transforming object models to relational schemas

Daniel Jackson
an object model

for a shopping site
› eg, Amazon

not complete
› inadequate state?
one model, many implementations
step 1: identify entities

an entity
› will become a database table
› is a “composite objects” (sort of)
› is more than a “value”

criteria
› more entities, more joins
› fewer entities, more redundancy

type
› clearly: User, Order, Item
› also: Address (why?)

note
› illustrates why “attributes” are not a good notion in first phase of modeling
step 2: assign attributes to entities

an attribute
› will become a database column
› in the table of the entity it’s assigned to

criteria
› assign to all adjacent entities

example
› User gets email
› Item gets descr, itemno, cost

other options: if entity is 1:1 with another (eg, User-Profile), can move attribute
step 3: pick data representation

data representation
 › either built-in type
 › or expand into multiple columns

criteria
 › try to use built-in types with validations
 › beware of length limitations
 › remember this is just the rep:
   model class can wrap with methods

eexample
 › Text as :text
 › ItemNo as :integer
 › Currency as :decimal
 › Email as :string
step 4: insert connecting sets

- connecting sets
  - needed for many-many relations

- criteria
  - one connecting set for each relation

- example
  - *items* is many:many
  - so add *OrderItem*
step 5: define classes

idea
› finally turn into code

criteria
› one ActiveRecord class/entity
› declare attributes as columns in migrations
› add association annotations in models
› declare associations in migrations

example
› declare class and migration for Item
› column for descr, etc; belongs_to :order_item

other options:
with “aggregation”, can assign >1 column to one ActiveRecord class
connecting set need not be a class
step 6: design class methods

idea
› database table is just the rep of an ADT
› access is through class methods

criteria (see 6.005!)
› simplify controller code
› localize maintenance of rep invariants
› hide representation details

sample questions
› what methods might Order have?
› what invariants should Order preserve?
› should OrderItem be exposed?
advanced considerations

redundancy
› avoid costly joins for frequent navigations
› add redundant data
› example: orderitem.order.user.email -> orderitem.email

security
› separate critical data into separate table(s)
› fine-grained access control supported by databases
› example: credit card number for customers

avoiding contention
› beyond this course (transactions, locking, etc)
› but good to separate high/low frequency
› example: don’t store user profile and user tracking in same table
traditional database design

- database schema
- in normal form?
  - yes: done
  - no: fix schema by adding tables
example of normal form violations

<table>
<thead>
<tr>
<th>reviewer</th>
<th>subject</th>
<th>rating</th>
<th>email</th>
<th>ratingstars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chloe Closure</td>
<td>Lucid</td>
<td>3</td>
<td>cc@mit</td>
<td>***</td>
</tr>
<tr>
<td>Chloe Closure</td>
<td>Clover</td>
<td>5</td>
<td>cc@mit</td>
<td>*****</td>
</tr>
<tr>
<td>Ann Alert</td>
<td>Clover</td>
<td>5</td>
<td>aa@mit</td>
<td>*****</td>
</tr>
<tr>
<td>Ben Bitdiddle</td>
<td>Cosi</td>
<td>3</td>
<td>ben@mit</td>
<td>***</td>
</tr>
<tr>
<td>Ben Bitdiddle</td>
<td>Lucid</td>
<td>4</td>
<td>ben@mit</td>
<td>****</td>
</tr>
</tbody>
</table>

data example
- a reviewing database in one table

second normal form
- no field depends on just part of a key
- key is (reviewer, subject), and email depends on reviewer alone

third normal form
- no field depends on another field but not on the key
- eg, ratingstars depends on rating alone
so?

these problems don’t arise
› if you started with an object model

but may arise if you
› started with tables
› or with a really bad object model!