Mongoose

Week 9 Lecture

Outline

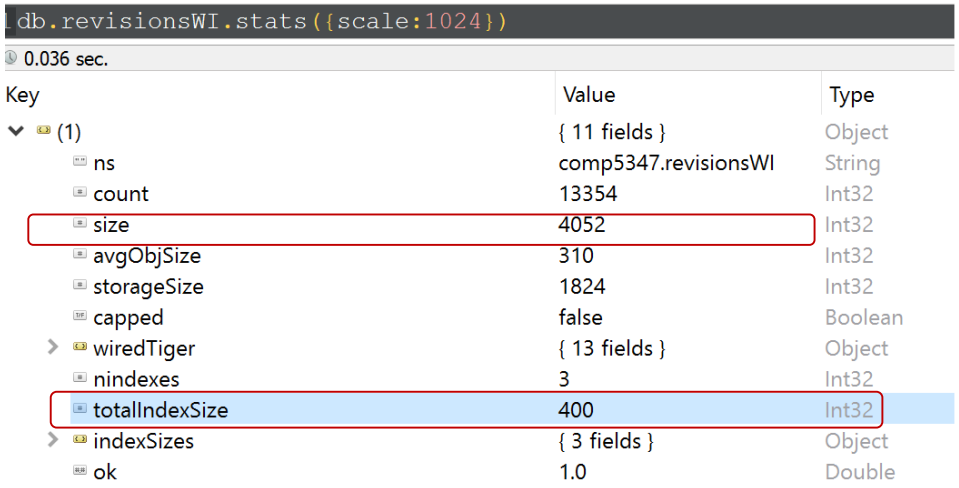
- MongoDB indexing
- Mongoose

Indexing

- An index on an attribute/field **A** of a table/collection is a data structure that makes it efficient to find those rows(document) that have a requited value for attribute/field **A**.
- An index consists of records (called index entries) each of which has a value for the attribute(s) eg of the form

attr. value	Pointer to data record

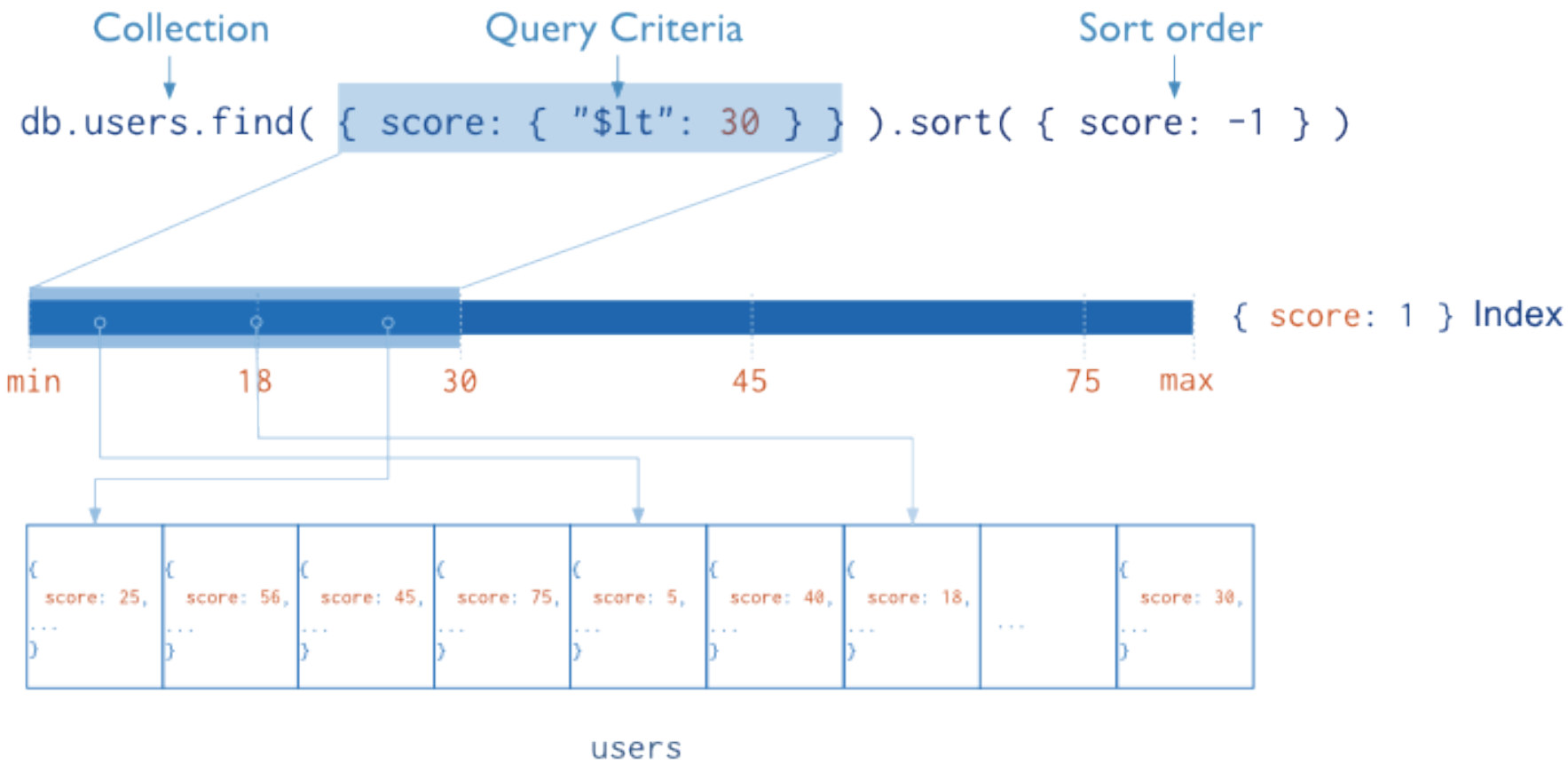
- Index files are typically much smaller than the original file



MongoDB Basic Indexes

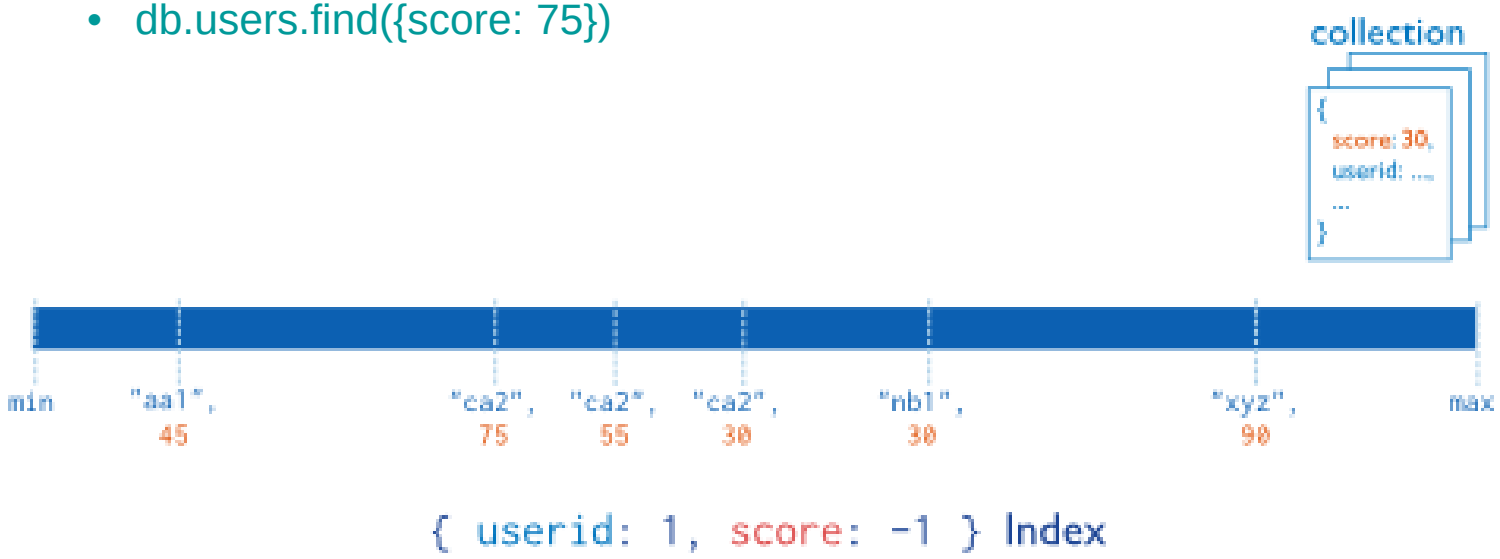
- The **_id** index
  - **_id** field is automatically indexed for all collections
  - The **_id** index enforces uniqueness for its keys
- Indexing on other fields
  - Index can be created on any other field or combination of fields
    - `db.<collectionName>.createIndex({<fieldName>:1});`
    - **fieldName** can be a simple field, array field or field of an embedded document (using dot notation)
      - db.blog.createIndex({author:1})
      - db.blog.createIndex({tags:1})
      - db.blog.createIndex({"comments.author":1})
    - the number specifies the direction of the index (1: ascending; -1: descending)
  - Additional properties can be specified for an index
    - **Sparseness**, **uniqueness**, **background**, ..
- Most MongoDB indexes are organized as **B-Tree** structure

Single field Index



Compound Index

- Compound Index is a single index structure that holds references to multiple fields within a collection
- The order of field in a compound index is very important
  - The indexes are sorted by the value of the first field, then second, third…
  - It supports queries like
    - db.users.find({userid: “ca2”, score: {$gt:30} })
    - db.users.find({userid: “ca2”})
  - But not queries like
    - db.users.find({score: 75})



Outline

- MongoDB indexing
- Mongoose

Database Drivers

- All database management systems work like a “server” application
  - Running on a host and waiting for connections from clients
    - Simple command line shell client
    - GUI shell client
    - Program based client
  - There are different protocols db server used to communicate with their clients
- All database management systems provide language based drivers to allow developers to write client in various languages
  - Open/close connection to database
  - Translate between language specific construct (functions, methods) and database queries
  - Translate between language specific data types and database defined data types
  - And others
- MongoDB provides many native drivers:
  - https://docs.mongodb.com/ecosystem/drivers/

Higher level module/package

- The native db drivers provide basic supports for client side programming
  - Powerful, flexible
  - But usually not easy to use
- Higher level modules usually provide more convenient ways to communicate with db servers
- Mongooes is the node.js module build on top of basic mongodb node.js driver
  - Data structure to match collection “schema”
  - Validation mechanism
  - Connection management
  - Etc.

Mongooes

- All database operations are considered as potentially blocking and should be implemented using event-driven programming style
  - Start an operation
  - Register a *callback* function to indicate what we want to do when the operation completes
  - Continue processing other parts of the program

Mongooes Basic Concepts

- Schema
  - Schema is an *abstract* data structure defines the shape of the documents in a collection
  - Each name/value pair is a path
- Model
  - Model is a compiled version of schema, model is the schema binded with a collection
- Document
  - Document is an instance of Model, mapped to the actual document in a collection

Example of Schema, Model and Document

- If we have a collection "**movies**" with the example document

```
{ "_id" : 1.0,
   "Title" : "Sense and Sensibility",
    "Year" : 1995.0,
     "Genres" : ["Comedy",  "Drama", "Romance"]
}
```

- We can define a schema as

```
var movieSchema = new Schema({
  Title: String,
   Year: Number,
  Genres: [String]
})
```

- The corresponding model
- And document

```
var Movie = mongooes.model('Movie',
movieSchema, 'movies')

var aMovie = new Movie({
    title="Ride With the Devil"})
```

Queries

- All Mongodb queries run on a model
  - This includes **`find`**, **`udpate`**, **`aggregate`**, and so on
  - The syntax is very similar to shell command query
  - A callback function needs to be specified if we want to do something with the query result.

Call back function

```
Movie.find({}, function(err,movies){
    if (err){
     console.log("Query error!")
    }else{
       console.log(movies)
    }
  }
)
```

```
Movie
.find({Year: 1996})
.select({Title:1,Year:1})
.exec(function(err,movies){
  if (err){
    console.log("Query error!")
  }else{
    console.log("Movies in year 1996:")
    console.log(movies)
  }
 }
)
```

```
var newMovie = new Movie(
{ MovieID: 292,
  Title: "Outbreak",
  Year: 1995,
  Genres: ['Action','Drama','Sci-Fi','Thriller']}
)
newMovie.save()
```

Static Methods

- If we know that we are going to run certain queries often on some collection, we can implement those queries either as static methods or as instance methods
- Static methods is defined on the Model (collection), any standard query/aggregation can be implemented as static method
- Static methods increase the reusability and modularity of database related code

```
movieSchema.statics.findByYear = function(year, callback){
    return this
            .find({Year: year})
            .select({Title:1,Year:1})
            .exec(callback)
}
var Movie = mongoose.model('Movie', movieSchema, 'movies')
Movie.findByYear(1995, function(err,movies){
        if (err){
         console.log("Query error!")
        }else{
           console.log("Movies in year 1995:")
           console.log(movies)
        }
})
```

this keyword refers to the current model that calls the method

We call the method on **Movie** model, **this** refers to Movie model, which represent the movies collection.

The call becomes:

```
Movie
.find(…)
.select(…)
.exec(callback)
```

A callback function is always supplied when we make the call, instead of predefined.

Instance Methods

- Instance methods is defined on document instance.
- It is often used to create queries based on a given document
  - E.g. Find all movies released in the given movie

```
schema.methods.findSimilarYear = function(cb) {
  return this.model('Movie').find({ Year: this.Year }, callback);
};

var newMovie = new Movie(
{MovieID: 292,
 Title: "Outbreak",
 Year: 1995,
 Genres: ['Action','Drama','Sci-Fi','Thriller']}
)
newMovie.findSimilarYear(function(err,movies){
  if (err){
    console.log("Query error!")
  }else{
    console.log("The movies released in the same year as " + newMovie.Title + " are:")
    console.log(movies)
 }
}
)
```

this keyword refers to the current document that calls the method, we can use it to access the model and individual property of the document

Instance methods are called on document instance

Database Connection

- Opening and closing connection to database is time consuming
- The best practice is to let all requests share a pool of connections and only close them when application shuts down
- Mongoose manages connection pool
- No application level open or close is required

```
var mongoose = require('mongoose')

mongoose.connect('mongodb://localhost/comp5347', function (err) {
  if (!err)
      console.log('mongodb connected')
})
```

Connection string or database URI

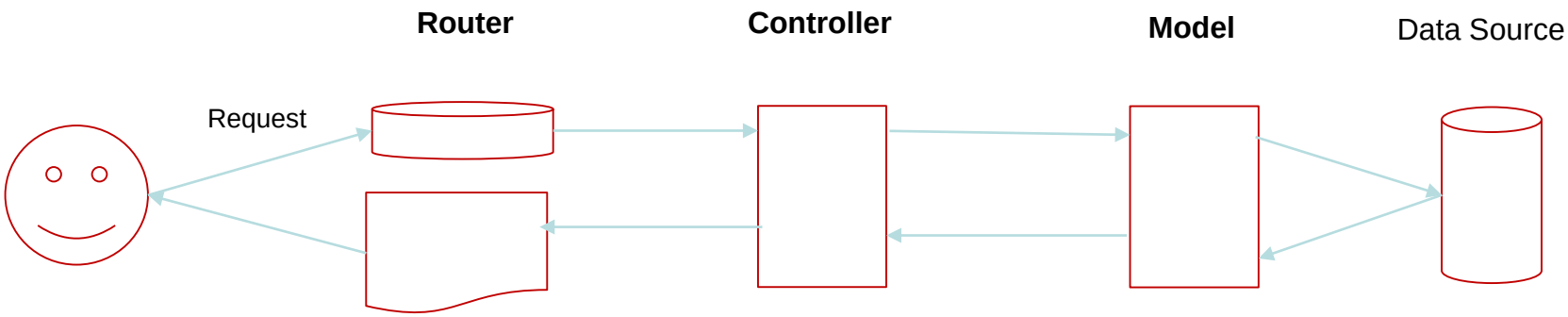
Full MVC Architecture

1. Request comes into application

2. Request gets routed to controller

3. Controller may send request to model for data

4. Model may need to talk to a data source (database) to manipulate data



8. View generate HTTP response and sends back to client

7. Controller pass data to view

6. Model responds to controller

5. Data source sends result back to model

Data base related code should be put in model layer
Controller should not have knowledge about the actual database
Modularity allows easy switching between technologies
e.g. different view templates, different database management systems

Admin

- We have two hour lab this week
  - Basic MongoDB query
  - Mongoose and Integration with Express
- Tuesday labs will be from 7-9pm in your allocated room
- Wednesday labs
  - We made a mistake about lab availability last week
  - There is no lab available for two hour block before 6pm in any day
  - Wednesday labs will be split in two one hour labs
    - Tuesday 7-8pm in lab 117 (part one)
    - Wednesday in your allocated time and room (part two)

Resources

- Haviv, Amos Q, MEAN Web Development
  - E-book, accessible from USYD library
  - Chapter 5
- Mongooes online documents:
  - Guide
    - http://mongoosejs.com/docs/guide.html