# Module 2, Lesson 1: Aggregation Framewework

The overall goal of the assignment is to:

- implement aggregation framework queries using the MongoDB Ruby Driver

The functional goal of the assignment is to:

- implement various document access methods for Race Results

Note that this assignment was written so that you can implement it in parts after each lecture. If you are performing the assignment in between lectures, stop at the next lecture boundary in the technical requirements section and resume once you have completed the lecture. You are free to experiment with other forms of the queries presented, but the grading will only be targeted at the specific requirements listed.

## Getting Started

1. Start your MongoDB server
2. Download and extract the starter set of files. The root directory of this starter set will be referred to as the root directory of your solution.

```
--- student-start
    |-- assignment.rb
    |-- race_results.json
    |-- .rspec (important hidden file)
    `-- spec
        |-- assignment_spec.rb
        `-- spec_helper.rb
```

   - assignment.rb - your solution must be placed within this file
   - spec - this directory contains tests to verify your solution. You should not modify anything in this directory

3. Install the following gems. You may already have them installed.

```
$ gem install rspec
$ gem install rspec-its
$ gem install mongo -v 2.1.2
```

4. Run the rspec command from the project root directory (i.e., `student-start` directory) to execute the unit tests within the spec directory. This should result in several failures until you complete your solution in `assignment.rb`.

```
$ rspec

(N) examples, (N) failures
...
```

5. Implement the Ruby technical requirements in `assignment.rb` within the provided class `Solution`. Helper methods have been provided to get a connection to Mongo and set the database and collection names. You can override these values using environment variables if you are not using the defaults.

- MONGO_URL='mongodb://localhost:27017'
- MONGO_DATABASE='test'
- RACE_COLLECTION='race1'

Implement all methods relative to the `@coll` instance variable setup to reference the collection.

## Technical Requirements

### Lecture 1: Introduction

In this section will use a few pre-canned queries to get warmed up on the syntax of the aggregation query. Follow-on sections will require more work on your part.

1. Load the `assignment.rb` script into the `irb` shell, verify you can get access to the collection, and assign that collection to the variable called `racers` (used in follow-on steps).

```
$ irb
> require './assignment.rb'
> racers=Solution.collection
 => #<Mongo::Collection:0x22344800 namespace=test.race1>
```

2. Issue a simple aggregation query that counts the number of race results in the collection.

```
> racers.find.aggregate([ {:$group=>{:_id=>0, :count=>{:$sum=>1}}}]).first
 => {"_id"=>0, "count"=>1000}
```

To break this query down

```
> racers.find.aggregate([
      {:$group=>{
          :_id=>0,
          :count=>{:$sum=>1}}
      }
  ]).first
 => {"_id"=>0, "count"=>1000}
```

- `aggregate()` takes an array of commands, `$group` is one of them. We can have many aggregate commands in that array and `$group` can occur multiple times. The order within the array is important.
- `$group` has two primary arguments: group by key, and group functions. The function results are associated with each resulting key. In this example, we are making the key be a single value – so all functions results are applied against every row. We have a single group.
- `_id` is assigned to our group key. Here it is a fixed `0` value to make every document processed a member of the same group.
- `count` is a property name we want in the results. We set it to the result of counting 1 for each document.
- `$sum` is an aggregate function that adds a number (`1` in this case) for each document it processes.
- `first()` is a function that returns only the first document from the result of `aggregate()`. Since we know we will have only a single result row with the count result, we can simply return the first (and only) result.

3. Issue a slightly more complex aggregation query that counts the number of race results by group. Please pardon the overload of the age-group `'$group'` (e.g., "14 and under") with the MongoDB aggregation function `:$group`.

```
> racers.find.aggregate([ {:$group=>{:_id=>"$group", :count=>{:$sum=>1}}}]).each {|r| pp r}
{"_id"=>"14 and under", "count"=>111}
{"_id"=>"40 to 49", "count"=>141}
{"_id"=>"20 to 20", "count"=>123}
{"_id"=>"60 to 69", "count"=>121}
{"_id"=>"50 to 59", "count"=>129}
{"_id"=>"masters", "count"=>117}
{"_id"=>"30 to 39", "count"=>127}
{"_id"=>"15 to 19", "count"=>131}
```

To again break the query down

- `aggregate` and (the aggregration function-)`$group` are being used exactly as they were before except we have changed the `$group` specification to use the name of the (age-)`group` as the key. As you saw above – this results in several documents broken down by distinct (age-)`group` name assigned to the `_id` of the document.
- `count` and `$sum` work the same as they did before except they have more (aggregate function-)`groups` to work with.
- `each` iterates through each document in the collection. Since we may have multiple documents `first` is of little value to us except to grab a sample.
- `{|r| pp r}` is a single line block where each document of the result is passed in as `r` and pretty print (`pp`) is used to print a more human readable form of document hashes.

```
> racers.find.aggregate([ {
      :$group=>{
          :_id=>"$group",
          :count=>{:$sum=>1}}
          }
          ]).each {|r| pp r}
```

If you are familiar with SQL, the above query would be similar to the following. Again, pardon our use of the term (age-)`group` versus a SQL-`group by`.

```
select 'group', count('_id')
from RACERS
group by 'group'
```

**Lecture 2: `$project`**

In this section you will be asked to reshape a document by promoting and building new properties required by downstream aggregation functions and in the final result. You will place all solutions within `assignment.rb`. Refer back to the lectures for the details of each query.

1. Implement an instance method called `racer_names` that
   - accepts no inputs
   - finds all racers
   - reduces the result to contain only `first_name` and `last_name` (Hint: $project)
   - returns the Mongo result object for the aggregation command

   You can try out your new method using the irb shell.

   ```
   > s=Solution.new
   > r=s.racer_names.to_a.slice(0,2)
     => [{"first_name"=>"SHAUN", "last_name"=>"JOHNSON"},
         {"first_name"=>"TUAN", "last_name"=>"JOHNSON"}]

   $rspec spec/lecture2_spec.rb -e rq01
   ```

2. Implement an instance method called `id_number_map` that
   - accepts no inputs
   - finds all racers
   - reduces the result to contain only `_id` and `number`
   - returns the Mongo result object for the aggregation command

   You can try out your new method using the irb shell.

   ```
   > r=s.id_number_map.to_a.slice(0,2)
   [{"_id"=>BSON::ObjectId('563e7555e301d0b356000000'), "number"=>0},
    {"_id"=>BSON::ObjectId('563e7555e301d0b356000001'), "number"=>1}]

   $rspec spec/lecture2_spec.rb -e rq02
   ```

3. Implement an instance method called `concat_names` that

   - accepts no inputs
   - finds all racers
   - reduces the result to contain only `number` and `name` field only where `name` is the result of concatenating `last_name, first_name` (Hint: $concat)
   - returns the Mongo result object for the aggregation command

   You can try out your new method using the irb shell.

```
> r=s.concat_names.to_a.slice(0,2)
 => [{"number"=>0, "name"=>"JOHNSON, SHAUN"}, {"number"=>1, "name"=>"JOHNSON, TUAN"}]
```

```
$rspec spec/lecture2_spec.rb -e rq03
```

**Lecture 3: $group**

In this section we will get some practice applying group functions around a selected set of sub-results.

1. Implement an instance method called `group_times` that

   - accepts no inputs
   - finds all racers
   - groups the racers into gender and age group (Hint: $group)
   - counts the number of racers in the group and assigns this to `runners`
   - calculates the fastest time for each group and assigns this value to `fastest_time` (Hint: $min)
   - returns the Mongo result object for the aggregation command

   You can try out your new method using the irb shell.

```
> r=s.group_times.to_a.slice(0,3)
 => [{"_id"=>{"age"=>"50 to 59", "gender"=>"F"}, "runners"=>65, "fastest_time"=>1269},
     {"_id"=>{"age"=>"30 to 39", "gender"=>"M"}, "runners"=>68, "fastest_time"=>1262},
     {"_id"=>{"age"=>"14 and under", "gender"=>"M"}, "runners"=>66, "fastest_time"=>1363}]
```

```
$rspec spec/lecture3_spec.rb -e rq01
```

2. Implement an instance method called `group_last_names` that

   - accepts no inputs
   - finds all racers
   - groups the racers into gender and (age-)group as above
   - creates an array[] of (**non-distinct**) last_names called `last_names` (Hint: $push)
   - returns the Mongo result object for the aggregation command

   You can try out your new method using the irb shell. Note the first group and the names within the group may not be in the same order as the example shows.

```
> r=s.group_last_names.first
{"_id"=>{"age"=>"50 to 59", "gender"=>"F"}, "last_names"=>["GARNER", "SINGH", ...]}
> r=s.group_last_names.first[:last_names].count
 => 65
```

```
$rspec spec/lecture3_spec.rb -e rq02
```

3. Implement an instance method called `group_last_names_set` that repeats the previous query except

   - creates an array[] of (**distinct**) last_names called `last_names` (Hint: $addToSet)

Note that because of the size of the array and the fact the contents are unsorted, it is hard to visually spot the duplicates, but `$addToSet` will de-dup the collection and `$push` will collect all members.

You can try out your new method using the irb shell.

```
> r=s.group_last_names_set.first
{"_id"=>{"age"=>"50 to 59", "gender"=>"F"}, "last_names"=>["GARNER", "SINGH", ...]}
> r=s.group_last_names_set.first[:last_names].count
 => 61
```

```
$rspec spec/lecture3_spec.rb -e rq03
```

### Lecture 4: $match

In this section we will limit documents in the query pipeline to those that match a certain criteria.

1. Reimplement your solution to `group_times` in a new instance method called `groups_faster_than` such that it:
   - accepts time input
   - finds all racers
   - groups the racers into gender and age group
   - counts the number of racers in the group and assigns this to `runners`
   - calculates the fastest time for each group and assigns this value to `fastest_time`
   - **reduces** the list of results to only those that have a fastest time less than or equal to the time provided. This is the only difference from before. (Hint: $match)
   - returns the Mongo result object for the aggregation command

   Note that this will require that you form a `$match` (`:fastest_time`) on a property that is not in the original document from the database.
   You can try out your new method using the irb shell.

   ```
   > r=s.groups_faster_than(1280).to_a
    => [{"_id"=>{"age"=>"50 to 59", "gender"=>"F"}, "runners"=>65, "fastest_time"=>1269},
        {"_id"=>{"age"=>"30 to 39", "gender"=>"M"}, "runners"=>68, "fastest_time"=>1262},
        {"_id"=>{"age"=>"30 to 39", "gender"=>"F"}, "runners"=>59, "fastest_time"=>1270},
        {"_id"=>{"age"=>"masters", "gender"=>"F"}, "runners"=>58, "fastest_time"=>1264}]
   ```

   ```
   $rspec spec/lecture4_spec.rb -e rq01
   ```

2. Reimplement the previous solution to `groups_faster_than` in a new instance method called `age_groups_faster_than` such that it:
   - accepts `criteria_time` and `age_group`
   - **finds** all racers in specified age group ("M" and "F"). This part is different.
   - groups the racers into gender and age group
   - counts the number of racers in the group and assigns this to `runners`
   - calculates the fastest time for each group and assigns this value to `fastest_time`
   - reduces the list of results to only those that have a fastest time less than or equal to the time provided.
   - returns the Mongo result object for the aggregation command

   Note that this can be implemented with two (2) `$match` functions. One prior to the `$group` function and one after.
   You can try out your new method using the irb shell. The gender="M" for this age group did not satisfy the `criteria_time` specified.

   ```
   > r=s.age_groups_faster_than "masters",1280
    => {"_id"=>{"age"=>"masters", "gender"=>"F"}, "runners"=>58, "fastest_time"=>1264}
   ```

   ```
   $rspec spec/lecture4_spec.rb -e rq02
   ```

**Lecture 5:**

In this section we will modify the database.

1. Implement an instance method called `avg_family_time` that
   - accepts a `last_name`
   - finds the racers having that same last name (Hint: `$match`)
   - determines the average of all their race times (Hint: `$group` and `$avg`)
   - forms an array of numbers for each member of the group (Hint: `$group` and `$push`)
   - returns the Mongo result object for the command

   You can try out your new method using the irb shell.

   ```
   > s.avg_family_time("JONES").first
    => {"_id"=>"JONES", "avg_time"=>2006.6, "numbers"=>[3, 4, 5, 6, 7]}
   ```

   ```
   $rspec spec/lecture5_spec.rb -e rq01
   ```

2. Extend the implementation of `avg_family_time` in an instance method called `number_goal`
   - accepts a `last_name`
   - finds the racers having that same last name (Hint: `$match`)
   - determines the average of all their race times (Hint: `$group` and `$avg`)
   - forms an array of numbers for each member of the group (Hint: `$group` and `$push`) This is where the difference starts.
   - forms a result with `avg_time` for each number (Hint: `$unwind`)
   - forms a result with `last_name`, `number`, and `avg_time` for each number in the family with no `_id` property (Hint: `$project`)
   - returns the Mongo result object for the command

   You can try out your new method using the irb shell.

   ```
   > s.number_goal("JONES").each {|r| pp r}
   {"avg_time"=>2006.6, "number"=>3, "last_name"=>"JONES"}
   {"avg_time"=>2006.6, "number"=>4, "last_name"=>"JONES"}
   {"avg_time"=>2006.6, "number"=>5, "last_name"=>"JONES"}
   {"avg_time"=>2006.6, "number"=>6, "last_name"=>"JONES"}
   {"avg_time"=>2006.6, "number"=>7, "last_name"=>"JONES"}
   ```

   ```
   $rspec spec/lecture5_spec.rb -e rq02
   ```

## Self Grading/Feedback

Unit tests have been provided in the bootstrap files that can be used to evaluate your solution. They must be run from the same directory as your solution.

```
$ rspec
........

(N) examples, 0 failures
```

## Submission

There is no submission required for this assignment but the skills learned will be part of a follow-on assignment so please complete this to the requirements of the unit test.

## Last Updated: 2016-01-31