From Structured to Semistructured

```
[
  {
    _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 }
    ]
  },
  {
    _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

SQL SELECT and MongoDB find()

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

How many results to return etc.

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

Like FROM clause, specifies the collection to use

Like WHERE clause, specifies which documents to return

Projection variables in SELECT clause

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}
    -)

{ beer: 1, price: 1, _id: 0}

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})

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.

URL For MongoDB operators
https://docs.mongodb.com/manual/reference/operator/query/

Regular Expressions

- **Query 5**
  - Count the number of manufacturers whose names have the partial string “am” in it – must be case insensitive
    - db.Beers.find(name: {$regex: /am/i}).count()
- **Query 6**
  - Same, but name starts with “Am”
    - db.Beers.find(name: {$regex: /^Am/}).count()
  - Starts with “Am” ends with “corp”
    - db.Beers.count(name: {$regex: /^Am.*corp$/})

Array Operations

- **Find items which are tagged as "popular" or "organic"**
  - db.inventory.find({tags: {$in: ["popular", "organic"]})
- **Find items which are *not* tagged as "popular" nor "organic"**
  - db.inventory.find({tags: {$nin: ["popular", "organic"]})
- **Find the 2nd and 3rd elements of tags**
  - db.inventory.find({}, { tags: { $slice: [1, 2] } })

  - db.inventory.find({}, tags: {$slice: -2})
- **Find a document whose 2nd element in tags is "summer"**
  - db.inventory.find(tags.1: "summer")

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

Compound Statements

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

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

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

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

- db.users.find({ 'points.0.points': { $lte: 80 } })
- db.users.find({ 'points.points': { $lte: 80 } })
- db.users.find({ "points.points": { $lte: 81 }, "points.bonus": 20 })

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