criteria

The following are a list of chainable criteria query methods in Mongoid along with their corresponding database selectors or options.

Please note again that criteria are lazy evaluated, and with each chained method it will be cloned and return a new criteria copy.

For more information on the underlying database selectors please refer to MongoDB directly.

query methods

- Model.all_in
- Model.all_of
- Model.also_in
- Criteria#and
- Model.any_in
- Model.any_of
- Model.asc
- Model.desc
- Criteria#distinct
- Model.excludes
- Model.includes
- Model.limit
- Model.near
- Model.not_in
- Model.only
- Model.order_by
- Model.skip
- Model.where
- Model.without

Model.all_in | Criteria#all_in

Adds a criterion that specifies values that must all match in order to return results. The corresponding MongoDB operation is `$all`.

mongoid

```
# Match all people with Bond and 007 as aliases.
Person.all_in(aliases: [ "Bond", "007" ])
```

mongodb query selector

```
{ "aliases" : { "$all" : [ "Bond", "007" ] }}
```

Model.all_of | Criteria#all_of

Adds a criterion that specifies expressions that must all match in order to return results. The corresponding MongoDB operation is `$and`.

mongoid

```
# Match all crazy old people.
Person.all_of(:age.gt => 60, mental_state: "crazy mofos")
```

mongodb query selector

```
{ "$and" : [{ "age" : { "$gt" : 60 }}, { "mental_state" : "crazy mofos" }] }
```

`Model.also_in` | `Criteria#also_in`

Adds a criterion that specifies values where any value can be matched in order to return results. This is similar to `Criteria#any_in` with the exception here that if if it chained with values for the same field it performs a union of the values where `any_in` perform an intersection. The underlying MongoDB expression is `$in`.

mongoid

```
# Match all people with either Bond or 007 as aliases.
Person.also_in(aliases: [ "Bond", "007" ])
Person.any_in(aliases: [ "Bond" ]).also_in(aliases: [ "007" ])
```

mongodb query selector

```
{ "aliases" : { "$in" : [ "Bond", "007" ] }}
```

`Criteria#and`

Adds another simple expression that must match in order to return results. This is the same as Criteria#where and is mostly here for syntactic sugar.

mongoid

```
# Match all people with last name Jordan and first name starting with d.
Person.where(last_name: "Jordan").and(first_name: /^d/i)
```

mongodb query selector

```
{ "last_name" : "Jordan", "first_name" : /^d/i }
```

`Model.any_in` | `Criteria#any_in`

Adds a criterion that specifies values where any value can be matched in order to return results. This is similar to `Criteria#also_in` with the exception here that if if it chained with values for the same field it performs an intersection of the values where `also_in` perform a union. The underlying MongoDB expression is `$in`.

mongoid

```
# Match all people with either Bond or 007 as aliases.
Person.any_in(aliases: [ "Bond", "007" ])
Person.
  any_in(aliases: [ "Bond", "007", "James" ]).
  any_in(aliases: [ "Bond", "007" ])
```

mongodb query selector

```
{ "aliases" : { "$in" : [ "Bond", "007" ] }}
```

Model.any_of | Criteria#any_of

Adds a criterion that specifies a set of expressions that any can match in order to return results. The underlying MongoDB expression is `$or`.

mongoid

```
# Match all people with either last name Penn or Teller
Person.any_of({ last_name: "Penn" }, { last_name: "Teller" })
```

mongodb query selector

```
{ "last_name" :
  { "$or" :
    [ { "last_name" : "Penn" }, { "last_name" : "Teller" } ]
  }
}
```

Model.asc | Criteria#asc

Adds ascending sort options for the provided fields.

mongoid

```
# Sort people by first and last name ascending.
Person.asc(:first_name, :last_name)
Person.ascending(:first_name, :last_name)
```

mongodb query options

```
{ "sort" :
    {[ [ "first_name", Mongo::ASCENDING ],
      [ "last_name", Mongo::ASCENDING ] ]} }
```

Model.desc | Criteria#desc

Adds descending sort options for the provided fields.

mongoid

```
# Sort people by first and last name descending.
Person.desc(:first_name, :last_name)
Person.descending(:first_name, :last_name)
```

mongodb query options

```
{ "sort" :
    {[ [ "first_name", Mongo::DESCENDING ],
      [ "last_name", Mongo::DESCENDING ] ]} }
```

Criteria#distinct(name)

Get the distinct values for the provided field.

mongoid

```
# Get the distinct values for last names
Person.all.distinct(:last_name)
```

mongodb query options

```
{ "distinct" : "last_name" }
```

Model.excludes | Criteria#excludes

Adds a criterion that specifies a set of expressions that cannot match in order to return results. The underlying MongoDB expression is `$ne`.

mongoid

```
# Match all people without either last name Teller and first name Bob.
Person.excludes(last_name: "Teller", first_name: "Bob")
```

mongodb query selector

```
{{ "last_name" : { "$ne" : "Teller" } }, { "first_name" : { "$ne" : "Bob" } }}
```

Model.includes | Criteria#includes

Adds a criterion that specifies a list of relational associations to eager load when executing the query. This is to prevent the n+1 issue when iterating over documents that access their relations during the iteration.

This only works with `has_many`, `has_one`, and `belongs_to` relations and only 1 level deep at the current moment. If you try to eager load a many to many an exception will get raised. Many to many is not supported due to the performance actually being slower despite lowering the number of database calls.

mongoid

```
# Eager load the posts and games when retrieving the people.
Person.includes(:posts, :game)
```

mongodb queries

```
people_ids = people.find({}, { "fields" : { "_id" : 1 }})
posts.find({ "person_id" : { "$in" : people_ids }})
games.find({ "person_id" : { "$in" : people_ids }})
```

In order for `#includes` to work the Mongoid identity map must be enabled in the mongoid.yml:

```
identity_map_enabled: true
```

Model.limit | Criteria#limit

Limits the number of returned results by the provided value.

mongoid

```
# Only return 20 documents.
Person.limit(20)
```

mongodb query options

```
{ "limit" : 20 }
```

Model.near | Criteria#near

Adds a criterion to find locations that are near the supplied coordinates. This performs a MongoDB `$near` selection and requires a 2d index to be on the provided field.

mongoid

```
# Match all bars near Berlin
Bar.near(location: [ 52.30, 13.25 ])
```

mongodb query selector

```
{ "location" : { "$near" : [ 52.30, 13.25 ] }}
```

Model.not_in | Criteria#not_in

Adds a criterion that specifies a set of expressions that cannot match in order to return results. The underlying MongoDB expression is `$nin`.

mongoid

```
# Match all people without last names Zorg and Dallas
Person.not_in(last_name: [ "Zorg", "Dallas" ])
```

mongodb query selector

```
{{ "last_name" : { "$nin" : [ "Zorg", "Dallas" ] } }}
```

Model.only | Criteria#only

Limits the fields returned from the database to those supplied to the method. Extremely useful

for list views where the entire documents are not needed. Cannot be used in conjunction with #without.

mongoid

```
# Return only the first and last names of each person.
Person.only(:first_name, :last_name)
```

mongodb query options

```
{ "fields" : { "first_name" : 1, "last_name" : 1 }}
```

Model.order_by | Criteria#order_by

Sorts the results given the arguments that must match the MongoDB driver sorting syntax (key/value pairs of field and direction).

mongoid

```
# Provide the sorting options.
Person.order_by([[:first_name, :asc], [:last_name, :desc]])
```

mongodb query options

```
{ "sort" :
    {[ [ "first_name", Mongo::ASCENDING ],
      [ "last_name", Mongo::DESCENDING ] ]} }
```

Model.skip | Criteria#skip

Skips the number of the results given the provided value, similar to a SQL "offset".

mongoid

```
# Skip 20 documents.
Person.skip(20)
```

mongodb query options

```
{ "skip" : 20 }
```

Model.where | Criteria#where

Adds a criterion that must match in order to return results. If provided a string it interperets it as a javascript function and converts it to the proper `$where` clause. Mongoid also provides convenience h4s on `Symbol` to make advanced queries simpler to write.

mongoid

```
# Match all people with first name Emmanuel
```

```
Person.where(first_name: "Emmanuel")

# Match all people with first name Emmanuel using Javascript.
Person.where("this.first_name == 'Emmanuel'")

# Match all people who live in Berlin, where address is embedded.
Person.where("addresses.city" => "Berlin")

# Example queries using symbol h4s to perform more complex criteria.
Person.where(:title.all => ["Sir"])
Person.where(:age.exists => true)
Person.where(:age.gt => 18)
Person.where(:age.gte => 18)
Person.where(:title.in => ["Sir", "Madam"])
Person.where(:age.lt => 55)
Person.where(:age.lte => 55)
Person.where(:title.ne => "Mr")
Person.where(:title.nin => ["Esquire"])
Person.where(:aliases.size => 2)
Person.where(:location.near => [ 22.50, -21.33 ])
Person.where(:location.within => { "$center" => [ [ 50, -40 ], 1 ] })
```

mongodb query selectors

```
# Match all people with first name Emmanuel
{ "first_name" : "Emmanuel" }

# Match all people with first name Emmanuel using Javascript.
{ "$where" : "this.first_name == 'Emmanuel'" }

# Match all people who live in Berlin, where address is embedded.
{ "addresses.city" : "Berlin" }

# Example queries using symbol h4s to perform more complex criteria.
{ "title" : { "$all" : [ "Sir" ] } }
{ "age" : { "$exists" : true } }
{ "age" : { "$gt" : 18 } }
{ "age" : { "$gte" : 18 } }
{ "title" : { "$in" : [ "Sir", "Madam" ] } }
{ "age" : { "$lt" : 55 } }
{ "age" : { "$lte" : 55 } }
{ "title" : { "$ne" : "Mr" } }
{ "title" : { "$nin" : [ "Esquire" ] } }
{ "aliases" : { "$size" : 2 } }
{ "location" : { "$near" : [ 22.50, -21.33 ] } }
{ "location" : { "$within" : { "$center" => [ [ 50, -40 ], 1 ] } } }
```

`Model.without` | `Criteria#without`

Limits the fields returned from the database to those NOT supplied to the method. Extremely useful for list views where the entire documents are not needed. Cannot be used in conjunction with #only.

mongoid

```
# Return all fields except first name and last name
Person.without(:first_name, :last_name)
```

mongodb query options

```
{ "fields" : { "first_name" : 0, "last_name" : 0 }}
```

Mongoid, 2009-2011, written and maintained by Durran Jordan.