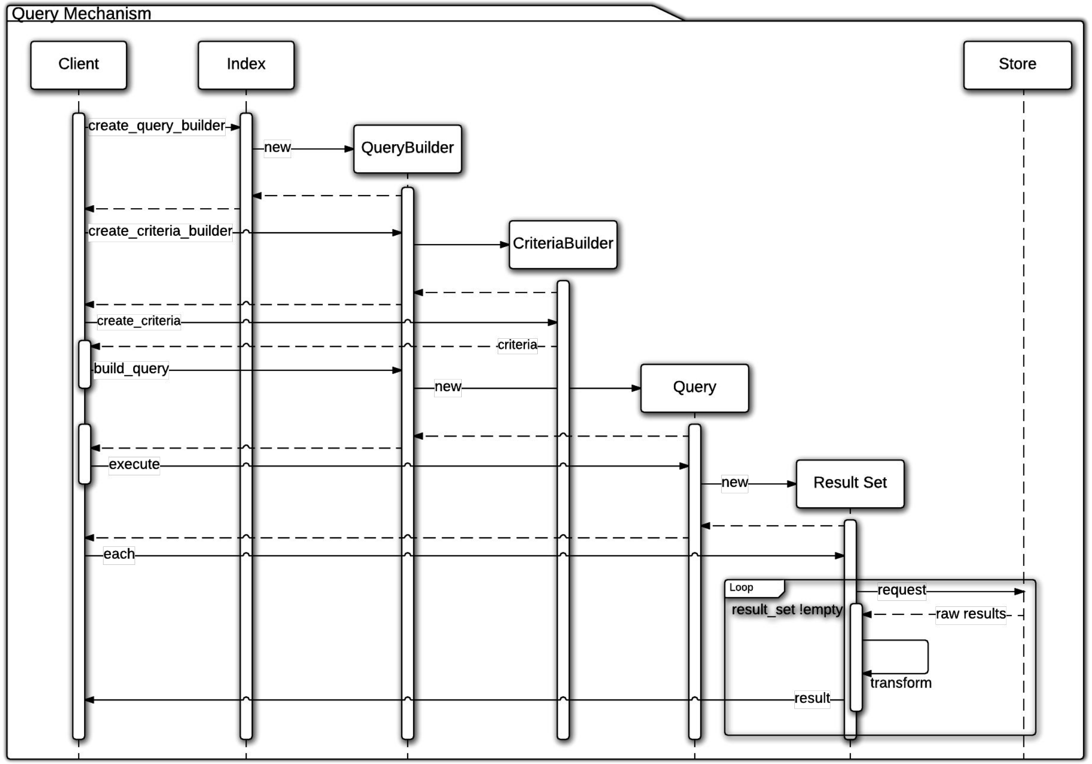
```ruby
class QueryValidator < Visitor
end
```

Queryable :: Query Execution

- Responsibility of query execution in the Executor class
  - Validates the Query
  - Evaluates the Query
  - Executes the Query (or not)

- Controlling execution
  - Create and return a ResultSet object
  - Iterate and execute narrowed queries
  - Transform raw results to schema like documents

Queryable :: Overview



Current Status

- Currently in production alpha (i.e. rolled out but not visible to users)
- Around 3.5m tasks
- 102 fields per document spread across 12 models
- Backend currently is Cloudsearch (AWS)
    - Solr custom fork with lots of minor nuisances
    - AWS recently announced an elasticsearch managed service

well DONE

BUT

Where’s my gem?

Thank you

Any questions?

pamediakopes.gr
engineering

Check us out at
http://engineering.pamediakopes.gr