plan

last lecture
› databases as systems
› storing lots of stuff but only selecting a tiny fraction
› ACID vs. eventual consistency; layered architecture; schemas

this lecture
› databases as programming technology
› MongoDB, mongoose and node drivers

next lecture
› how to design a database schema
3 ways to store data
Showtimes for The Maze Runner

- **Showcase Cinema de Lux Legacy Place** - Map
  - **Starting Times:**
    - 3:50pm, 4:40, 6:35, 7:20, 9:05, 10:05

- **Regal Fenway Stadium 13 & RPX** - Map
  - **Starting Times:**
    - 4:05pm, 7:00, 7:20, 10:00, 10:30

- **Embassy Cinema** - Map
  - **Starting Times:**
    - 4:05pm, 7:15

*All times are in ET*

---

**The Maze Runner**

- **Rating:** 7.7/10 - IMDb
- **Rating:** 63% - Rotten Tomatoes

*Synopsis*

When Thomas wakes up trapped in a massive maze with a group of other boys, he has no memory of the outside world other than strange dreams about a mysterious organization known as W.C.K.D. Only by piecing together fragments of his past with clues he discovers in the maze can Thomas hope to uncover … More
object-oriented heap

App
- showings
- movies
- theaters

Map
- root

redundancy
- Map

classes
- showings
- name
- location

collections
- Map

references
- Map

no primitives
- Time
  - 7:00pm

Map
- Showing
  - screen
  - time
  - movie
  - theater

Map
- Movie
  - title
  - rating
  - genre

String
- "Fury"
- "Legacy Place"
- "Dedham"
### Relational Database

#### Showings

<table>
<thead>
<tr>
<th>id</th>
<th>theater</th>
<th>screen</th>
<th>movie</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>7:00pm</td>
</tr>
</tbody>
</table>

#### Theaters

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>“Legacy Place”</td>
<td>“Dedham”</td>
</tr>
</tbody>
</table>

#### Movies

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>rating</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>“Fury”</td>
<td>“R”</td>
<td>“action”</td>
</tr>
<tr>
<td>primary key</td>
<td>Showings</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>_id</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>title</td>
<td>&quot;Fury&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>&quot;7:00pm&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rating</td>
<td>&quot;R&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>genre</td>
<td>&quot;action&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>theater</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>name</td>
<td>&quot;Legacy Place&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>location</td>
<td>&quot;Dedham&quot;</td>
<td></td>
</tr>
</tbody>
</table>

**document store**

**collection**

**document**

**nested structure**

**primitives**
<table>
<thead>
<tr>
<th></th>
<th>what structures?</th>
<th>how do you query?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>object</strong></td>
<td>objects, classes (incl. arrays)</td>
<td>arbitrary navigations compiled or interpreted</td>
</tr>
<tr>
<td><strong>document</strong></td>
<td>documents, collections</td>
<td>structural match then loops, map, etc</td>
</tr>
<tr>
<td><strong>relational</strong></td>
<td>tuples (rows), tables</td>
<td>declarative queries automatically optimized</td>
</tr>
<tr>
<td></td>
<td>good</td>
<td>bad</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td><strong>object</strong></td>
<td>easily make arbitrary structures; referencing abstracted</td>
<td>not persistent! need a “root”; no declarative queries</td>
</tr>
<tr>
<td><strong>document</strong></td>
<td>heterogeneous collections</td>
<td>no joins</td>
</tr>
<tr>
<td><strong>relational</strong></td>
<td>query language; transactions; fault tolerance</td>
<td>no heterogeneity; no subtypes; exposed references</td>
</tr>
</tbody>
</table>
how we got here
oracle: the RDBMS leader

Stock value versus S&P 500
Current market capitalization: about $175B
projected growth of “NoSQL”
interlude: queries vs the whole DB
“Downloading the Internet” by Mitchell & Webb, a British comedy duo
the MongoDB API
three essential notions

“connection”
› database runs as a server; client accesses via connection
› basis of authentication (can use SSL, eg)
› connection has a thread, preserves order of operations

“collection”
› group of related documents, not necessarily homogeneous
› target of query and update
› indexes associated with collections

“document”
› unit of storage in a collection
› queries can look at document structure (cf. key-value store)
› all operations are atomic at document level
**mongoDB operations**

```python
db = client.connect("mongodb://localhost:27017/moviedb")
› connect to database server and return connection

movies = db.collection("movies")
› get collection; create if necessary

movies.insert({title: "Fury", theater: 123, time: "7pm"})
› insert a document into a collection

cursor = movies.find({title: "Fury"})
› find documents matching query, and return cursor

cursor = movies.update({.......})
› replace a document
```
queries

movies.find({})
▷ find all movies

movies.find({title: "Fury"})
▷ find movies with title "Fury"

movies.find({title: {$in: ["Fury", "Lucy"]}})
▷ find movies with a title in the given array

movies.find({title: "Fury", time: "7pm"})
▷ find movies with title "Fury" showing at 7pm

movies.find({$or: [{title: "Fury"}, {time: "7pm"}]})
▷ find movies with title "Fury" or showing at 7pm

other features:
$and
$gt, $lt
nested match
array matches
projections & sorts

movies.find({time: "7pm"}, {title: 1, theater: 1})
› find movies at 7pm, but return only title and theater

movies.find({time: "7pm"}, {time: 0})
› find movies at 7pm, but drop time from results

movies.find({title: "Fury"}).sort({time: 1})
› find movies with title "Fury", sort by time

performance issues?
why bother to project?
what impact will sort have?
the node.js driver API

an asynchronous API
› all non-instantaneous ops take callback instead of returning

examples
› not `movies.insert({title: "Fury"...}); rest();`
› but `movies.insert({...}, function (err, result) {rest();})`

sequencing non trivial
› can be “callback hell”
› more on this when we see sample code later
two ways to structure data in mongo
one document in the collection Movies
relational

one document in the collection Movies

one document in the collection Theaters
pros and cons?

**embedded**

```json
{
  title: "Fury",
  time: "7:00pm",
  theater: {
    name: "West Newton Cinema",
    location: "Newton"
  }
}
```

one document in the collection *Movies*

**relational**

```json
{
  title: "Fury",
  time: "7:00pm",
  theater: 1
}
```

one document in the collection *Movies*

```
{
  _id: 1,
  name: "West Newton Cinema",
  location: "Newton"
}
```

one document in the collection *Theaters*

time to read?
time to write?
space required?
consistency?
ease of coding?
demo & examples
mongoose
what is mongoose?

an “object-document mapper”
› like an “object-relational mapper” for RDBs
› but documents closer to objects than tuples

what mongoose provides
› schema declarations (good for validation)
› model classes (can extend with methods; insert with save)
› helpful API, especially populate method
var theaterSchema = mongoose.Schema({
    _id: Number,
    name: String,
    location: String
});

var movieSchema = mongoose.Schema({
    title: String,
    time: String,
    theater: {type: Number, ref: 'Theater'}
});

create schema

Theater = mongoose.model('Theater', theaterSchema);
Movie = mongoose.model('Movie', movieSchema);

create model classes

var data = {title: "Fury", time: "7:00pm", theater: 2};
var m = new Movie(data);
m.save();

create object and save to database
what populate does

```javascript
Movie.find({time: 'Fury'}, function (err, movies) {
    Movie.populate(movies, {path: "theater"}, function (err, result) {
        // result is an array of objects such as
        {
            title: "Fury",
            time: "7:00pm",
            theater: {
                name: "West Newton Cinema",
                location: "Newton"
            }
        }
    })
})
})
```

```javascript
var theaterSchema = mongoose.Schema({
    _id: Number,
    name: String,
    location: String
});

var movieSchema = mongoose.Schema({
    title: String,
    time: String,
    theater: {type: Number, ref: 'Theater'}
});
```
all doors open
A Time to Talk

When a friend calls to me from the road
And slows his horse to a meaning walk,
I don’t stand still and look around
On all the hills I haven’t hoed,
And shout from where I am, ‘What is it?’
No, not as there is a time to talk.
I thrust my hoe in the mellow ground,
Blade-end up and five feet tall,
And plod: I go up to the stone wall
For a friendly visit.

Robert Frost
1920
some questions to ponder

When did someone find a time to talk to you?
How can we all find more time to talk?
Are there 6170 projects that might help?