Chapter 3 Get Only What You Need, and Fast

You can now query collections with ease and collect documents to examine and analyze with Python. But this process is sometimes slow and onerous for large collections and documents. This chapter is about various ways to speed up and simplify that process.

- Projection
- Shares of the 1903 Prize in Physics
- Rounding up the G.S. crew
- Doing our share of data validation
- Sorting
- What the sort?
- Sorting together: MongoDB + Python
- Gap years
- What are indexes?
- High-share categories
- Recently single?
- Born and affiliated
- Limits
- Setting a new limit?
- The first five prizes with quarter shares
- Pages of particle-prized people

Shares of the 1903 Prize in Physics

You want to examine the laureates of the 1903 prize in physics and how they split the prize. Here is a query without projection:

```
db.laureates.find_one({"prizes": {"$elemMatch": {"category": "physics", "year": "1903"}}})
```

Which projection(s) will fetch ONLY the laureates' full names and prize share info? I encourage you to experiment with the console and re-familiarize yourself with the structure of laureate collection documents.

```
In [4]:
db.laureates.find_one({"prizes": {"$elemMatch": {"category": "physics", "year": "1903"}}}, projection ={"firstname": 1, "surname": 1, "prizes.share": 1, "_id": 0})
Out[4]:

{'firstname': 'Antoine Henri',
 'surname': 'Becquerel',
 'prizes': [{'share': '2'}]}
```

```
{"firstname": 1, "surname": 1, "prizes.share": 1, "_id": 0}
```

Rounding up the G.S. crew

In chapter 2, you used a regular expression object `Regex` to find values that follow a pattern. We can also use the regular expression operator `$regex` for the same purpose. For example, the following query:

```
{ "name": {"$regex": "^Py"}    }
```

will fetch documents where the field 'name' starts with "Py". Here the caret `symbol ^ means "starts with".`

In this exercise, you will use regular expressions, projection, and list comprehension to collect the full names of laureates whose initials are "G.S.".

- First, use regular expressions to fetch the documents for the laureates whose "firstname" starts with "G" and whose "surname" starts with "S".

```
# Find laureates whose first name starts with "G" and last name
starts with "S"
docs = db.laureates.find(
       filter= {"firstname" : {"$regex" : "^G"},
                  "surname" : {"$regex" : "^S"}  })
# Print the first document
print(docs[0])
```

In the previous step, we fetched all the data for all the laureates with initials G.S. This is unnecessary if we only want their full names!

- Use projection and adjust the query to select only the "firstname" and "surname" fields.

```
# Use projection to select only firstname and surname
docs = db.laureates.find(
        filter= {"firstname" : {"$regex" : "^G"},
                  "surname" : {"$regex" : "^S"}  },
     projection= {'firstname':1,'surname':1}  )

# Print the first document
print(docs[0])
```

```
{'_id': ObjectId('5bc56154f35b634065ba1cb4'), 'firstname': 'Glenn
Theodore', 'surname': 'Seaborg'}
```

Now the documents you fetched contain only the relevant information!

- Iterate over the documents, and for each document, concatenate the first name and the surname fields together with a space in between to obtain full names.

```
# Use projection to select only firstname and surname
docs = db.laureates.find(
       filter= {"firstname" : {"$regex" : "^G"},
                "surname" : {"$regex" : "^S"}  },
   projection= ["firstname", "surname"]  )

# Iterate over docs and concatenate first name and surname
full_names = [doc["firstname"] + " " + doc["surname"] for doc in
docs]

# Print the full names
print(full_names)
```

```
<script.py> output:
    ['Glenn Theodore Seaborg', 'George D. Snell', 'Gustav
Stresemann', 'George Bernard Shaw', 'Giorgos Seferis', 'George J.
Stigler', 'George F. Smoot', 'George E. Smith', 'George P. Smith']
```

Doing our share of data validation

In our Nobel prizes collection, each document has an array of laureate subdocuments "laureates", each containing information such as the prize share for a laureate:

```
{'_id': ObjectId('5bc56145f35b634065ba1997'),
 'category': 'chemistry',
 'laureates': [{'firstname': 'Frances H.',
   'id': '963',
   'motivation': '"for the directed evolution of enzymes"',
   'share': '2',
   'surname': 'Arnold'},
  {'firstname': 'George P.',
   'id': '964',
   'motivation': '"for the phage display of peptides and
antibodies"',
   'share': '4',
   'surname': 'Smith'},
 {...
```

Each "laureates.share" value appears to be the reciprocal of a laureate's fractional share of that prize, encoded as a string. For example, a laureate "share" of "4" means that this laureate received a share of the prize. Let's check that for each prize, all the laureates' shares add up to 1!

Notice the quotes around the values in the "share" field: these values are given as strings! You'll have to convert them to numbers before you find the reciprocals and add up the shares.

- Save a list of prizes (prizes), projecting only the "laureates.share" values for each prize.
- For each prize, compute the total share as follows:

- Initialize the variable total_share to 0.
- Iterate over the laureates for each prize, converting the "share" field of the "laureate" to float and adding the reciprocal of it (that is, 1 divided by it) to total_share.

```
# Save documents, projecting out laureates share
prizes = db.prizes.find({}, {"laureates.share":1})

# Iterate over prizes
for prize in prizes:
    # Initialize total share
    total_share = 0

    # Iterate over laureates for the prize
    for laureate in prize["laureates"]:
        # add the share of the laureate to total_share
        total_share += 1 / float(laureate['share'])

    # Print the total share
    print(total_share)
```

```
<script.py> output:
    1.0
    1.0
…
    1.0
    1.0
```

Phenominal! It seems like all the shares add up to 1 for all the prizes!

What the sort?

This block prints out the first five projections of a sorted query. What "sort" argument fills the blank?

```
docs = list(db.laureates.find(
    {"born":   {"$gte":   "1900"},   "prizes.year":   {"$gte":
"1954"}},
    {"born": 1, "prizes.year": 1, "_id": 0},
    sort=____))
for doc in docs[:5]:
    print(doc)
```

```
{'born': '1916-08-25', 'prizes': [{'year': '1954'}]}
{'born': '1915-06-15', 'prizes': [{'year': '1954'}]}
{'born': '1901-02-28', 'prizes': [{'year': '1954'}, {'year':
'1962'}]}
{'born': '1913-07-12', 'prizes': [{'year': '1955'}]}
{'born': '1911-01-26', 'prizes': [{'year': '1955'}]}
```

[("prizes.year", 1), ("born", -1)]

Yes! Does the 'prizes.year' field sort like you expect?

Sorting together: MongoDB + Python

In this exercise you'll explore the prizes in the physics category.You will use Python to sort laureates for one prize by last name, and then MongoDB to sort prizes by year:

```
1901: Röntgen
1902: Lorentz and Zeeman
1903: Becquerel and Curie and Curie, née Sklodowska
```

You'll start by writing a function that takes a prize document as an argument, extracts all the laureates from that document, arranges them in alphabetical order, and returns a string containing the last names separated by " and ".

The Nobel database is again available to you as db. We also pre-loaded a sample document sample_prize so you can test your laureate-extracting function.

(Remember that you can always type help(function_name) in console to get a refresher on functions you might be less familiar with, e.g. help(sorted)!)

- Save a list of prizes (prizes), projecting out only the "laureates.share" values for each prize.
- For each prize, compute the total share as follows:
- Initialize the variable total_share to 0.

Iterate over the laureates for each prize, converting the "share" field of the "laureate" to float and adding the reciprocal of it (that is, 1 divided by it) to total_share.

```
from operator import itemgetter

def all_laureates(prize):
  # sort the laureates by surname
  sorted_laureates = sorted(prize['laureates'], key=itemgetter('surname'))

  # extract surnames
  surnames = [laureate['surname'] for laureate in sorted_laureates]

  # concatenate surnames separated with " and "
  all_names = " and ".join(surnames)

  return all_names

# test the function on a sample doc
print(all_laureates(sample_prize))
```

```
Becquerel and Curie and Curie, née Sklodowska
```

2) Save a list of prizes (prizes), projecting out only the "laureates.share" values for each prize.

- For each prize, compute the total share as follows:
- Initialize the variable total_share to 0.
- Iterate over the laureates for each prize, converting the "share" field of the "laureate" to float and adding the reciprocal of it (that is, 1 divided by it) to total_share.

```
from operator import itemgetter

def all_laureates(prize):
  # sort the laureates by surname
  sorted_laureates = sorted(prize["laureates"], key=itemgetter("surname"))

  # extract surnames
  surnames = [laureate["surname"] for laureate in sorted_laureates]

  # concatenate surnames separated with " and "
  all_names = " and ".join(surnames)

  return all_names

# find physics prizes, project year and first and last name, and sort by year
docs = db.prizes.find(
           filter= {"category":"physics"},
           projection= ["year", "laureates.firstname", "laureates.surname"],
           sort= [('year', 1)])
```

3) Now that you have the prizes, and the function to extract laureates from a prize, print the year and the names of the

laureates (use your all_laureates() function) for each prize document.

```
from operator import itemgetter

def all_laureates(prize):
  # sort the laureates by surname
  sorted_laureates = sorted(prize["laureates"], key=itemgetter("surname"))

  # extract surnames
  surnames = [laureate["surname"] for laureate in sorted_laureates]

  # concatenate surnames separated with " and "
  all_names = " and ".join(surnames)

  return all_names

# find physics prizes, project year and name, and sort by year
docs = db.prizes.find(
           filter= {"category": "physics"},
           projection= ["year", "laureates.firstname", "laureates.surname"],
           sort= [("year", 1)])

# print the year and laureate names (from all_laureates)
for doc in docs:
  print("{year}: {names}".format(year=doc['year'], names=all_laureates(doc)))
```

```
<script.py> output:
    1901: Röntgen
    1902: Lorentz and Zeeman
    1903: Becquerel and Curie and Curie, née Sklodowska
    1904: (John William Strutt)
    1905: von Lenard
    1906: Thomson
…
    2015: Kajita and McDonald
    2016: Haldane and Kosterlitz and Thouless
    2017: Barish and Thorne and Weiss
    2018: Ashkin and Mourou and Strickland
```

Excellent! You worked through stages of filtering, projecting, sorting, adding a derived field ("names"), and producing formatted output for each document.

Gap years

The prize in economics was not added until 1969. For many years, prizes in one or more of the original categories were not awarded.

In this exercise, you will utilize sorting by multiple fields to see which categories are missing in which years.

For now, you will just print the list of all documents, but in the next chapter, you'll learn how to use MongoDB to group and aggregate data to present this information in a more convenient format.

- Find the original prize categories established in 1901 by looking at the distinct values of the "category" field for prizes from year 1901.
- Fetch ONLY the year and category from all the documents (without the "_id" field).
- Sort by "year" in descending order, then by "category" in ascending order.

```
# original categories from 1901
original_categories = db.prizes.distinct('category', {'year':'1901'})
print(original_categories)

# project year and category, and sort
docs = db.prizes.find(
        filter={},
        projection={"year": 1, "category": 1, "_id": 0},
        sort=[("year", -1), ("category", 1)]
)

#print the documents
for doc in docs:
  print(doc)
```

```
<script.py> output:
[]
{'year': '2018', 'category': 'chemistry'}
{'year': '2018', 'category': 'economics'}
{'year': '2018', 'category': 'medicine'}
{'year': '2018', 'category': 'peace'}
{'year': '2018', 'category': 'physics'}
…
{'year': '1901', 'category': 'literature'}
{'year': '1901', 'category': 'medicine'}
{'year': '1901', 'category': 'peace'}
{'year': '1901', 'category': 'physics'}
```

Great work!

High-share categories

In the year 3030, everybody wants to be a Nobel laureate. Over the last thousand years, many new categories have been added. You serve a MongoDB prizes collection with the same schema as we've seen. Many people theorize that they have a better chance in "high-share" categories. They are hitting your server with similar, long-running queries. It's time to cover those queries with an index.

Which of the following indexes is best suited to speeding up the operation

```
db.prizes.distinct("category", {"laureates.share": {"$gt": "3"}})?
```

```
[("laureates.share", 1), ("category", 1)]
```

Excellent! For a distinct query the filter argument is passed as a second argument, whereas the projected field is passed first.

Recently single?

A prize might be awarded to a single laureate or to several. For each prize category, report the most recent year that a single laureate -- rather than several -- received a prize in that category. As part of this task, you will ensure an index that speeds up finding prizes by category and then sorting results by decreasing year

- Specify an index model that indexes first on category (ascending) and second on year (descending).
- Save a string report for printing the last single-laureate year for each distinct category, one category per line. To do this, for each distinct prize category, find the latest-year prize (requiring a descending sort by year) of that category (so, find matches for that category) with a laureate share of "1".

```
# Specify an index model for compound sorting
index_model = [('category', 1), ('year', -1)]
db.prizes.create_index(index_model)

# Collect the last single-laureate year for each category
report = ""
for category in sorted(db.prizes.distinct("category")):
    doc = db.prizes.find_one(
        {'category': category, "laureates.share": "1"},
        sort=[('year', -1)]
    )
    report += "{category}: {year}\n".format(**doc)

print(report)
```

<script.py> output:

```
    chemistry: 2011
    economics: 2017
    literature: 2017
    medicine: 2016
    peace: 2017
    physics: 1992
```

This code creates an index to optimize the query, finds the latest single-laureate year for each category, and then prints a report with that information.

Simply singular! It seems that physics is the most consistently shared prize category in modern times.

Born and affiliated

Some countries are, for one or more laureates, both their country of birth ("bornCountry") and a country of affiliation for one or more of their prizes ("prizes.affiliations.country"). You will find the five countries of birth with the highest counts of such laureates.

- Specify an index model that indexes first on category (ascending) and second on year (descending).
- Save a string report for printing the last single-laureate year for each distinct category, one category per line. To do this, for each distinct prize category, find the latest-year prize (requiring a descending sort by year) of that category (so, find matches for that category) with a laureate share of "1".

```
from collections import Counter

# Ensure an index on country of birth
db.laureates.create_index([('bornCountry', 1)])

# Collect a count of laureates for each country of birth
n_born_and_affiliated = {
    country: db.laureates.count_documents({
        "bornCountry": country,
        "prizes.affiliations.country": country
    })
    for country in db.laureates.distinct("bornCountry")
}

five_most_common = Counter(n_born_and_affiliated).most_common(5)
print(five_most_common)
```

```
<script.py> output:
    [('USA', 241), ('United Kingdom', 56), ('France', 26), ('Germany', 19),
('Japan', 17)]
```

Good work! As you may guess, simple string matching of country names for this dataset is problematic, but this is a solid first pass.

Setting a new limit?

How many documents does the following expression return?

```
list(db.prizes.find({"category": "economics"},
                    {"year": 1, "_id": 0})
     .sort("year")
     .limit(3)
     .limit(5))
```

5: the second call to limit overrides the first

```
In [1]:
list(db.prizes.find({"category": "economics"},
                    {"year": 1, "_id": 0})
     .sort("year")
     .limit(3)
     .limit(5))
Out[1]:

[{'year': '1969'},
 {'year': '1970'},
 {'year': '1971'},
 {'year': '1972'},
 {'year': '1973'}]
```

Correct! You can think of the query parameters as being updated like a dictionary in Python: d = {'limit': 3}; d.update({'limit': 5}); print(d) will print "{'limit': 5}"

The first five prizes with quarter shares

Find the first five prizes with one or more laureates sharing 1/4 of the prize. Project our prize category, year, and laureates' motivations.

- Save to filter_ the filter document to fetch only prizes with one or more quarter-share laureates, i.e. with a "laureates.share" of "4".
- Save to projection the list of field names so that prize category, year and laureates' motivations ("laureates.motivation") may be fetched for inspection.
- Save to cursor a cursor that will yield prizes, sorted by ascending year. Limit this to five prizes, and sort using the most concise specification.

```
from pprint import pprint

# Fetch prizes with quarter-share laureate(s)
filter_ = {"laureates.share": '4'}

# Save the list of field names
projection = ['category','year', "laureates.motivation"]

# Save a cursor to yield the first five prizes
```

```
cursor = db.prizes.find(filter_, projection).sort('year').limit(5)
pprint(list(cursor))
```

```
<script.py> output:
    [{'_id': ObjectId('5bc56145f35b634065ba1bd5'),
      'category': 'physics',
      'laureates': [{'motivation': '"in recognition of the extraordinary services '
                                   'he has rendered by his discovery of '
                                   'spontaneous radioactivity"'},
                    {'motivation': '"in recognition of the extraordinary services '
                                   'they have rendered by their joint researches '
                                   'on the radiation phenomena discovered by '
                                   'Professor Henri Becquerel"'},
                    {'motivation': '"in recognition of the extraordinary services '
                                   'they have rendered by their joint researches '
                                   'on the radiation phenomena discovered by '
                                   'Professor Henri Becquerel"'}],
      'year': '1903'},
     {'_id': ObjectId('5bc56145f35b634065ba1b2a'),
      'category': 'chemistry',
      'laureates': [{'motivation': '"for his discovery that enzymes can be '
                                   'crystallized"'},
                    {'motivation': '"for their preparation of enzymes and virus '
                                   'proteins in a pure form"'},
                    {'motivation': '"for their preparation of enzymes and virus '
                                   'proteins in a pure form"'}],
      'year': '1946'},
     {'_id': ObjectId('5bc56145f35b634065ba1b26'),
      'category': 'medicine',
      'laureates': [{'motivation': '"for their discovery of the course of the '
                                   'catalytic conversion of glycogen"'},
                    {'motivation': '"for their discovery of the course of the '
                                   'catalytic conversion of glycogen"'},
                    {'motivation': '"for his discovery of the part played by the '
                                   'hormone of the anterior pituitary lobe in the '
                                   'metabolism of sugar"'}],
      'year': '1947'},
     {'_id': ObjectId('5bc56145f35b634065ba1af2'),
      'category': 'medicine',
      'laureates': [{'motivation': '"for their discovery that genes act by '
                                   'regulating definite chemical events"'},
                    {'motivation': '"for their discovery that genes act by '
                                   'regulating definite chemical events"'},
                    {'motivation': '"for his discoveries concerning genetic '
                                   'recombination and the organization of the '
                                   'genetic material of bacteria"'}],
      'year': '1958'},
     {'_id': ObjectId('5bc56145f35b634065ba1ad7'),
      'category': 'physics',
      'laureates': [{'motivation': '"for his contributions to the theory of the '
                                   'atomic nucleus and the elementary particles, '
                                   'particularly through the discovery and '
                                   'application of fundamental symmetry '
                                   'principles"'},
                    {'motivation': '"for their discoveries concerning nuclear '
                                   'shell structure"'},
                    {'motivation': '"for their discoveries concerning nuclear '
                                   'shell structure"'}],
      'year': '1963'}]
```

Great work! For all of these prizes, there were two laureates with quarter shares for their work together, and there was a third laureate with a half share for separate work (as evidenced by the motivation fields).

Pages of particle-prized people

You and a friend want to set up a website that gives information on Nobel laureates with awards relating to particle phenomena.

You want to present these laureates one page at a time, with three laureates per page. You decide to order the laureates chronologically by award year. When there is a "tie" in ordering (i.e. two laureates were awarded prizes in the same year), you want to order them alphabetically by surname.

- Complete the function get_particle_laureates that, given page_number and page_size, retrieves a given page of prize data on laureates who have the word "particle" (use $regex) in their prize motivations ("prizes.motivation"). Sort laureates first by ascending "prizes.year" and next by ascending "surname".
- Collect and save the first nine pages of laureate data to pages.

```
from pprint import pprint

# Write a function to retrieve a page of data
def get_particle_laureates(page_number=1, page_size=3):
    if page_number < 1 or not isinstance(page_number, int):
        raise ValueError("Pages are natural numbers (starting from
1).")
    particle_laureates = list(
        db.laureates.find(
            {"prizes.motivation": {"$regex": "particle"}},
            ["firstname", "surname", "prizes"])
        .sort([('prizes.year', 1), ('surname', 1)])
        .skip(page_size * (page_number - 1))
        .limit(page_size))
    return particle_laureates

# Collect and save the first nine pages
pages = [get_particle_laureates(page_number=page) for page in
range(1,9)]
pprint(pages[0])
```

```
<script.py> output:
    [{'_id': ObjectId('5bc56154f35b634065ba1c3b'),
      'firstname': 'Luis Walter',
      'prizes': [{'affiliations': [{'city': 'Berkeley, CA',
                                    'country': 'USA',
                                    'name': 'University of California'}],
                  'category': 'physics',
                  'motivation': '"for his decisive contributions to elementary '
                                'particle physics, in particular the discovery of '
                                'a large number of resonance states, made possible '
                                'through his development of the technique of using '
                                'hydrogen bubble chamber and data analysis"',
                  'share': '1',
                  'year': '1968'}],
      'surname': 'Alvarez'},
     {'_id': ObjectId('5bc56154f35b634065ba1c47'),
      'firstname': 'Aage Niels',
      'prizes': [{'affiliations': [{'city': 'Copenhagen',
                                    'country': 'Denmark',
                                    'name': 'Niels Bohr Institute'}],
                  'category': 'physics',
                  'motivation': '"for the discovery of the connection between '
                                'collective motion and particle motion in atomic '
```

```
                                'nuclei and the development of the theory of the '
                                'structure of the atomic nucleus based on this '
                                'connection"',
                  'share': '3',
                  'year': '1975'}],
      'surname': 'Bohr'},
     {'_id': ObjectId('5bc56154f35b634065ba1c6f'),
      'firstname': 'Georges',
      'prizes': [{'affiliations': [{'city': 'Paris',
                                    'country': 'France',
                                    'name': 'École Supérieure de Physique et '
                                            'Chimie'},
                                   {'city': 'Geneva',
                                    'country': 'Switzerland',
                                    'name': 'CERN'}],
                  'category': 'physics',
                  'motivation': '"for his invention and development of particle '
                                'detectors, in particular the multiwire '
                                'proportional chamber"',
                  'share': '1',
                  'year': '1992'}],
      'surname': 'Charpak'}]

<script.py> output:
    [{'_id': ObjectId('5bc56154f35b634065ba1c04'),
      'firstname': 'Charles Thomson Rees',
      'prizes': [{'affiliations': [{'city': 'Cambridge',
                                    'country': 'United Kingdom',
                                    'name': 'University of Cambridge'}],
                  'category': 'physics',
                  'motivation': '"for his method of making the paths of '
                                'electrically charged particles visible by '
                                'condensation of vapour"',
                  'share': '2',
                  'year': '1927'}],
      'surname': 'Wilson'},
     {'_id': ObjectId('5bc56154f35b634065ba1c1a'),
      'firstname': 'Sir John Douglas',
      'prizes': [{'affiliations': [{'city': 'Harwell, Berkshire',
                                    'country': 'United Kingdom',
                                    'name': 'Atomic Energy Research '
                                            'Establishment'}],
                  'category': 'physics',
                  'motivation': '"for their pioneer work on the transmutation of '
                                'atomic nuclei by artificially accelerated atomic '
                                'particles"',
                  'share': '2',
                  'year': '1951'}],
      'surname': 'Cockcroft'},
     {'_id': ObjectId('5bc56154f35b634065ba1c1b'),
      'firstname': 'Ernest Thomas Sinton',
      'prizes': [{'affiliations': [{'city': 'Dublin',
                                    'country': 'Ireland',
                                    'name': 'Trinity College'}],
                  'category': 'physics',
                  'motivation': '"for their pioneer work on the transmutation of '
                                'atomic nuclei by artificially accelerated atomic '
                                'particles"',
                  'share': '2',
                  'year': '1951'}],
      'surname': 'Walton'}]
```

Great! Particles may be small, but discoveries related to them have made quite an impact!