purposes

after this class you’ll understand
why data models are useful
what a data model is and what it means
how to construct a data model

key ideas
separation of concerns: rep vs content
abstract structure with relations & sets
data models as invariants
why data models?
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>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Theaters

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>location</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<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>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>“Fury”</td>
<td>“R”</td>
<td>“action”</td>
</tr>
</tbody>
</table>

Showings

<table>
<thead>
<tr>
<th>_id</th>
<th>title</th>
<th>time</th>
<th>rating</th>
<th>genre</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>“Fury”</td>
<td>“7:00pm”</td>
<td>“R”</td>
<td>“action”</td>
</tr>
</tbody>
</table>

Theaters

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

Movies

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Fury”</td>
</tr>
</tbody>
</table>

Object-oriented databases: so many choices!
in mongo: still choices!

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

one document in the collection Movies

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

one document in the collection Movies

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

one document in the collection Theaters
But nothing is gained —on the contrary! — by tackling these various aspects simultaneously. It is what I sometimes have called “the separation of concerns”, which, even if not perfectly possible, is yet the only available technique for effective ordering of one’s thoughts that I know of. This is what I mean by “focussing one’s attention upon some aspect”: it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect’s point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

Dijkstra, On the role of scientific thought (EWD447)
separation of concerns

a tool for thinking
one concern at a time: a way to make progress
helps expand representation possibilities
surprise: focus exposes subtle issues in what data is

other advantages
can separate roles: designer vs. implementer

what the data is vs how to represent it

biggest idea in programming?
data is central

state characterizes behavior
invariant is like a planet’s orbit

not just databases
rich transient state too
wanted...

a notation for data
lightweight, minimal & easy to learn
more succinct than code
more precise than sketches

representation independent
leaves rep choices open
not tied to a particular database
an analogy: architectural plans

- Stata
- Student Street
- column
- doorway
- stair
syntax of data models
example email app

Teabox

4 days left to order your box of surprises

Join Fresh Beginnings, our subscription service and get 25% off on your first box!

SUBSCRIBE NOW
Classify

It will do you good.

Spread recycling!! To save limited natural resources for our children's future.

© 1991 Produced by Super Planning Company Limited
identify sets of atoms

Message

Address

Folder

a SET of messages
specialize sets

Folder

UserDefined
Predefined

Folder

Inbox
Sent
Trash

classification is IMMUTABLE: Inbox can’t become Trash

a folder is user-defined or predefined

these happen to be singleton sets (but might not be)
identify dynamic subsets

no immutability marking: a message can be unread now and not unread later

not italicized: may have messages that are not unread
relate atoms (attributes)

Message \[\text{from}\] Address

relation is target
IMMUTABLE: to address doesn’t change
relate atoms (fluents)

don’t want one relation represented by two arcs

a generalization, introduced to unify relations
determine multiplicities

Message \rightarrow Address from !

Address \rightarrow \text{Object}

Object \rightarrow \text{Folder}

Object \rightarrow \text{Message}

contents

exactly one

one or more

at most one

?
putting it all together
adding textual constraints

one inbox, trash and sent folder
predefined folders not contained by or contain others
no unread messages in sent folder
contents is acyclic
syntax summary sets

A \subseteq B

A \subseteq B

A_1 \subseteq B \land A_2 \subseteq B

A_1 \cap A_2 = \emptyset

A_1 \subseteq B \land A_2 \subseteq B

B \subseteq A_1 \cup A_2
syntax summary relations

$R \subseteq A \times B$

- over time, each $A$ is mapped by $R$ to the same $B$s
- over time, $R$ maps the same $A$s to each $B$
- $R$ maps each $A$ to $n$ $B$s
- $R$ maps $m$ $A$s to each $B$
- $+$ one or more
- $*$ zero or more
- $!$ exactly one
- $?$ at most one
- omitted $= *$

short for:

used when $R_1$ and $R_2$ share same properties
semantics of data models
model = set of instances

\{x, y: Nat \mid x+y = 4\}

syntax: a constraint on a pair of numbers

\{(x:0, y:4), (x:1, y:3), (x:2, y:2), (x:3, y:1), (x:4, y:0)\}

semantics: a set of instances

syntax: model

semantics: set of instances
instance

- (Inbox)
  - contents
  - (Unread)
    - from
    - to
    - (Address)
  - (Sent)
    - contents
    - (Message)
      - from
      - to
      - (Address)
    - (Address)
  - (Trash)
    - contents
    - (UserDef)
  - (UserDef)

- (Address)
  - (Address)
  - (Address)
example satisfying instance

\[ \epsilon \]

one Inbox and one Trash and one Sent
no Predefined.contents & Folder
no Sent.contents & Unread
acyclic [contents]
example relating wrong kinds

one Inbox and one Trash and one Sent
no Predefined.contents & Folder
no Sent.contents & Unread
acyclic [contents]
example multiplicity violation

\[ \varepsilon \]
example multiplicity violation

one Inbox and one Trash and one Sent
no Predefined.contents & Folder
no Sent.contents & Unread
acyclic [contents]
example textual constraint violation

one Inbox and one Trash and one Sent
no Predefined.contents & Folder
no Sent.contents & Unread
acyclic [contents]
monotonicity

adding diagram features adds constraints, so you can grow the diagram

from $\subseteq$ Message $\times$ Address

message

\[ \text{Message} \rightarrow \text{Address} \]

all \( m : \text{Message} \mid #m.\text{from} = 1 \)

all \( m : \text{Message} \mid m.\text{from} \text{ unchanging} \)
exercise:

reading data

models
javascript objects

- Value
- Undefined
- Primitive Value
- Function
- String
- Object
- Slot
- Constructor
- Prototype
- Slots
- Name

exercise
draw some interesting instances
add some textual constraints

review constructors at Prototypes/Constructors in JavaScript Live slides
solution: javascript

some constraints
slots of an object have distinct names
object can’t be its own prototype
function object has slot with name ‘prototype’

an interesting case
object and its prototype have slots with same name

a tricky case: is this true?
all o: Object | some s: o.constructor.slots | s.name = ‘prototype’ and s.value = o.prototype
comparing notations
origins of our modeling notation

- Mathematical logic
  - ZF set theory
  - relational calculus (Tarski)
  - relational model (Codd)
  - Z notation
  - ER & other data models
  - object model notations (OMT etc)
- Object-oriented development
  - logic diagrams (Euler, Venn, Peirce)
  - model checking
- Software verification
  - Alloy Language
  - Alloy Diagrams
- Relational databases
  - Unified Modeling Language
the entity-relationship model

 allows 3-way relations

relations are diamonds

a deletion rule

Fig. 11. An entity-relationship diagram for analysis of information in a manufacturing firm

alloy diagram notation

- can express generalization
- all relations binary
- for 3-way relationship, introduce tuple
- mutability
why Alloy diagrams?

- simple & small syntax
- comparable to ER
- much less complex than UML

- easy to draw
- requires fewer special widgets

- crucial features for modern programming
  - generalization (subsets/supersets)
  - mutability markings

- backed by textual notation
- but don’t expect you to use it