April 22: Vladimir Lenin’s Birthday
Housekeeping

Lab 3: now with a deadline (Friday midnight + grace period)
Lab 4: Friday -- Monday, May 4 (gives you time)
Lab 5: Hadoop

Friday: quiz. Be **ON TIME**

Monday: 12:10pm - Lab Test. Read email/slack for details

**Grading:** Lab 2 -> Quiz -> Lab 2 -> Lab Test -> Lab 1
Back into the fray
Very Tersely

Filtering
- Given a condition - keep only objects that satisfy it

Projection Transformation
- Modify the contents of its object based solely on what’s in the object itself

Grouping
- Break collection into groups, each representing objects with same values of some keys

Aggregation
- Compute an aggregate value over a set of objects

Join
- Combine objects from two different collections based on matches in values of some keys

Sort
- Return objects in a specific order
... and a few more

**Ungrouping**

Opposite of grouping - build an object for each element of an array

**Unwinding**

**Limit**

Return a specific number of documents

**Skip**

Return documents after skipping a specified number

**Sample**

Return a random sample of documents

**Facets**

Run multiple operations concurrently, combine results in a single document
This is a lot to take in
How do we actually solve problems with

```
db.collection.aggregate()
```
Key things to remember

Filtering
Selection
Grouping
Aggregation
Faceting

Projection
Transformation
Join
Unwind

Other operations - as needed to assist the main flow
Key things to remember

Selections/Filters are EASY to recognize

For all days in March, find the number of hospitalized people in the state of California.

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.
Key things to remember

Selections/Filters are EASY to recognize

For all days in March, find the number of hospitalized people in the state of California.

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

What are the tell-tales?
Key things to remember

Selections/Filters are EASY to recognize

For all days in March, find the number of hospitalized people in the state of California.

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.
Key things to remember

Selections/Filters are EASY to recognize

For all days in March, find the number of hospitalized people in the state of California.

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

What are the tell-tales?

constants
Key things to remember

Selections/Filters are EASY to recognize

For all days in **March**, find the number of hospitalized people in the state of **California**.

Report each day when the number of new cases exceeded **10%** of the number of cumulative cases.

**What are the tell-tales?**

**constants**

**comparisons**
Key things to remember

Projections are everywhere

Use Case #1: Show only the things we are interested in

Shows up in support of other operations
(selection, join, grouping)

Use Case #2: Transform the output

Central activity in an information request
Key things to remember

Projections are everywhere

Use Case #1: Support

For all days in March, find the number of hospitalized people in the state of California.

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.
Key things to remember

Projections are everywhere

Use Case #1: Support

For all days in March, find the number of hospitalized people in the state of California.

Explicit restrictions

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.
Key things to remember

**Projections are everywhere**

*Compute the ratio of people on ICU to all hospitalized people*

*Create a “status” attribute. Set “status” to “in trouble” if the number of new deaths exceeds 10% of the number of new cases. Otherwise, set status to “coping”.*

**Use Case #2: Main Target**
Key things to remember

Projections are everywhere

Use Case #2: Main Target

Compute the ratio of people on ICU to all hospitalized people

Create a “status” attribute. Set “status” to “in trouble” if the number of new deaths exceeds 10% of the number of new cases. Otherwise, set status to “coping”.

Computation (using single object data)
Key things to remember

Projections are everywhere

**Use Case #2: Main Target**

- **Compute the ratio** of people on ICU to all hospitalized people

- **Create a “status” attribute**. Set “status” to “in trouble” if the number of new deaths exceeds 10% of the number of new cases. Otherwise, set status to “coping”.
Key things to remember

Projections are everywhere

Use Case #3: Implicit Cleanup after Joins/Unwinds/Grouping
Key things to remember

Projections are everywhere

Use Case #3: Implicit Cleanup after Joins/Unwinds/Grouping

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```json
{state: <state>,
  badICUDays: <nDays>}
```
For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
  badICUDays: <nDays>}
```

```javascript
{$match: {...}},
{$group: {_id: "$state",
          badICUDays: {$sum:1}}}
```
Key things to remember

Projections are everywhere

Use Case #3: Implicit

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
 badICUDays: <nDays>}
```

```
{$match: {...}},
{$group: {_id: "$state",
           badICUDays: {$sum:1}}}
```

```
{ _id: "CA",
  badICUDays: 21 }
```
Key things to remember

Projections are everywhere

Use Case #3: Implicit

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
  badICUDays: <nDays>}
```

```
{$match: {...}},
{$group: {_id: "$state",
  badICUDays: {$sum:1}}}
```

```
{ _id: "CA",
  badICUDays: 21 }
```
Key things to remember

Projections are everywhere

Use Case #3: Implicit

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
  badICUDays: <nDays>}
```

```javascript
{{$match: {...}},
 {{$group: {_id:"$state",
         badICUDays: {$sum:1}}},
  {$project: {_id:0, state:"$_id"}}

{state: "CA",
  badICUDays: 21 }
```
Key things to remember

Grouping combines data from multiple documents into one

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
  badICUDays: <nDays>}
```

Is this a grouping and aggregation query?
Key things to remember

Grouping combines data from multiple documents into one

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

\[
\{\text{state: } \langle\text{state}\rangle, \\
\text{badICUDays: } \langle\text{nDays}\rangle\}
\]

Is this a grouping and aggregation query?

Yes, with daily.json data
For each state report the total number of days with more than 10 ICU patients.

Report results in the form:

{state: <state>,
  badICUDays: <nDays>}

Is this a grouping and aggregation query?

Yes, with daily.json data

```json
{
  "_id" : ObjectId("5e941e9cf9e720b73b7d96ff"),
  "date" : 20200405,
  "state" : "AK",
  "positive" : 185,
  "negative" : 6099,
  "pending" : null,
  "hospitalizedCurrently" : null,
  "hospitalizedCumulative" : 20,
  "inIcuCurrently" : 12,
  "inIcuCumulative" : null,
  "onVentilatorCurrently" : null,
  "onVentilatorCumulative" : null,
  "recovered" : null,
  "hash" : "661d7b0f627847a2dceb5d704e9260965031cc2",
  "dateChecked" : "2020-04-05T20:00:00Z",
  "death" : 6,
  "hospitalized" : 20,
  "total" : 6284,
  "totalTestResults" : 6284,
  "posNeg" : 6284,
  "fips" : "02",
  "deathIncrease" : 1,
  "hospitalizedIncrease" : 4,
  "negativeIncrease" : 230,
  "positiveIncrease" : 14,
  "totalTestResultsIncrease" : 244
}
```
Key things to remember

Grouping combines data from multiple documents into one

For each state, report the total number of days with more than 10 ICU patients.
Report results in the form:

```
{state: <state>,
 badICUDays: <nDays>}
```

Is this a grouping and aggregation query?

No, with other input data
Key things to remember

Grouping combines data from multiple documents into one.

For each state report the total number of days with more than 10 ICU patients.

Report results in the form:

{state: <state>,
  badICUDays: <nDays>}

Is this a grouping and aggregation query?

No, with other input data

{ "_id" : 8888,
  state: "CA",
  month: "March",
  badICUDays: 9,
  goodICUDays: 4,
  noInfo: 17,
  cumulativeICUPatients: 88
}
Key things to remember

Grouping combines data from multiple documents into one

For each state report the total number of days with more than 10 ICU patients. Report results in the form:

```
{state: <state>,
  badICUDays: <nDays>}
```

KNOW YOUR DATA!!!
Key things to remember

Grouping does NOT always mean aggregation

For each state create a list of dates when there were more than 10 ICU patients
Key things to remember

**Grouping does NOT always mean aggregation**

*For each state create a list of dates when there were more than 10 ICU patients*

$push
$addToSet

Are your biggest friends!
Key things to remember

We can “hide” information while grouping

`$push`  
`$addToSet`

Are your biggest friends!

```javascript
{$group:
   {_id:"$state",
    avgPatients:{$avg:"$hospitalized"},
    $push: {$hospitalized}
   }
}
```
Key things to remember

We can “hide” information while grouping

$push
$addToSet  Create array attributes

We can “unhide” information AFTER grouping
Key things to remember

We can “hide” information while grouping

$push
$addToSet

Create array attributes

We can “unhide” information AFTER grouping

```javascript
${group:
  {_id: "$state",
   avgPatients: {$avg: "$hospitalized"},
   data: $push: {$hospitalized}
  },

{${unwind: "$data"}}}
```

$unwind after $group
Key things to remember

Grouping combines data from multiple documents into one

Grouping does NOT always mean aggregation

We can “hide” information while grouping

We can “unhide” information AFTER grouping

Compound Keys  $first, $last  constant key values
Key Things To Remember

Joins involve comparisons of documents to documents
Key Things To Remember

Joins involve comparisons of documents to documents

Use Case #1: Join to different collections

Use Case #2: Self Join

Often can be avoided by embedding documents
SELECT * FROM Students s, take t, Courses c
WHERE s.StudentID = t.StudentId and t.Course = c.CourseID
    and t.Section = c.Section
Students take Courses

(StudentID, Name)  (StudentID, Course, Section)  (CourseID, Name, Section, Instructor)

```
{course: "CSC 369",
 roster: [{student: "Bob Smith"},
          {student: "Alice Lee"},
          ...]
}
```
Key Things To Remember

Joins involve comparisons of documents to documents

**Use Case #1**: Join to different collections

**Use Case #2**: Self Join

$lookup is expensive. Self Joins can be “tricked”
Key Things To Remember

Joins involve comparisons of documents to documents

**Use Case #1**: Join to different collections

**Use Case #2**: Self Join

$lookup is expensive. Self Joins can be “tricked” with $group
Tricks and dealing with MongoDB idiosyncrasies
Trick 1: $project as a filter

Problem: $match cannot compare two attributes to each other

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

```json
{ ... 
  positive: 566,
  positiveIncrease: 65 
  ... 
}
```
Trick 1: $project as a filter

Problem: $match cannot compare two attributes to each other

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

```json
{ ...
  positive: 566,
  positiveIncrease: 65
 ...
}

{$project:
  {flag: {$cond: [{gt: [{positiveIncrease: 65}, {$multiply: [{positive: 0.1}]}, True, False]}}]}}
```
Trick 1: $project as a filter

Problem: $match cannot compare two attributes to each other

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

```json
{ ...
  positive: 566,
  positiveIncrease: 65
 ...
}
```

```javascript
{$project:
  {flag: {$cond: [{gt: [{"$positiveIncrease":
    {$multiply: ["$positive",
      0.1]}]},
      True,
      False
]},
  ]}
},

{$match: {flag: True}}
```
Trick 1: $project as a filter

Problem: $match cannot compare two attributes to each other

Report each day when the number of new cases exceeded 10% of the number of cumulative cases.

```json
{ ...
    positive: 566,
    positiveIncrease: 65
 ...
}

{ $project:
    { flag: {
        $cond: [{
            $gt: [{
                "positiveIncrease",
                $multiply: [{"positive",
                              0.1}]
            }],
        True,
        False
        ]}
    },
    $match: { flag: True }
}
Trick 1: $project as a filter

Problem: $match cannot compare two attributes to each other

All computations can and should be done in $project
Trick 2: Who has the optimal value?

Problem: \{\$sort: \{foo:-1\}\},\{limit:1\} fails when there are ties

Report the state and the date of the largest single increase in the number of positive cases.
Trick 2: Who has the optimal value?

Problem: \{\text{$sort$: \{\text{foo:-1}\}},\text{$limit$:1}\} fails when there are ties

Report the state and the date of the largest single increase in the number of positive cases.

Step 1: Use $group $push to “hide” data
Use constant for grouping value

Step 1.5: Unwind
Trick 2: Who has the optimal value?

Report the state and the date of the largest single increase in the number of positive cases.

```
{   _id: "1",     largestIncrease: 10841     data: [{...},{...},..., {...}] }

{$unwind: "$data"}

{   _id: "1",     largestIncrease: 10841     data: {...} }
```
Trick 2: Who has the optimal value?

Report the state and the date of the largest single increase in the number of positive cases.

```json
{   _id: "1",     largestIncrease: 10841     data: [{...},{...},..., {...}] }

{$unwind:"$data"}

{   _id: "1",     largestIncrease: 10841     data: {...} }

Get rid of embedding if needed
Trick 2: Who has the optimal value?

Problem: \{\$sort: \{foo:-1\}\},\{limit:1\} fails when there are ties

Report the state and the date of the largest single increase in the number of positive cases.

Step 1: Use $group $push to “hide” data
Use constant for grouping value

Step 1.5: Unwind
And $project if desired

Step 2. See Trick 1 to finish
Trick 3: Join Avoidance

Self-joins can be done outside of $lookup

But with some painful manipulations

Leverage $group $push / $addToSet
$unwind
$project
Trick 4: Generalizing Joins

Problem: `$lookup` is a left outer equijoin

Joins can be more complex:
Trick 4: Generalizing Joins

Problem: $lookup is a left outer equijoin

Joins can be more complex:

Report governors of all states with less than 400 positive cases per million on April 4, 2020

```
daily.json

{state: “CA”
governor:”Gavin Newsom”
population: 39510000}
```
Trick 4: Generalizing Joins

Problem: $lookup is a left outer equijoin

Joins can be more complex:

```
Report governors of all states with less than 400 positive cases per million on April 4, 2020
```

```
{...
    state: "CA"
    positive: 12026
...
}
```

```
{state: "CA"
  governor: "Gavin Newsom"
  population: 39510000}
```
Trick 4: Generalizing Joins

Problem: $lookup is a left outer equijoin

Joins can be more complex:

Report governors of all states with less than 400 positive cases per million on April 4, 2020

Join = Cartesian Product followed by Selection
Trick 4: Generalizing Joins

Problem: `$lookup` is a left outer equijoin.

Joins can be more complex:

Report governors of all states with less than 400 positive cases per million on April 4, 2020.

Join = Cartesian Product followed by Selection

Use for comparisons (Trick 1)