6.1040 · software studio · fall 2023

data design

Daniel Jackson & Arvind Satyanarayan

today's learning objectives

get fuller grasp of relational state declarations
be aware of different database models, esp NoSQL
know about classical data modeling
design concept data models
implementation considerations

database models

Movies playing near Back Bay East, Boston, MA

All Genres

Crazy Rich Asians
Drama/Come...

White Boy Rick
Drama/Myste...

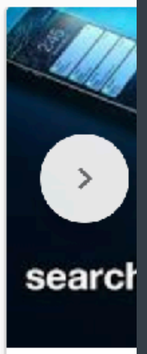
Peppermint
Drama/Thrille...

Fahrenheit 11/9
Political cine...

Life Itself
Drama/Roma...

The Meg
Thriller/Fanta...

Little Women
Drama/Family



Searching
Drama/Th

Showtimes for Crazy Rich Asians

All times are in ET

Today | Tomorrow | Tue, Oct 2 | Wed, Oct 3

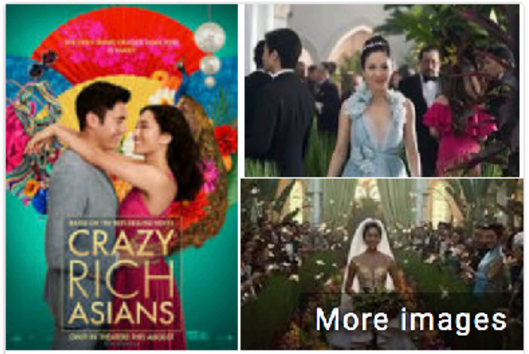
All times | Morning | Afternoon | Evening | Night

AMC Loews Boston Common 19 - Map
Standard 4:40pm 7:30pm 10:20pm

Regal Fenway Stadium 13 & RPX - Map
Standard 4:10pm 7:20pm 10:30pm

ShowPlace ICON at Seaport with ICON-X - Map
Standard 4:45pm 6:10pm 7:45pm 9:10pm 10:30pm

More showtimes

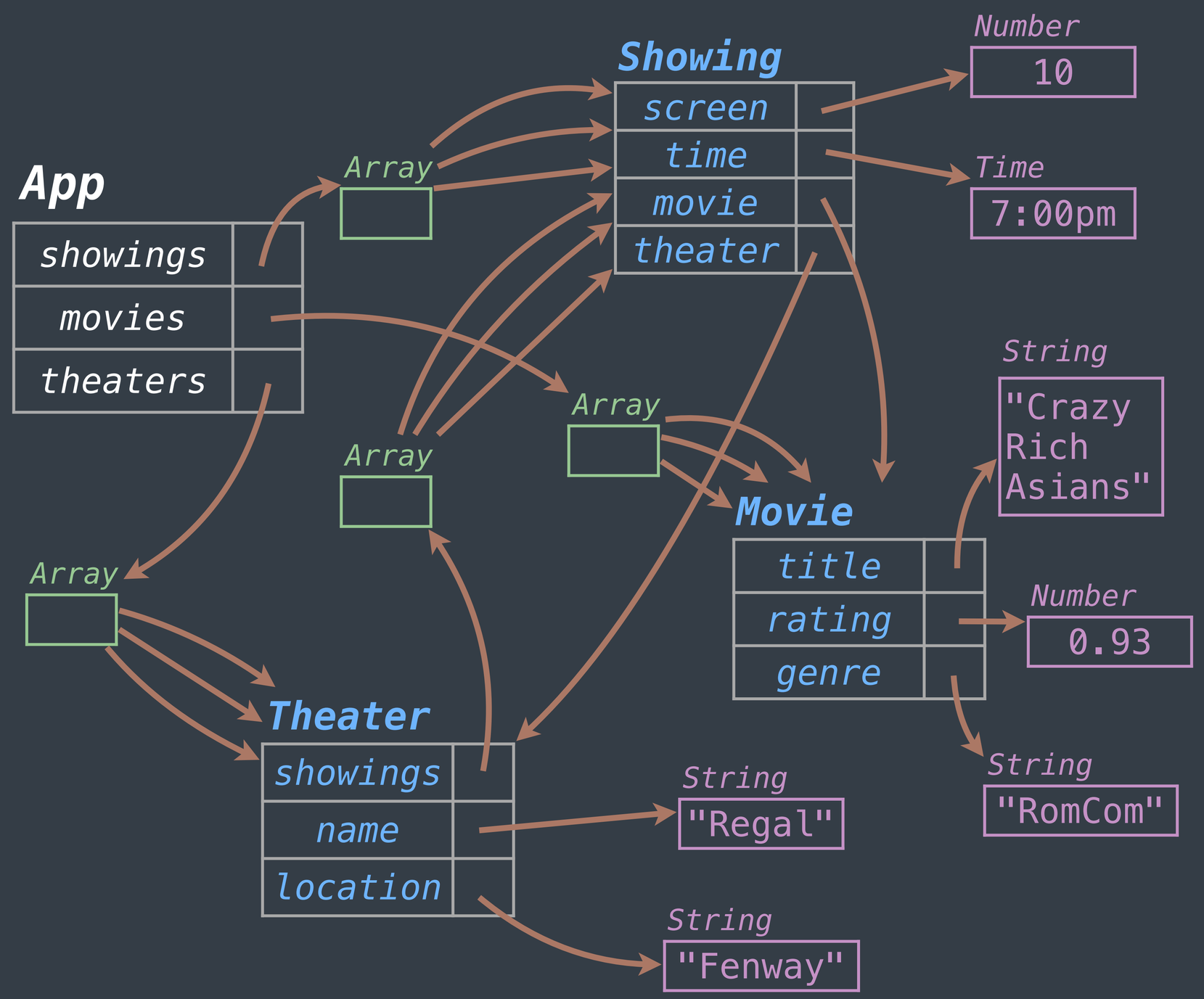


Crazy Rich Asians

PG-13 2018 · Drama/Comedy-drama · 2h 1m

7.5/10	93%	74%
IMDb	Rotten Tomatoes	Metacritic

93% liked this movie
Google users



Object Model

Application root references collections of class instances that describe primitive data.

- ✓ Quick to prototype.
- ✓ Easy to experiment with arbitrary data structures.
- ✘ Refactoring is difficult.
- ✘ No advanced querying: can only iterate over collections, follow references.

Relational Model (SQL)

Showings

id	theater	screen	movie	time
1	3	5	2	7:00pm
..	...	...	...	

Theaters

id	name	location
...	...	...
3	"Regal"	"Fenway"

Movies

id	title	rating	genre
...	...	...	
2	“Crazy Rich	“PG-13”	“RomCom”

Relations (aka tables) of attributes (aka columns) and tuples (aka rows).

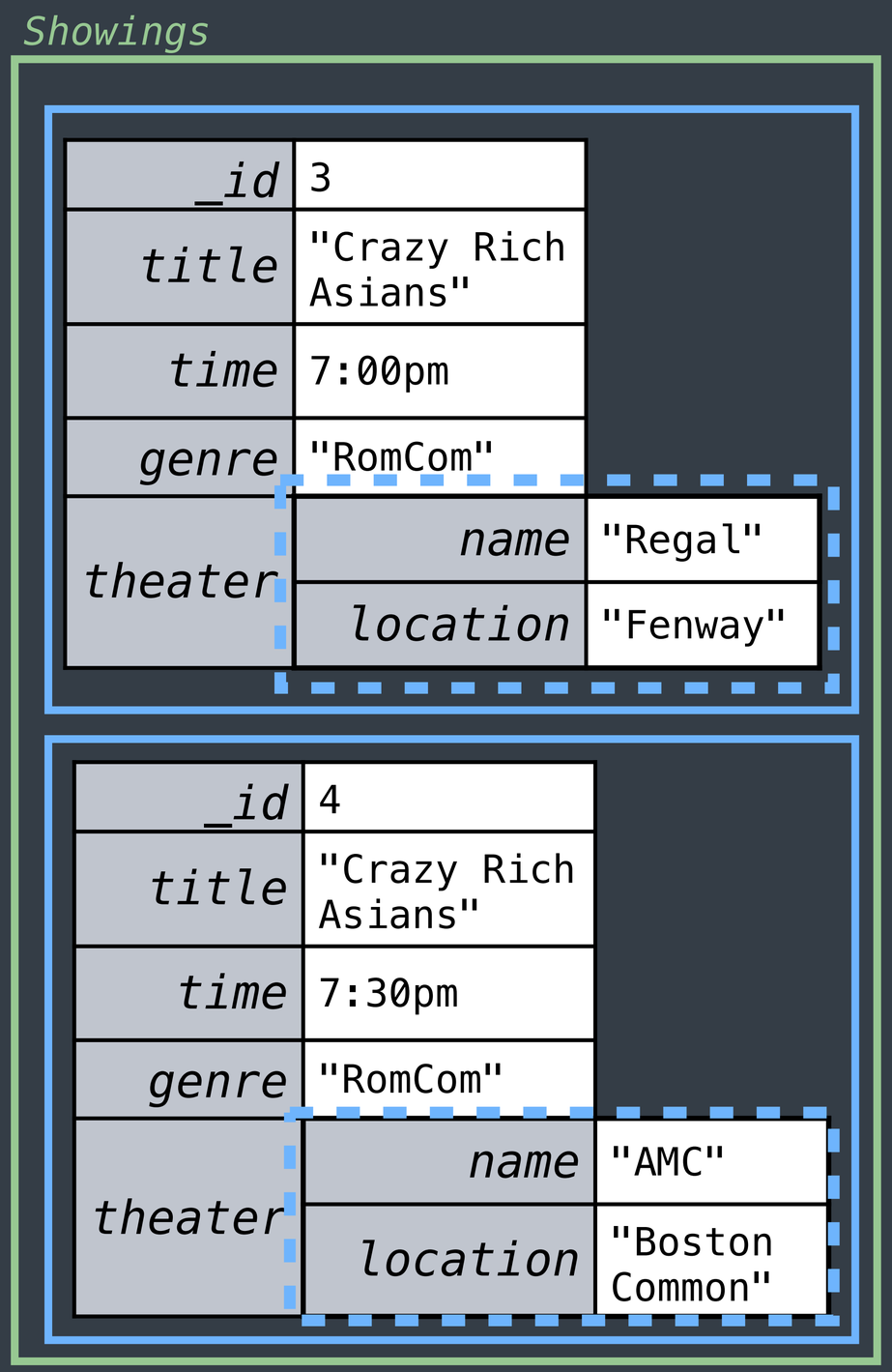
✓ Standardized query language (SQL) regardless of backend engine (MySQL, PostgreSQL, SQLite, ...).

✓ Relational theory encourages better separation of concerns (called "normalization").

✓ Over 40 years of research into performance and robustness (indexing, transactions, integrity, ...).

✘ (Until recently) did not offer JSON types.

✘ (Until recently) difficult to scale horizontally. Vertical scaling (i.e., make server more powerful) was the easiest option.



NoSQL

"Not Only SQL"

Collections of *nested* documents (or graph structures).

✓ Quick to prototype (documents stored as JSON).

✓ Easy to experiment with arbitrary data structures.

✓ Pattern matching by document structure.

✓ *Horizontal* performance (i.e., many less-powerful servers, rather than a single very powerful one).

✘ No standardized query language.

✘ Embedded documents = easier to make poor design decisions.

✘ (Until recently) no references between collections: complexity of lookups occurs at the application level.

Mongo:
a NoSQL database

MongoDB CRUD Operations

```
db.showings.insertOne({})
db.showings.insertMany([{}, {}, ...])
```

```
{
  "_id": ObjectId(),
  "title": "Crazy Rich Asians",
  "genre": "RomCom",
  "showtime": Date("2022-10-07 15:30"),
  "theater": {
    "name": "AMC",
    "location": "Boston Common"
  }
}
```

Documents are JSON-like structures ("BSON") that offers additional data types like `Date`, `RegExp`, or binary data.

Every document must have an `_id`, and it must be unique within the collection.

`_id` is generated automatically by MongoDB via `ObjectId` (you can override it, but you really shouldn't!).

MongoDB CRUD Operations

```
db.showings.insertOne({})
db.showings.insertMany([{}, {}, ...])

db.showings.findOne({})
db.showings.find({})

{"title": "Crazy Rich Asians"}

{
  "theater": {
    "name": "AMC"
  }
}

{
  "title": "Crazy Rich Asians",
  "theater.name": "AMC"
}
```

```
{"$or": [
  {"title": "Crazy Rich Asians"},
  {"theater.name": "AMC"}
]}

{"theater.name": {
  "$in": ["AMC", "Regal"]
}}

{"showtime": {
  "$gte": Date("2022-10-07")
  "$lte": Date("2022-10-10")
}}
```

MongoDB CRUD Operations

```
db.showings.insertOne({})
db.showings.insertMany([{}, {}, ...])

db.showings.findOne({})
db.showings.find({})

db.showings.updateOne({}, {"$set": ...})
db.showings.updateMany({}, {"$set": ...})
db.showings.replaceOne({}, {})

db.showings.deleteOne({})
db.showings.deleteMany({})
db.showings.drop()
```

Multiple Collections vs. Embedded Documents

```
db.theaters.insertOne({
  "_id": 1, "name": "AMC", ...
})

db.movies.insertOne({
  "_id": 3,
  "title": "Crazy Rich Asians",
  ...
})

db.showings.insertOne({
  "_id": 5, "theater": 1, "movie": 3,
  "showtime": Date()
})
```

```
db.movies.insertOne({
  "_id": 3,
  "title": "Crazy Rich Asians",
  "showings": [
    {
      "theater": {"name": "AMC", ...},
      "showtime": Date()
    }
  ]
})
```

Multiple Collections vs. Embedded Documents

✓ More flexible querying (e.g., sorting results)

✘ Separate collections require more work: you have to manually join things together.

```
amc = db.theaters.find({"name": "AMC"})
amc_ids = amc.map(t => t._id)
movies = db.movies.find({
  "theater": "$in": amc_ids
})
```

✘ Limited to insertion order

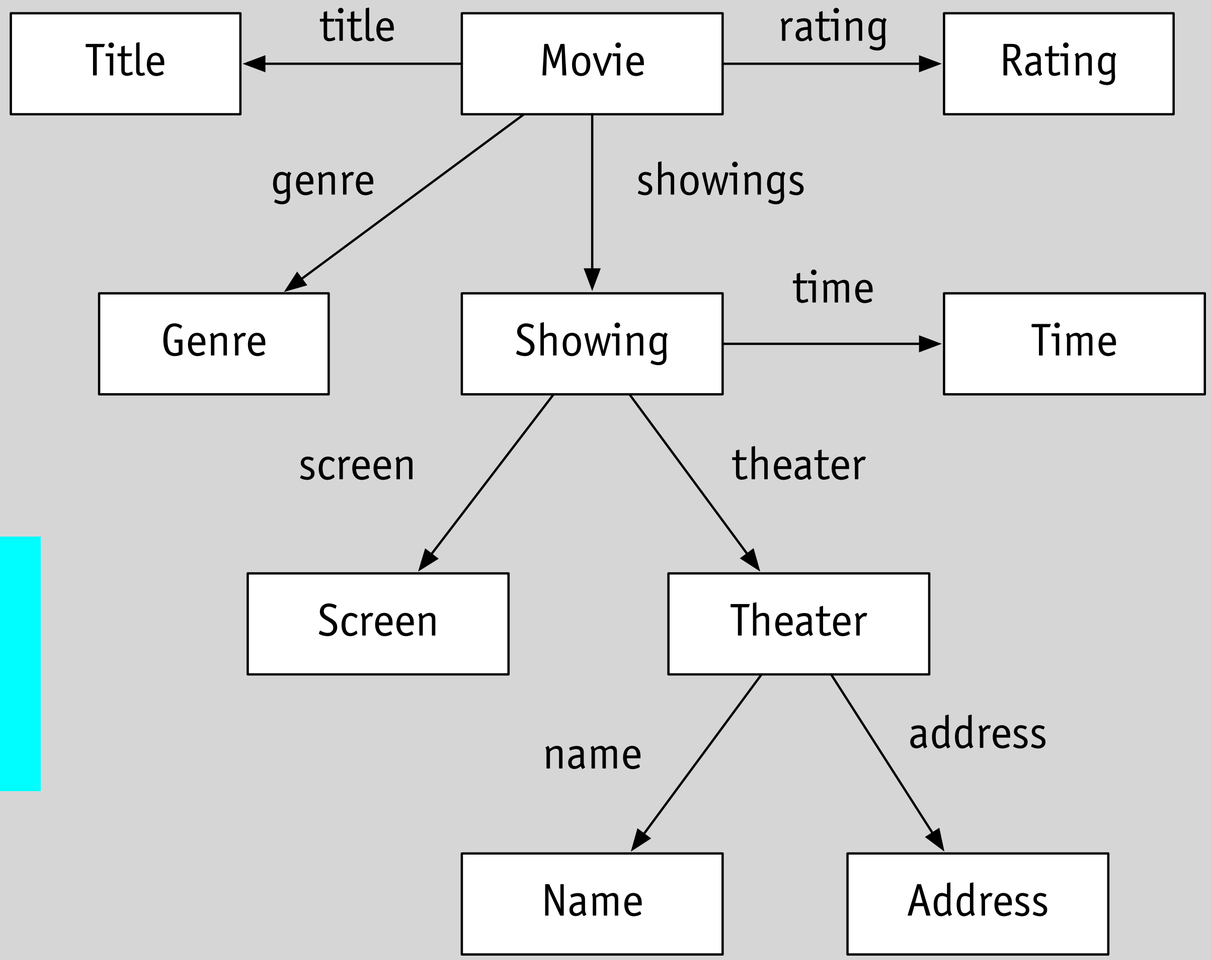
✘ Each document (including all embedded documents, arrays, etc) cannot be larger than 16MB.

```
{"theater.name": "AMC"}
```

1. How many embedded objects do you have? One? A few? Many?
2. Does the embedded document relate to any other collections?
3. How often will you need the embedded document *without* the parent, or vice versa?

designing a database: the classic approach

step 1: identify entities and relationships



big idea:
boxes are **sets**,
arrows are **relations**

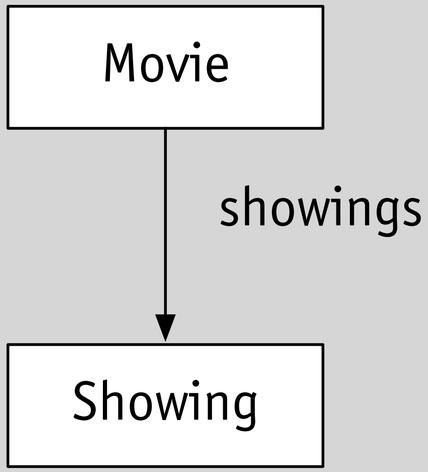
why is this good?
simple semantics
rep-independent

relations + diagram:
show possible
navigations

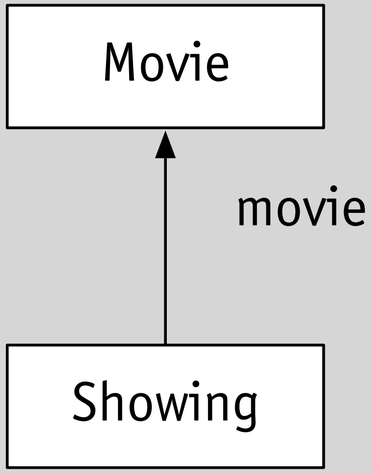
relations = predicates
(Barbie, 4.1) in rating
IsRated (Barbie, 4.1)

a common confusion: arrow direction

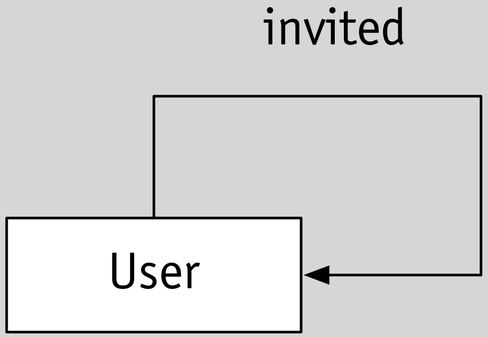
arrow direction is **NOT** navigation or containment



can **switch direction** so long as relation is interpreted consistently

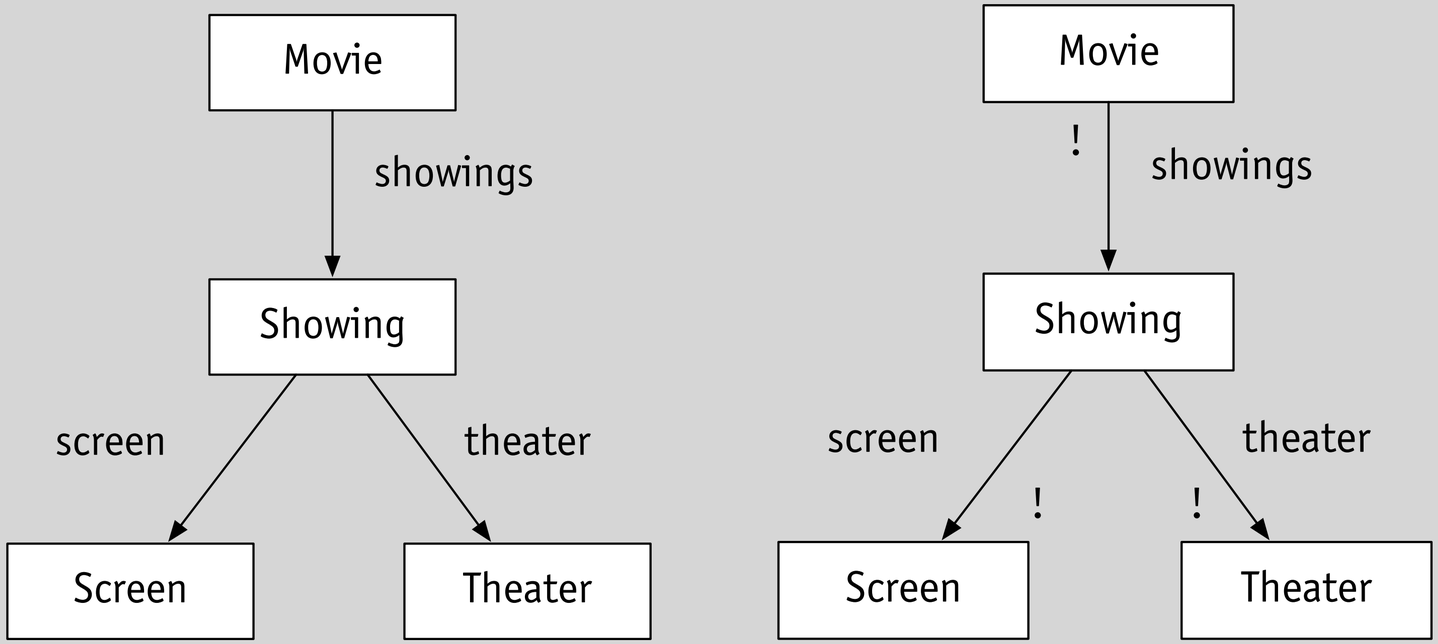


really matters for **homogeneous** relations



(alice, bob) in invited:
alice invited bob
or bob invited alice?

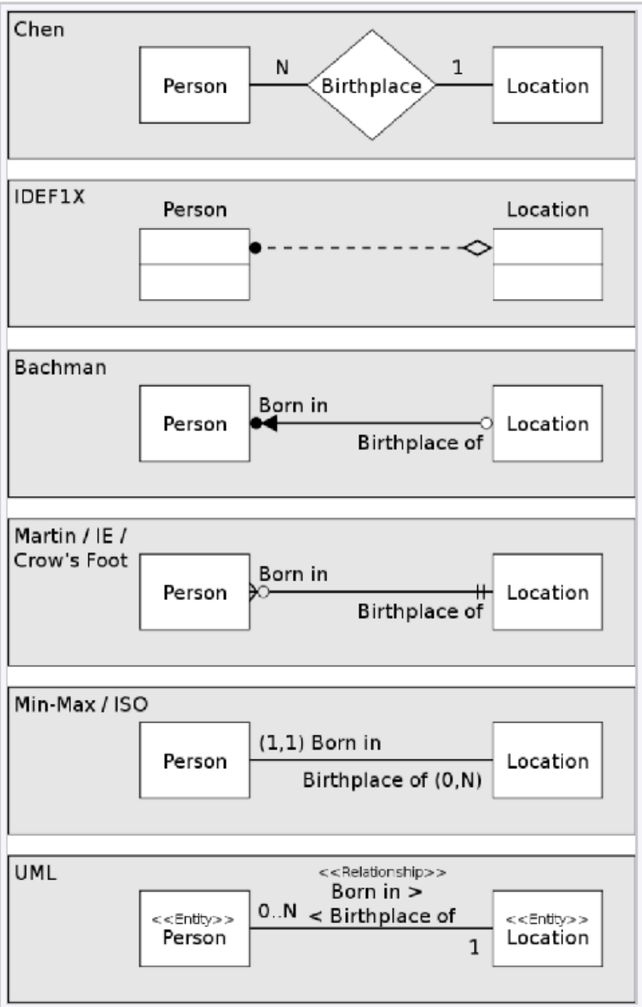
step 2: adding multiplicities



multiplicities:
= 1: **one, !**
<= 1: **lone, opt, ?**
>= 1: **some, +**
>= 0: **set, *default***

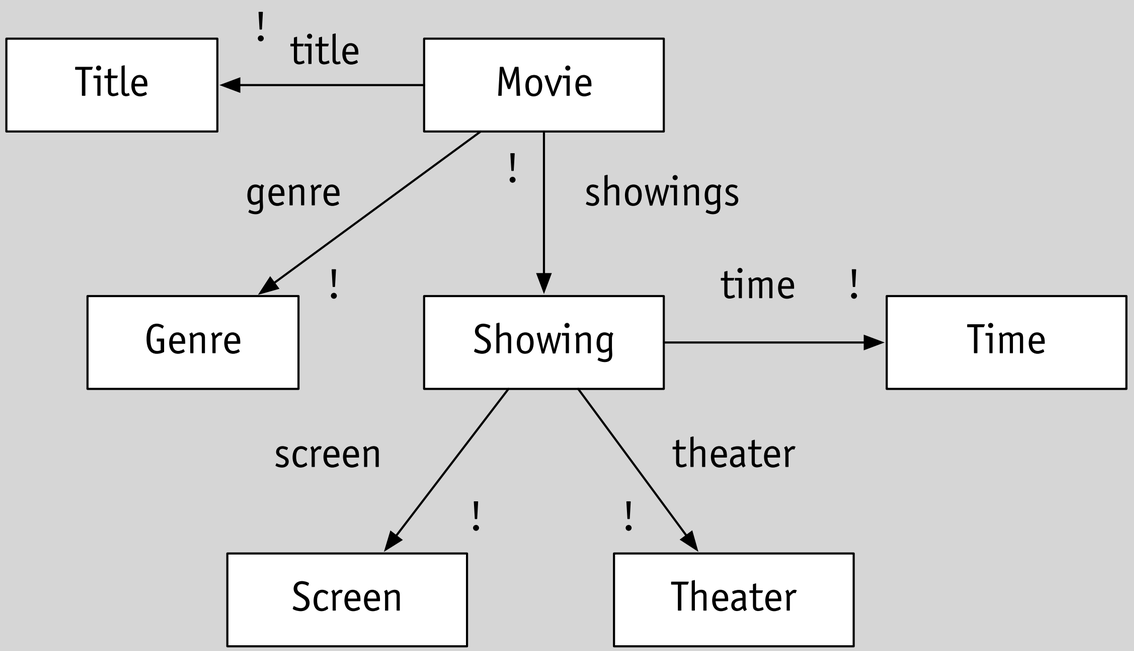
tells you how many
on that end of the arrow:
one movie per showing
any showings per movie

many different notations for abstract data models



Various methods of representing the same one to many relationship. In each case, the diagram shows the relationship between a person and a place of birth: each person must have been born at one, and only one, location, but each location may have had zero or more people born at it.

step 3: transform to a database schema (relational)



constraint:
no **set-valued** columns

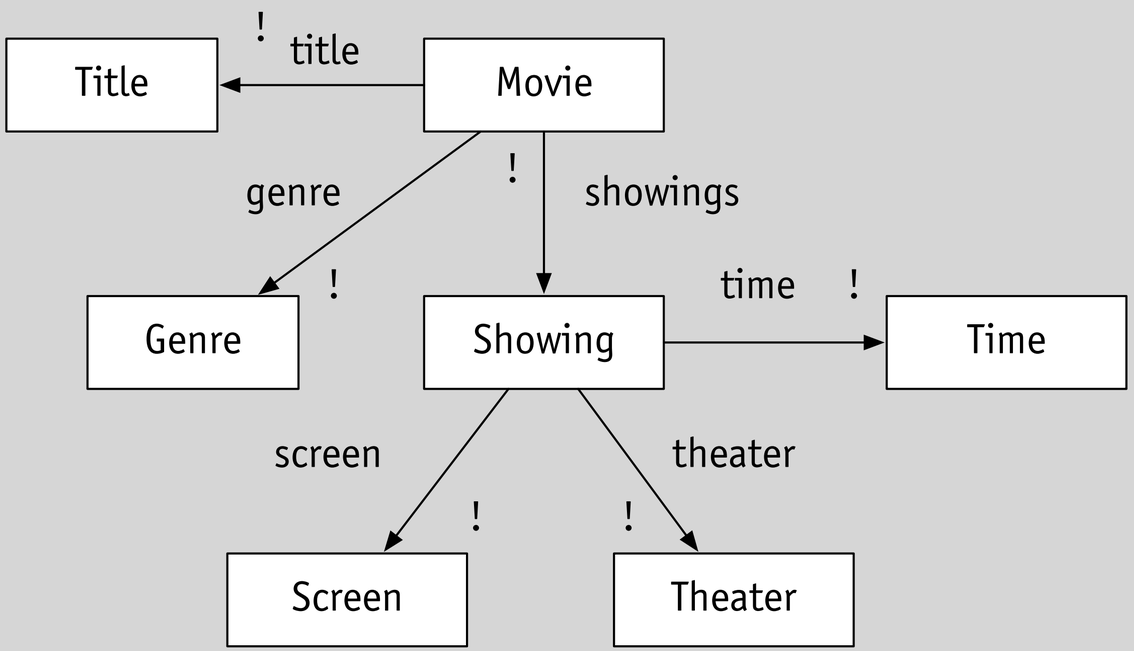
movies

id	*title*	*genre*
1	Crazy Rich Asians	RomCom
2	Barbie	Fantasy

showings

id	*movie*	*screen*	*theater*	*time*
1	1	2	35	3:00pm
2	1	1	23	7:00pm

step 3: transform to a database schema (object oriented)

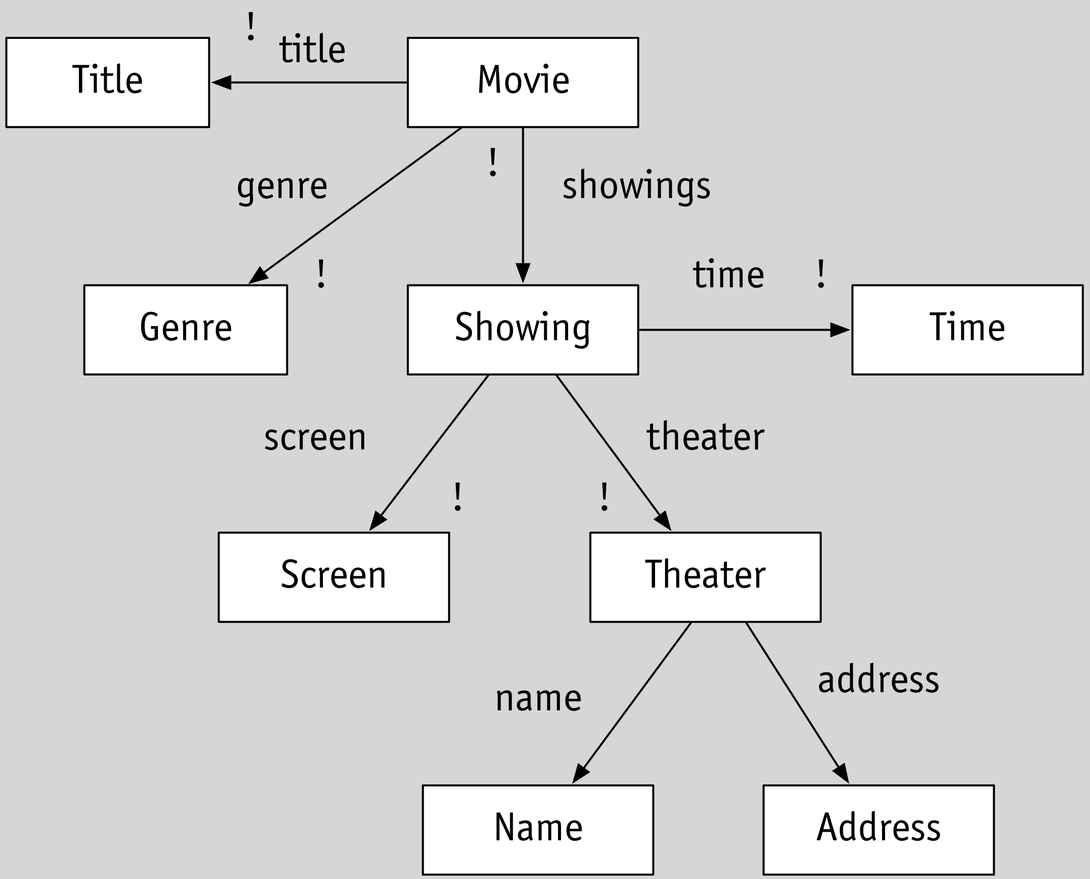


constraint:
queries follow **fields**

```
class Movie {
   Title title;
   Genre genre;
   Map [Date, Set [Showing]] showings;
 }
```

```
class Showing {
   Screen screen;
   Theater theater;
    Date time;
}
```

step 3: transform to a database schema (collection database)



constraint:
embedded objects
preferably **immutable**

showings

id	1	
title	Crazy Rich Asians	
time	7:00pm	
genre	"RomCom"	
screen	2	
theater	*name*	"AMC"
	address	"401 Park Dr"

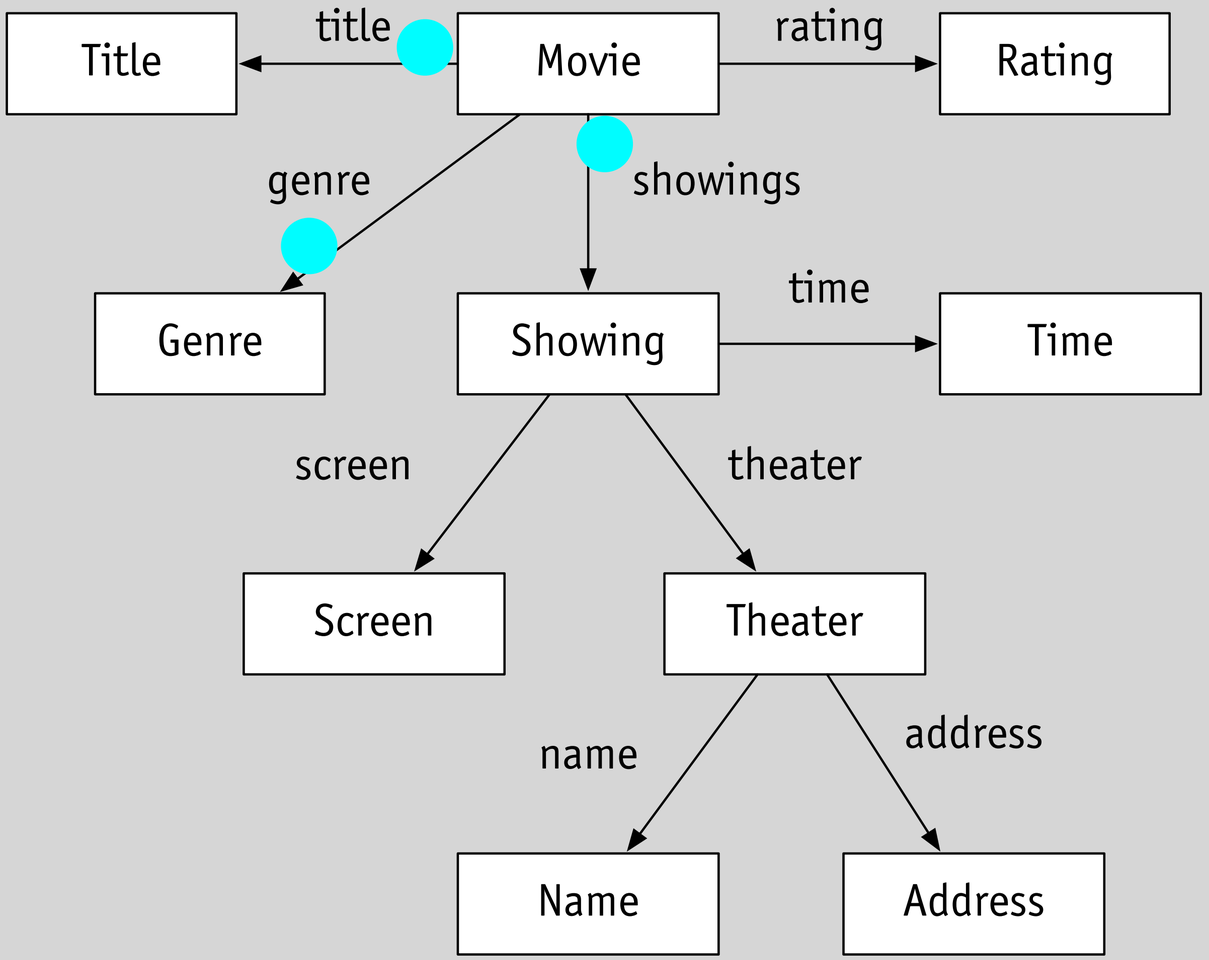
some considerations in classic schema design

what's even possible to represent?
in relational database, fields must be scalars

what's the cost of queries?
relational joins can be costly, but mitigated by indexes
in Mongo etc, joins are not as efficient as in SQL
in OOP, beware of queries that require search

what's the cost of updates?
may need to lock the table/document/object, preventing reads
in Mongo, embedded objects must be kept consistent

multiplicities are important!



discuss: what are these multiplicities?

and what's the **programming impact** of getting them wrong?

Barbie / Genres

Comedy	Children's film
Adventure	Drama
Romance	Fantasy
Romantic comedy	Narrative

movie with more than one genre

Two chilling, bold, mesmerizing, futuristic detective thrillers.

Ridley Scott's visually stunning *Blade Runner* set a new benchmark for science fiction upon its release in 1982. In 2017, director Denis Villenueve did the unthinkable with *Blade Runner 2049*, crafting an atmospheric and riveting sequel that is not only worthy of the original, but may actually surpass it. See them back to back in this special double-feature and decide for yourself!

a double feature of two movies with one title

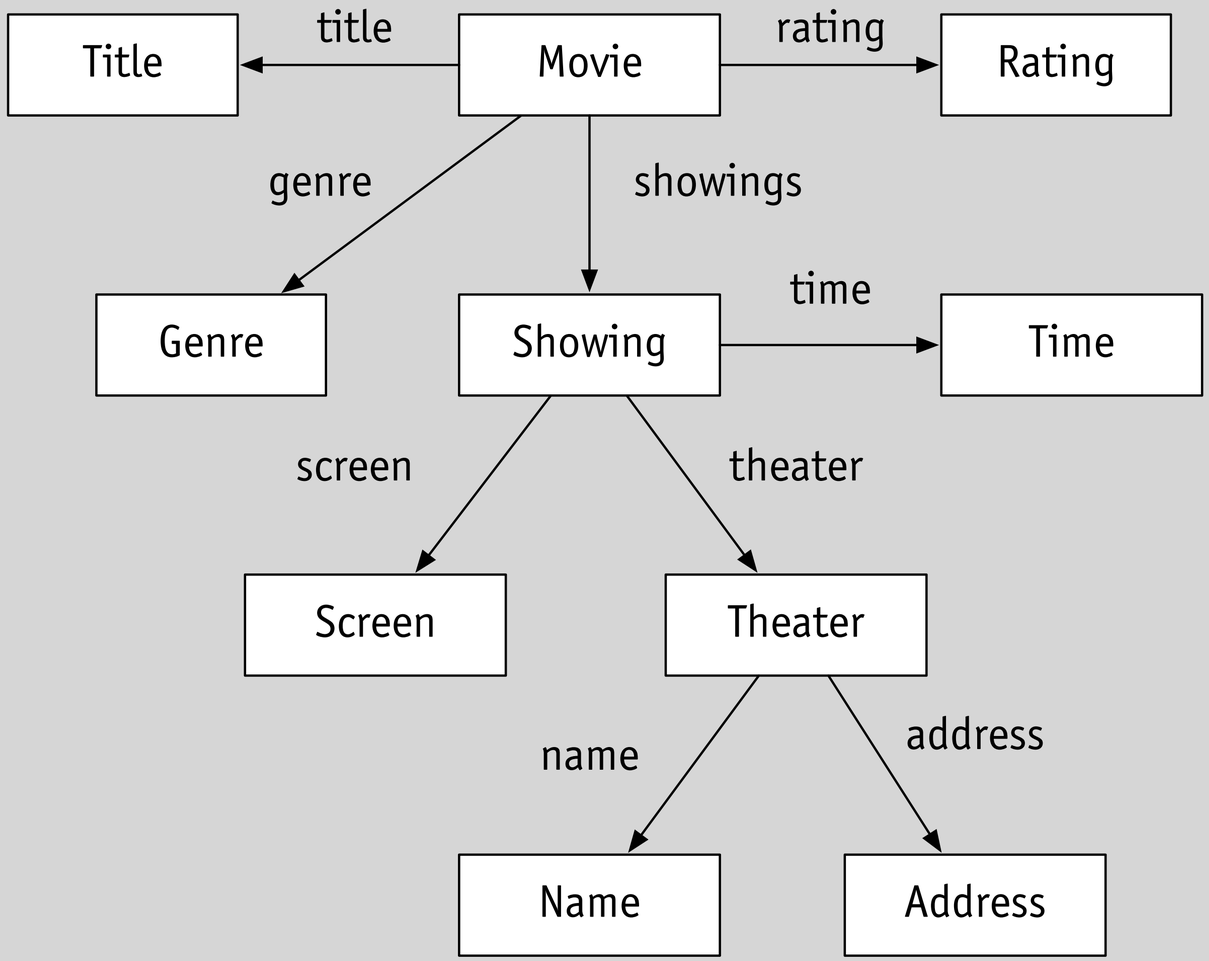
10 Films With The Same Title That Are Not The Same Movie

- 'Night Moves' (1975) and (2013) ...
- 'Missing' (1982) and (2023) ...
- 'Twilight' (1998) and (2008) ...
- 'Possession' (1981) and (2002) ...
- 'Rush' (1991) and (2013) ...
- 'Heat' (1986) and (1995) ...
- 'Kicking and Screaming' (1995) and (2005) ...
- 'The Gift' (2000) and (2015)

More items... • Jan 5, 2023

movies with the same title

problems with the classic approach



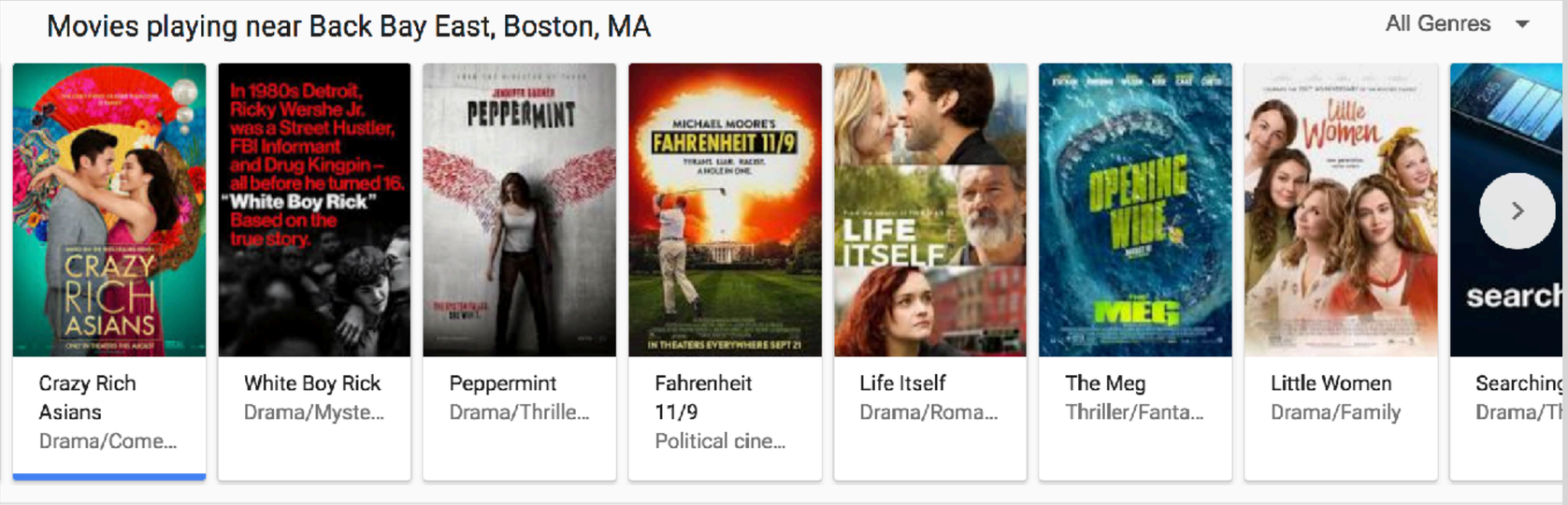
what's in the model?
how do you decide
what data to include?

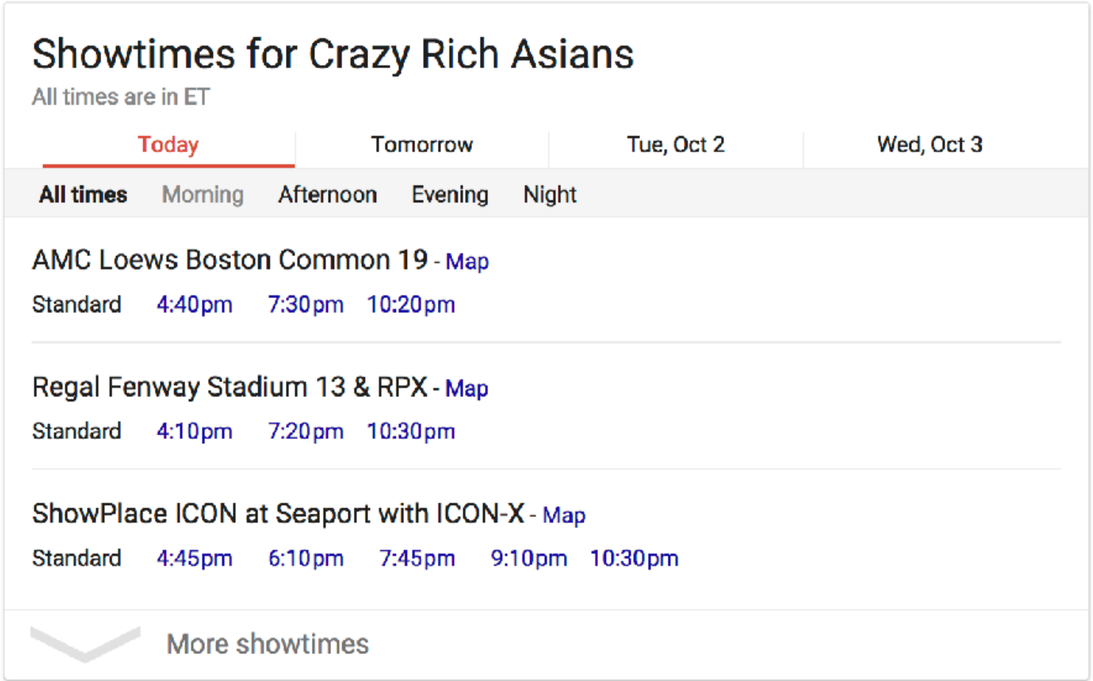
modularity?
how do you make a
modular app?

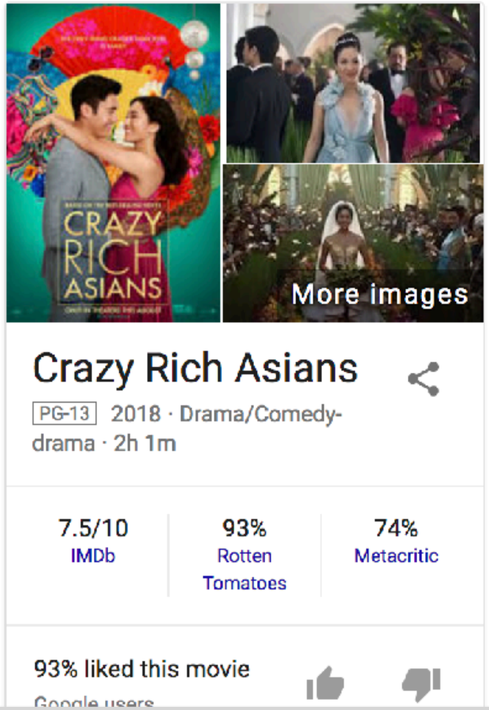
reuse?
every data model
is a new one!

designing a database: the concept approach

what if we identified concepts instead?







what are the key concepts?

can we disentangle services?

TMDB OAS Service Status Support

v3 Guides API Reference Changelog

FIND
Find By ID GET

GENRES
Movie List GET
TV List GET

GUEST SESSIONS
Rated Movies GET
Rated TV GET
Rated TV Episodes GET

KEYWORDS
Details GET
Movies GET

LISTS
v3 or v4 lists? 🤔

Movie List

GET https://api.themoviedb.org/3/genre/movie/list

Get the list of official genres for movies.

YOUR REQUEST HISTORY

0 Calls

Your API calls will appear here. Make a request to get started!

QUERY PARAMS

language string

RESPONSE

200
200

Rotten Tomatoes Search movies,

TRENDING ON RT The Creator Sex Education The Fall of

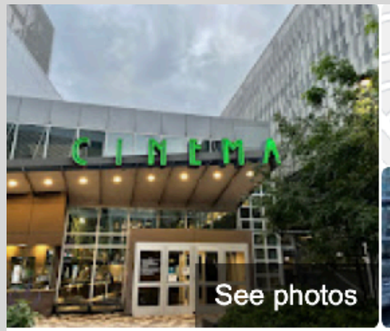
Movies in Theaters (2023)

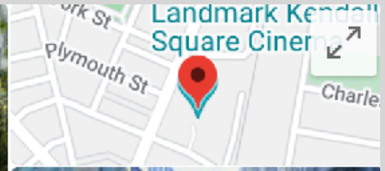
IN T

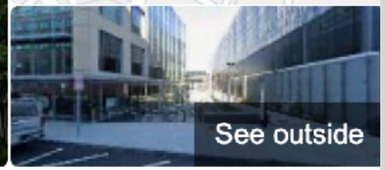
SORT GENRE RATING AUDIE

67% --
Surprised by Oxford
ed Sep 27, 2023
WATCHLIST

-- --
The Blind
Opens Sep 28, 2023
WATCHLIST







Landmark Kendall Square Cinema

Website Directions Save

4.6 ★★★★½ 934 Google reviews

Movie theater in Cambridge, Massachusetts

Movie theater screening new releases as well as independent, foreign & avant-garde flicks.

Located in: One Kendall Square

Address: 355 Binney St, Cambridge, MA 02139

Phone: (617) 621-1202

standard criteria:
clear **purpose**
separable & **reusable**
familiar

other hints:
who **updates**?
frequency of updates?

some candidate concepts

concept Movie
purpose info about all movies
state
genres: Movie -> **set** Genre
title: Movie -> **one** Title
year: Movie -> **one** Year
remakeOf, sequelTo: Movie -> **opt** Movie

concept Showing [Movie, Theater]
purpose info on current movie showings
state
movie: Showing -> **one** Movie
theater: Showing -> **one** Theater
time: Showing -> **one** Date
screen: Showing -> **one** String

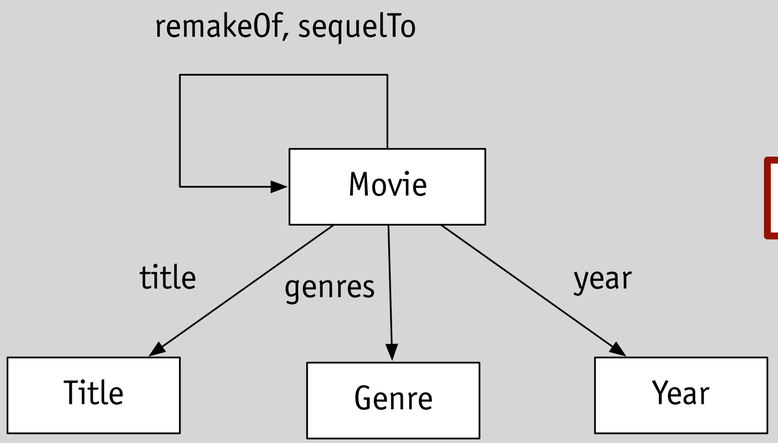
concept Business [Location]
purpose info on businesses
state
name: Business -> **one** String
address: Business -> **one** Address
website: Business -> **one** URL
location: Business -> **one** Location

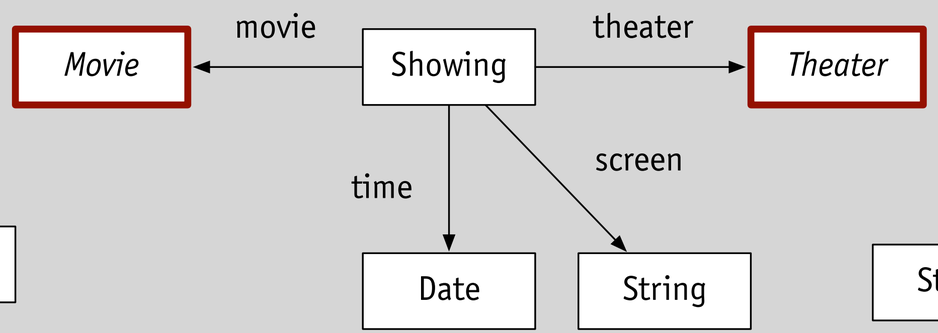
update frequency
Movie: 2/day
Business*: 1/day
Showing: 10k/day
**theaters only*

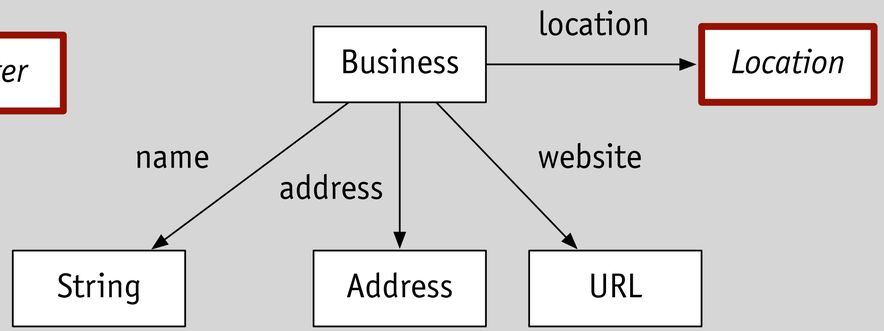
update method
Movie: approved users?
Business: verified?
Showing: verified?

new concepts
will need concept
for verifying businesses

drawing a global data model (1)





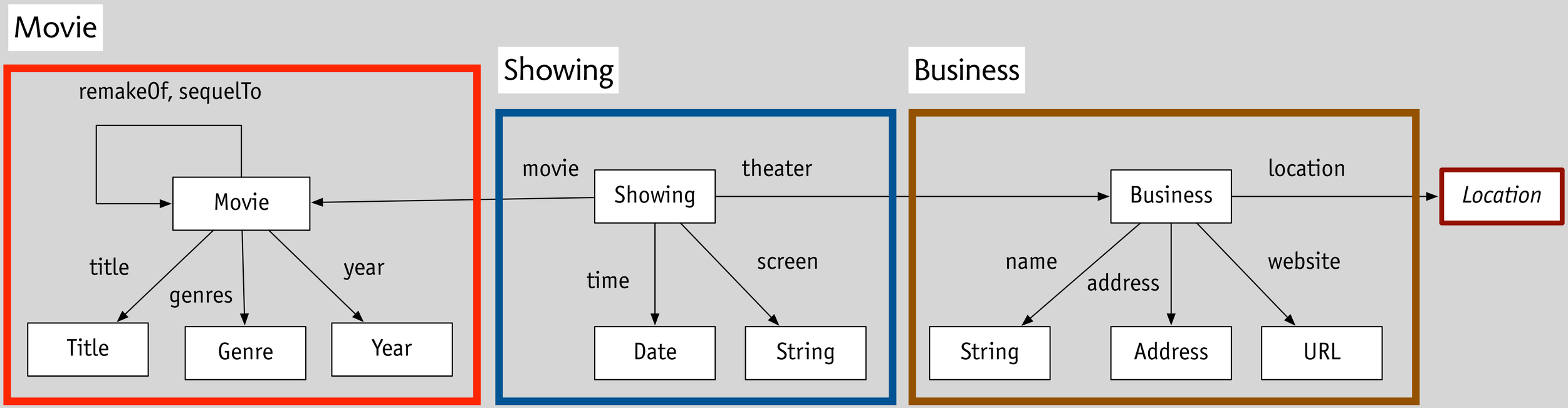


concept Movie
purpose info about all movies
state
genres: Movie -> **set** Genre
title: Movie -> **one** Title
year: Movie -> **one** Year
remakeOf, sequelTo: Movie -> **opt** Movie

concept Showing [Movie, Theater]
purpose info on current movie showings
state
movie: Showing -> **one** Movie
theater: Showing -> **one** Theater
time: Showing -> **one** Date
screen: Showing -> **one** String

concept Business [Location]
purpose info on businesses
state
name: Business -> **one** String
address: Business -> **one** Address
website: Business -> **one** URL
location: Business -> **one** Location

drawing a global data model (2)




```
app ShowtimeDatabase
include
  Movie
  Showing [Movie.Movie, Business.Business]
```

queries across concepts
application view
composes elements from
each concept

how to handle locations?

concept Location [POI]

purpose find points of interest by location

principle

after POIs are added using makeLoc/add,
you can find nearby POIs using makeLoc/findNearby:
makeLoc (a1, l1); add (p1, l1); ...
makeLoc (a2, l2); add (p2, l2); ...
makeLoc (a, l); findNearby (l, s)
{s contains POIs from p1, ... near to address a}

state

location: POI -> **one** Location

actions

add (p: POI, l: Location)
makeLoc (addr: String, **out** l: Location)
findNearby (l: Location, **out** s: **set** POI)

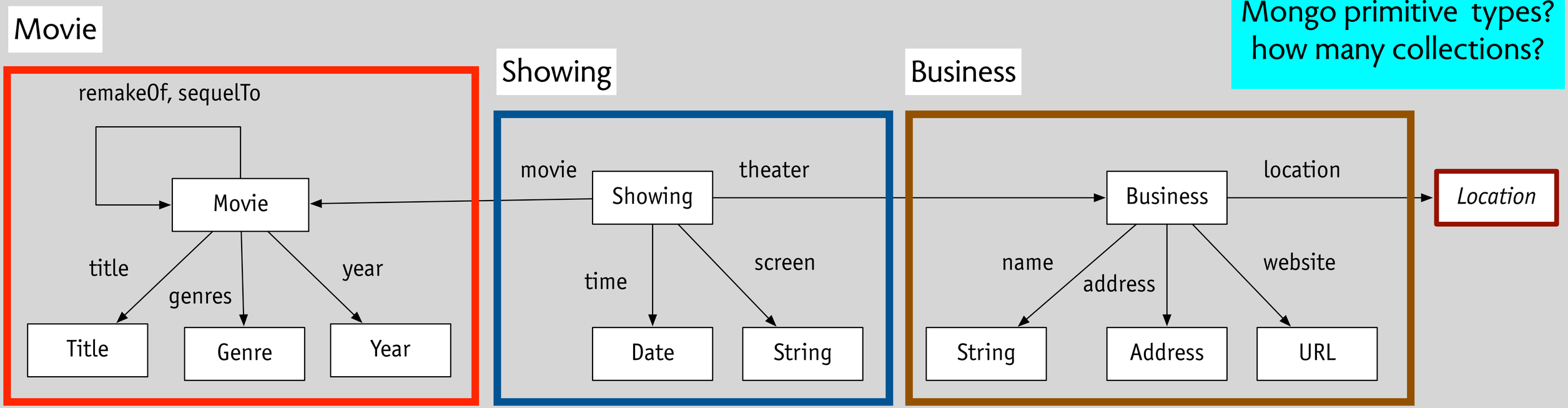
data structure/algs
hierarchical regions
quadtree, eg

can you add ratings of movies and theaters?

```
concept Rating [Item, User]
purpose crowdsource quality measure
state
  avgRating: Item -> one Int
  vote: User -> set Vote
  for: Vote -> one Item
  rating: Vote -> one Int
```

implementing in MongoDB

decisions
Mongo primitive types?
how many collections?

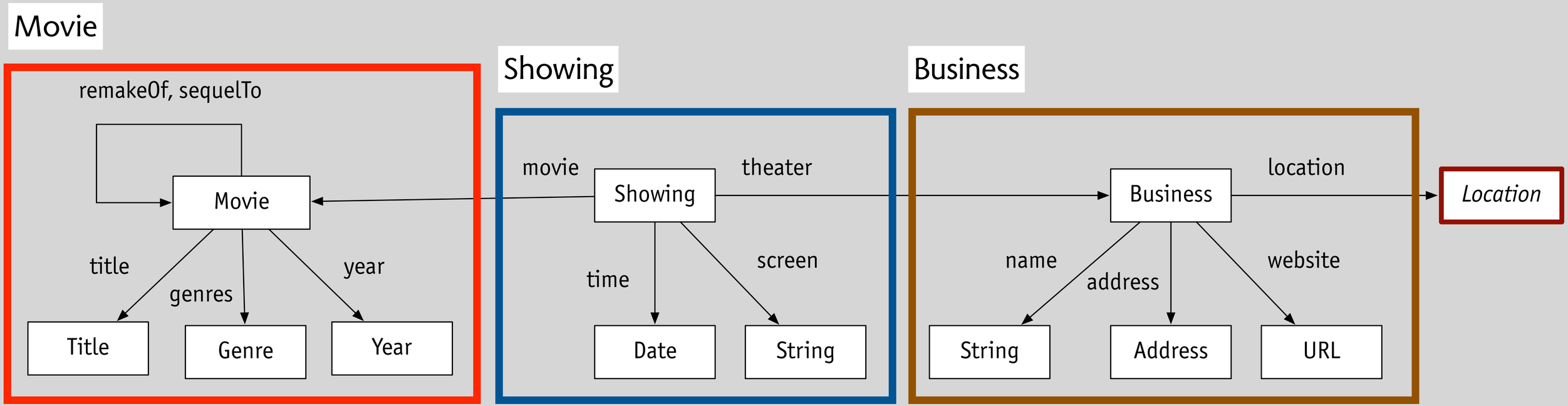



```
Movie {
 _id: ObjectId
 title: String
 genres: [String]
 year: Int
 }
```

```
Showing {
 _id: ObjectId
 movie: ObjectId
 time: Date
 screen: String
 theater: ObjectId
 }
```

```
Business {
 _id: ObjectId
 name: String
 website: String
 location: ObjectId
 address: String
 }
```

what about these?




```
Showing {
  _id: ObjectId
  movie: ObjectId
  time: Date
  screen: String
  theater: ObjectId
  }
```

```
MovieShowings {
  _id: ObjectId
  movie: ObjectId
  showings: [
    {
      theater: ObjectId
      screen: String
      time: Date
    } ]
  }
```

```
TheaterShowings {
  _id: ObjectId
  theater: ObjectId
  showings: [
    {movie: ObjectId
      screen: String
      time: Date
    } ]
  }
```

```
TheaterMovieShowings {
  _id: ObjectId
  theater: ObjectId
  movie: ObjectId
  showings: [
    {
      screen: String
      time: Date
    } ]
  }
```

tradeoffs
relative benefits of
these representations?

summary of what you learned today

https://tinyurl.com/6104-feedback

database models

OO, relational, collection

abstract data models

relations & sets, global diagram

concept-driven data

focus on separable services

implementation

types, collection structure