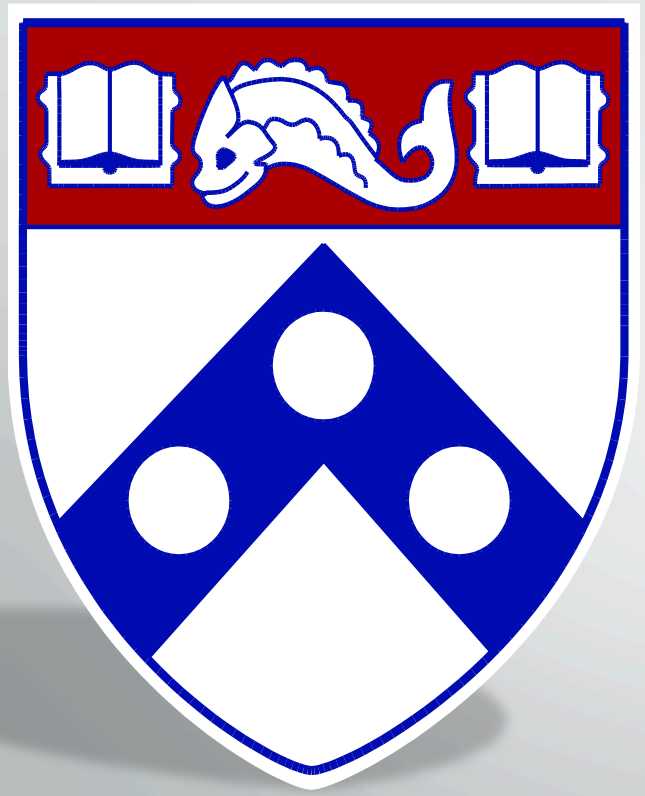


JSON, MongoDB, JSONique

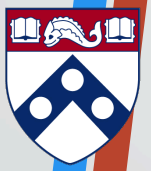
Susan B. Davidson

CIS 700: Advanced Topics in Databases

MW 1:30-3

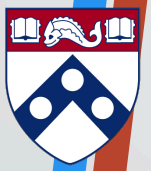
Towne 309

http://www.cis.upenn.edu/~susan/cis700/homepage.html



JSON

- JSON (JavaScript Object Notation) is a lightweight data exchange format for structured data
- Supports objects (string-to-value maps), arrays (ordered sequences of value), other simple types (integers, strings, reals, booleans)
- MongoDB is a NoSQL solution based on JSON
- JSONiq is a query language for JSON based on XQuery



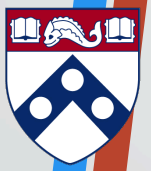
Sample JSON Document

```
{  _id: 1,
   name: { first: "John", last: "Backus" },
   birthyear: 1924,
   contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
   awards: [ { award_id: "NMS001",
               year: 1975 },
             { award_id: "TA99",
               year: 1977} ]
              }
```

MongoDB: Always indexed, automatically assigned unless provided

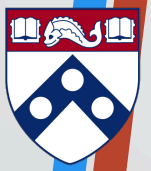
Array of documents

MongoDB



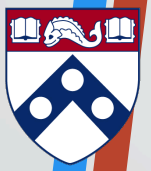
MongoDB Querying

- Use find( ) function and a query document
- Ranges, set inclusion, inequalities using $conditionals
- Complex queries using $where clause
- Queries return a database cursor
- Meta-operations on cursor include skipping some number of results, limiting the number of results returned, sorting results.



Sample document

```
d={
    _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
    author : "Kevin",
    date : new Date("February 2, 2012"),
    text : "About MongoDB...",
    birthyear:  1980,
    tags : [ "tech", "databases" ]
    }

> db.posts.insert(d)
```



Find

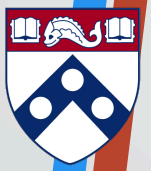
Return entire collection in posts:

```
db.posts.find( )
```

Return posts that match condition (conjunction):

```
db.posts.find({author: "Kevin", birthyear: 1980})
```

```
{ _id : ObjectId("4c4ba5c0672c685e5e8aabf3"), author : "Kevin",
date : Date("February 2, 2012"), birthyear:  1980,
 text : "About MongoDB...",  tags : [ "tech", "databases" ]}
```



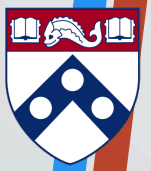
Specifying which keys to return

```
db.people.find({}, {name:1, contribs:1})
```

```
{
   _id: 1,
   name: { first: "John", last: "Backus" },
   contribs: [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ]
}
```

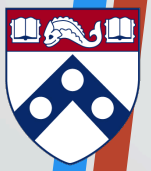
```
db.people.find({}, {_id: 0, name:1})
```

```
{
   name: { first: "John", last: "Backus" }
}
```



Ranges, Negation, OR-clauses

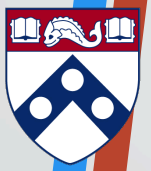
- Comparison operators: $lt, $lte, $gt, $gte
  - db.posts.find({birthyear: {$gte: 1970, $lte: 1990}})
- Negation: $ne
  - db.posts.find({birthyear: {$ne: 1982}})
- Or queries: $in (single key), $or (different keys)
  - db.posts.find({birthyear: {$in: [1982, 1985]}})
  - db.posts.find({$or: [{birthyear: 1982}, {author:"John"}]})



Arrays

- db.posts.find({tags: “tech”})
  - Print complete information about posts which are tagged “tech”
- db.posts.find({tags: {$all: [“tech”, “databases”]}, {author:1, tags:1})
  - Print author and tags of posts which are tagged with both “tech” and “databases” (among other things)
  - Contrast this with:

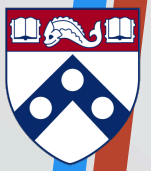
db.posts.find({tags: [“databases”, “tech”]})



Querying Embedded Documents

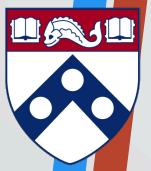
- db.people.find({“name.first”: “John”})
  - Finds all people with first name John
- db.people.find({“name.first”: “John”, “name.last”: “Smith”})
  - Finds all people with first name John and last name Smith.
  - Contrast with

db.people.find({“name”: {“first”: “John”, “last”: “Smith”}})



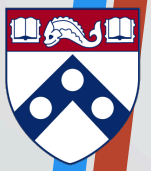
Joins in MongoDB

- "Do joins while write, not on reads."
  - Use embedded relationships
- Otherwise, you need to use semi-joins to get an array of keys from the first collection on which to search the second collection for matches using cursor methods



Relationships: Embedded

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1924,
  contribs: [ "Fortran", "ALGOL",
              "Backus Naur Form", "FP" ],
  awards: [ {title: "National Medal of Science" ,
             by: "National Science Foundation",
             year: 1975 },
            {title: "Turing Award",
             by: "ACM" ,
             year: 1977} ]  }
```



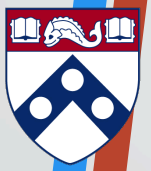
Relationships: Referenced

People:

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1924,
  contribs: [ "Fortran", "ALGOL",
              "Backus Naur Form", "FP" ],
  awards: [ { award_id: "NMS001", year: 1975 },
            { award_id: "TA99",  year: 1977} ]  }
```

Awards:

```
{_id: "NMS001",
 title:  "National Medal of Science" ,
 by: "National Science Foundation"}
{_id: "TA99",
 title:  "Turing Award",
 by: "ACM" }
```

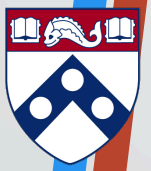


"SemiJoins"

- Suppose you want to print people who have won Turing Awards using referenced relationship
  - Problem: object id of Turing Award is in collection "awards", collection "people" references it.

```
turing= db.awards.findOne({title: "Turing Award"})
db.people.find({"awards.award_id": turing._id]})
```

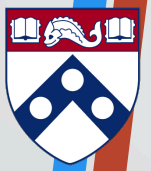
  - But this only works for one award with title "Turing Award", suppose there were more.



Iterating using cursors

- Now suppose there is more than one award named "Turing Award"

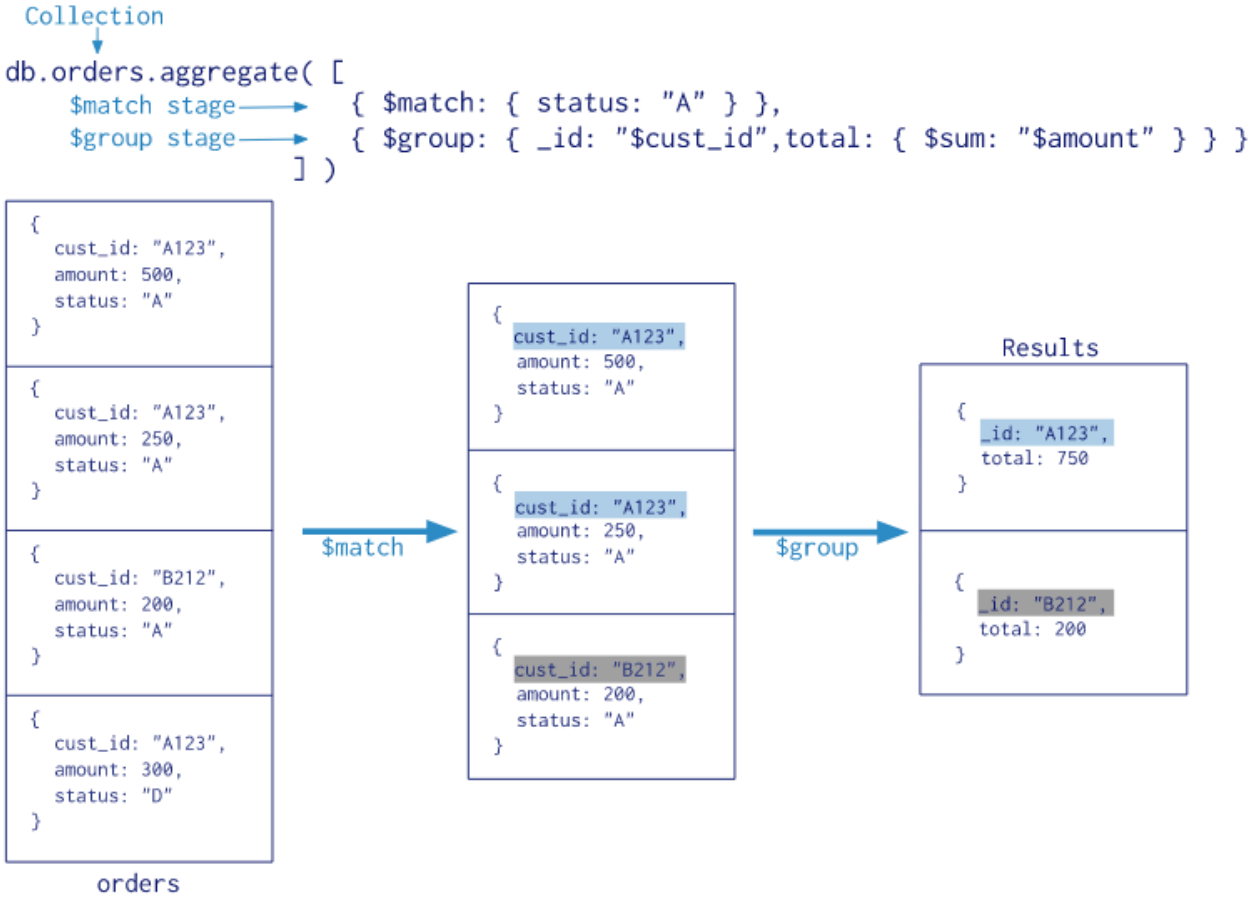
```
turing= db.awards.findMany ({title: "Turing Award"})

while (turing.hasNext())  {
db.people.find({"awards.award_id": turing.next()._id})
}
```

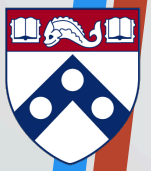


Aggregation

- A framework to provide "group-by" and aggregate functionality without the overhead of map-reduce.
- Conceptually, documents from a collection pass through an aggregation pipeline, which transforms the objects as they pass through (similar to UNIX pipe "|")
- Operators include: $project, $match, $group, $sort, $skip, $limit, $unwind

https://docs.mongodb.com/manual/aggregation/





Aggregation: $group

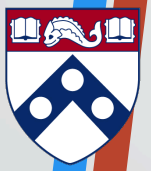
- Every group expression must specify an _id field.
- Suppose we wanted to find how many people were born each year

```
> db.people.aggregate( { $group :
     { _id : "$birthyear", birthsPerYear : { $sum : 1}} )
```

```
{ "result" : [ { "_id" : 1924, "birthsPerYear" : 1 } ], "ok" : 1 }
```

- Contrast with aggregate operation over entire result

```
> db.people.count( )
> db.people.find({“name.first”: “John”}).count( )
> db.people.count({“name.first”: “John”})
```



Aggregation: $unwind

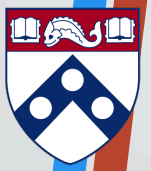
- Deconstructs an array field to output a document for each element.

```
Posts:   {
         _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
         author : "Kevin",
         date : new Date("February 2, 2012"),
         text : "About MongoDB...",
         birthyear:  1980,
         tags : [ "tech", "databases" ]
         }
```

```
>db.posts.aggregate( { $project : { author : 1, tags : 1 }}, { $unwind : "$tags" } )
```

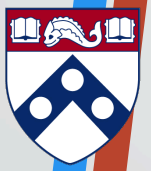
Result of unwind

{ "result" : [{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
"author" : "Kevin",
"tags" : ”tech" },
{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
"author" : "Kevin",
"tags" : ”databases" }],
"OK" : 1 }



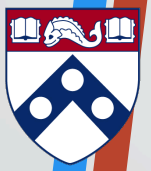
Summary of MongoDB

- The MongoDB query language is limited, and oriented around "collection" (relation) at a time processing
  - Joins are done via a query language
- The power of the solution lies in the distributed, parallel nature of query processing (not covered)
  - Replication and sharding



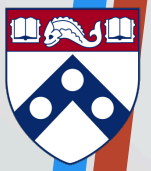
Wouldn’t it be nice if there was a better language for JSON?

There is … JSONiq



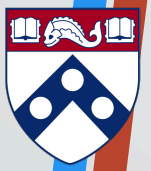
JSONiq

- JSONiq borrows a lot from XQuery
  - structure and semantics of a FLWOR construct
  - functional aspect of the language
- However it is not concerned with the "peculiarities" of XML
  - mixed content
  - ordered children
  - confusion between attributes and elements
  - complexities of namespaces and XML Schema, etc.



prizes.json


```
{"year": "2017”,
"category": "physics”,
"laureates": [{"id": "941”,"firstname": "Rainer”,"surname": "Weiss",
            "motivation": "\"for contributions to the LIGO Detector \"",
            "share": "2”},
               {"id": "942”,"firstname": "Barry C.”,"surname": "Barish",
            "motivation": "\"for contributions to the LIGO detector\"",
             "share": "4”},
             {"id": "943”,"firstname": "Kip S.”,"surname": "Thorne”,
             "motivation": "\"for contributions to the LIGO detector\"”,
             "share": "4”}]
        }
```



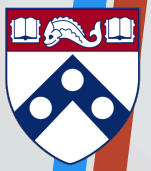
Total number of Nobel prizes in medicine

MongoDB

```
db.prizes.find({"category":"medicine"}).count()
```

JSONiq

```
return count(
for $i in $prizes
where $i.category="medicine"
return $i)
```



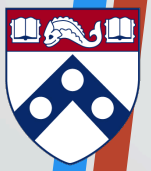
Nobel Laureates who are the sole recipients of a prize in physics

MongoDB

```
db.prizes.find({"category": "physics", "laureates": {$size: 1}})
```

JSONiq

```
for $i in $prizes
where size($i.laureates)=1 and $i.category="physics"
return $i
```



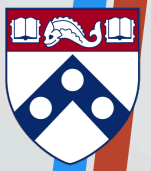
Number of Nobel Laureates who were either born in Philadelphia or affiliated with Penn.

MongoDB

```
db.laureates.find({$or: [{"bornCity": "Philadelphia, PA"},
{"prizes.affiliations.name": "University of Pennsylvania"}]}).count()
```

JSONiq

```
return count(
for $i in $laureates, $j in jn:members($i.prizes),
    $k in jn:members($j.affiliations)
where $i.bornCity="Philadelphia, PA" or
        $k.name="University of Pennsylvania"
return $i)
```



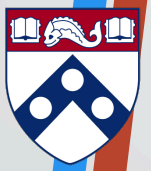
First and last names of all the female Nobel prize Laureates who have won a Nobel prize in either Physics or Chemistry.

MongoDB

```
db.laureates.find({$and: [
{$or: [{"prizes.category": "physics"}, {"prizes.category": "chemistry"}]},
{"gender":"female"}]}, {"firstname":1, "surname": 1, "_id": 0})
```

JSONiq

```
for $i in $laureates, $j in jn:members($i.prizes)
where $i.gender="female" and
      ($j.category="physics" or $j.category="chemistry")
return {firstname: $i.firstname, lastname: $i.surname}
```



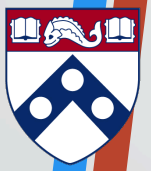
For each of the categories, print the number of Nobel prizes awarded, sort them in decreasing order.

MongoDB

```
db.prizes.aggregate([{$group: {_id: "$category", num: {$sum: 1}}},
{$sort: {num: -1}}])
```

JSONiq

```
for $i in $prizes
group by $category:= $i.category
order by count($i) descending
return {category: $category, "count": count($i)}
```



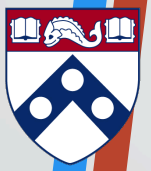
Years where Nobel Prizes were not awarded in all the six categories.

MongoDB

```
db.prizes.aggregate([{$group: {_id: "$year", num: {$sum: 1}}},
{$match: {num: {$lt: 6}}}])
```

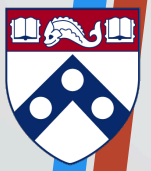
JSONiq

```
for $i in $prizes
group by $year:= $i.year
where count($i)<6
return $year
```



laureates.json

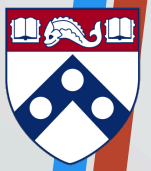
{"id": "3",
"firstname": "Pieter", "surname": "Zeeman",
"born": "1865-05-25", "died": "1943-10-09",
"bornCountry": "the Netherlands”, "bornCity": "Zonnemaire",
"diedCountry": "the Netherlands",
"diedCity": "Amsterdam",
"gender": "male",
"prizes": [{"year": "1902”,"category": "physics”,"share": "2",
"motivation": "\”influence of magnetism …\"",
"affiliations": [{"name": "Amsterdam University”,
"city": "Amsterdam",
"country": "the Netherlands”
}]
}



Print the DOB of each laureate who won the Nobel prizes in Physics with John Bardeen.

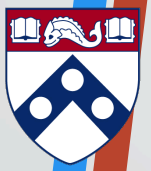
MongoDB

```
var arr =[]
db.prizes.find({"laureates.firstname":"John”,
                ”laureates.surname":"Bardeen"},
{"laureates.id": 1}).forEach(function(doc)
{doc.laureates.forEach(function(x) arr[arr.length] = x.id)})

db.laureates.find().forEach(function(doc)
{if(arr.indexOf(doc.id) = -1) {printjson(doc.born)}})
```



JSONiq

```
for $i in $prizes, $j in jn:members($i.laureates)
where $j.firstname="John" and $j.surname="Bardeen"
return (for $k in jn:members($i.laureates), $l in $laureates
      where $k.id= $l.id and $l.id ne $j.id
      return $l.born)
```



JSON

- JSON (JavaScript Object Notation) is a lightweight data exchange format for structured data
- MongoDB is a NoSQL solution based on JSON: good for simple queries, sharding/parallelism are features
- JSONiq is a query language for JSON based on XQuery: good for joins and complex queries, Turing complete