From Structured to Semistructured

substructures

```
[
① {
      _id: 1,
      name: "sue",
      age: 19,
      type: 1,
      status: "P",
      favorites: { artist: "Picasso", food: "pizza" },
      finished: [ 17, 3 ],
      badges: [ "blue", "black" ],
      points: [
         { points: 85, bonus: 20 },
         { points: 75, bonus: 10 }
      ]
   },
②{
      _id: 2,
      name: "john",
      age: 21
   }
]
```

Key-value pair
(key → value)

Named Tuple
(Tuple-key → tuple)
(Tuple-key, attrib-key→attrib-value)

Named Array
(Array-key→array)
(Array-key, position → array-element)
(Array-key, value-list → matching-values)

Named Array of unnamed Tuples — addressed by their positions

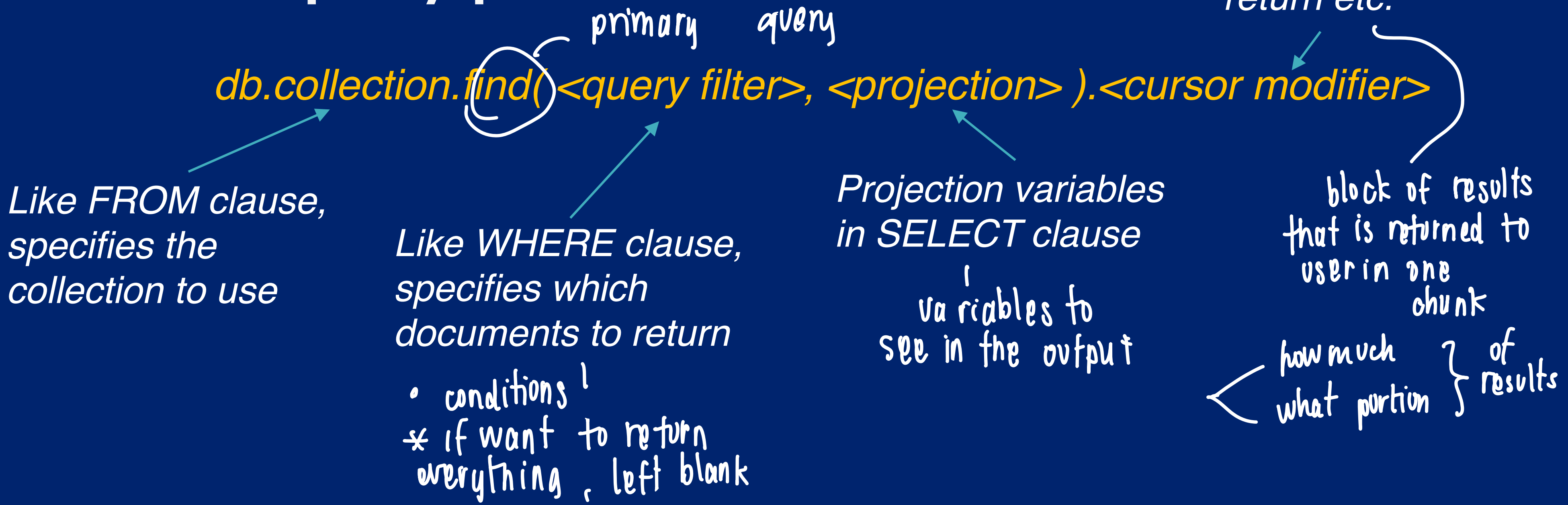
*nesting

* operations to navigate from one structure to any of the embedded structures

SQL SELECT and MongoDB find()

states which parts of which documents from doc collection should be returned

- MongoDB is a collection of documents
- The basic query primitive



primary query

`db.collection.find( <query filter>, <projection> ).<cursor modifier>`

Like FROM clause, specifies the collection to use

Like WHERE clause, specifies which documents to return

- conditions !

* if want to return everything, left blank

Projection variables in SELECT clause

variables to see in the output

How many results to return etc.

block of results that is returned to user in one chunk

how much / what portion } of results

Some Simple Queries

- **Query 1**
  - SQL
    - SELECT * FROM Beers
  - MongoDB
    - db.Beers.find()
- **Query 2**
  - SQL
    - SELECT beer, price FROM Sells
  - MongoDB
    - db.Sells.find(
    - { },
    - { beer: 1, price: 1}
    -)

↳ 1 if attribute is output → (only variables with 1 are required)
0 if it is not

{ beer: 1, price: 1, _id: 0} — to not return this designated attribute

every mongoDB doc has this identifier

Adding Query Conditions

- **Query 3**
  - SQL
    - SELECT manf FROM Beers WHERE name = ‘Heineken’
  - MongoDB
    - db.Beers.find({ name: “Heineken" }, { manf: 1, _id: 0 })
- **Query 4**
  - SQL
    - SELECT DISTINCT beer, price FROM Sells WHERE price > 15
  - MongoDB
    - db.Sells.distinct({price: {$gt: 15} }, {beer:1, price:1, _id:0})

mongoDB 's name for operator

non equality operators in a query

special query functions for operations

Some Operators of MongoDB

Symbol	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$in	Matches any of the values specified in an array.
$nin	Matches none of the values specified in an array.
$or	Joins query clauses with a logical OR.
$and	Joins query clauses with a logical AND.
$not	Inverts the effect of a query expression.
$nor	Joins query clauses with a logical NOR.

comparison

array operations

logical operations that combine two conditions in different ways

→ used to specify queries when neither of two conditions must hold

URL For MongoDB operators
https://docs.mongodb.com/manual/reference/operator/query/ ← find here ✳

Regular Expressions

— to specify partial string matches

- Query 5
  - Count the number of manufacturers whose names have the partial string “am” in it – must be case insensitive
    - db.Beers.find(name: {$regex: /am/i}).count()

      post operation

- Query 6
  - Same, but name starts with “Am”
    - db.Beers.find(name: {$regex: /^Am/}).count()

      partial string is at beginning of name
  - Starts with “Am” ends with “corp”
    - db.Beers.count(name: {$regex: /^Am.*corp$/})

      any character

      zero or more

      has a number of characters in the middle

      must appear at the end

      find( ). count( )

Array Operations

consider array as list — intersection operations ① ②

position ③ ④

- ① Find items which are tagged as “popular” or “organic”
  - db.inventory.find({tags: {$in: [“popular”, “organic”]}}) →

if this strings belong to the array

- ② Find items which are *not* tagged as “popular” nor “organic” * when there's intersection nothing is returned
  - db.inventory.find({tags: {$nin: [“popular”, “organic”]}})

{ _id: 1,
item: “bud",
qty: 10,
tags: [“popular”, “summer”, “Japanese”],
rating: “good” }

- ③ Find the 2nd and 3rd elements of tags
  - db.inventory.find({}, { tags: { $slice: [1, 2] } }) → [“summer”, “japanese”]

number of variable limits to skip — *Skip count*

Return how many — number of variable limits to extract after skipping

  - db.inventory.find({}, tags: {$slice:-2}}) — two elements; system should count from the end
- ④ Find a document whose 2nd element in tags is “summer”
  - db.inventory.find(tags.1 ”summer”)

Compound Statements

– queries with multiple query conditions

```
db.inventory.find( {
  $and : [
    { $or : [ { price : 3.99 }, { price : 4.99 } ] },
    { $or : [ { rating : good }, { qty : { $lt : 20 } } ] }
    {item: {$ne: "Coors"}}
  ]
} )
```

$and
$or } need a list (array) of arguments

desired item should not be Coors

```
{ _id: 1,
item: "bud",
qty: 10,
tags: [ "popular", "summer",
"Japanese"],
rating: "good",
price: 3.99 }
```

```
SELECT * FROM inventory
WHERE ((price = 3.99) OR (price=4.99)) AND
        ((rating = "good") OR (qty < 20)) AND
         item != "Coors"
```

Queries over Nested Elements

①
```
_id: 1,
    points: [
        { points: 96, bonus: 20 },
        { points: 35, bonus: 10 }
    ]
```

②
```
_id: 2,
    points: [
        { points: 53, bonus: 20 },
        { points: 64, bonus: 12 }
    ]
```

③
```
_id: 3,
    points: [
        { points: 81, bonus: 8 },
        { points: 95, bonus: 20}
    ]
```

three docs part of a collection

first tuple

first element of that tuple

- `db.users.find( { 'points.0.points': { $lte: 80 } } )` ②

points in ANY tuple

- `db.users.find( { 'points.points': { $lte: 80 } } )` ①②

without array index specified

- `db.users.find( { "points.points": { $lte: 81 }, "points.bonus": 20 } )` ①② ~~③~~

comma is implicit AND in SAME TUPLE

MongoDB does not have adequate support to perform recursive queries over nested substructures