Assignment 5: Data Models

Overview:

There was no single correct data model for any problem in this assignment. However, we have put together this document to help you understand the key takeaways for each problem.

To keep grading consistent, each TA was assigned to grade a question for all students in the class. For each problem, the grader has provided an example data model, along with example textual constraints, explanations (if necessary), and insights based on the data model. Remember that other correct solutions are possible.

The grader has also noted common errors your solutions. Please read through these carefully.

If you would like to email the grader, find the appropriate TA in the list below. As the subject of the email, put: 6.170 Data Models <problem number> Grade Request.

- A1: Donald (dmlittle@mit.edu)
- A2: Katie (ksiegel@mit.edu)
- A3: Harihar (hsubrama@mit.edu)
- A4: Neha (npatki@mit.edu)
- A5: Tony (tuantran@mit.edu)
- A6: Rachel (rswang@mit.edu)
- A7: Katie (ksiegel@mit.edu)
- A8: Rachel (rswang@mit.edu)
- A9: Harihar (hsubrama@mit.edu)
- A10: Neha (npatki@mit.edu)
- B1: Tony (tuantran@mit.edu)
- B2: Kim (toyk2a@mit.edu)
- C: Henry (hnassif@mit.edu)
- D: Dishaan (disaan@mit.edu)
Problem A1: Drawing

Example Model:

Possible Insights:
1. When different elements are grouped together the relative depth of a subset of the elements might change in order to accommodate for elements in groups having a common depth (objects being displayed together). For example, if we have 3 different objects A, B and C where A is in front of B which is in front of C and we group A and C together, C must now be displayed in front of B or A must be behind B.
2. Because the relative order of a group changes when they are grouped, what happens to the order of the elements after they are ungrouped? Does the element go back to the original depth before it was grouped or does it keep the new depth it acquired once it was grouped?

Common Errors:
1. Model / Query: Data model wasn’t able to support the relative order between multiple elements in the same group or between different groups. If two elements are grouped together, both elements should be in front or behind an element outside the group, but those two elements should still have a relative order between each other.
Problem A2: HTML/CSS

Example Model:

![DOM Element Diagram]

Example Textual Constraints:
1. The DOM Elements form a tree structure, which means that the parent relationship is acyclic.

Example Explanations:
1. A “value” for a CSS property encapsulates multi-part values, such as “0px 0px 0px 0px !important.”

Possible Insights:
1. The root DOM element does not have a parent.
2. Changing formatting via DOM modifications for a class does not change the CSS properties for that class; instead, changes are made by modifying the style of specific DOM elements with that class.

Common Errors:
Diagram
1. Missing association between HTML object and property
2. CSS classes missing names
3. CSS properties missing names and values

Query:
5. Property not having property name or property value, or class not having class name.
   You need the name to query the property or class.

Textual Constraints:
6. Detailing inheritance as a textual constraint. Inheritance properties should be an insight, rather than a textual constraint. Textual constraints involve some aspect of data state,
and inheritance explains how a third party (e.g. a browser) would interpret that state to, for example, render something on a page. I did give points to individuals who explained that the properties of a parent should be copied over to the children, as technically this is a constraint on data state.

Problem A3: Contact List

Example Data Model:

![Diagram of Contact List]

Example Textual Constraints:
1. A Phone Number cannot be associated with an Email Label
2. An Email cannot be associated with a Phone Label

Example Definitions:
1. User Defined represents a user-defined location label (i.e. a label that could be created using the app)

Possible Insights:
1. Requiring names to have a first name and representing names with a first name / last name structure carries some risk. For example, what if I don't know somebody's last name or I want to store a middle name?
2. Notice that the model allows a phone number to be associated with multiple elements of PhoneInfo. This is because we cannot assume that contacts all have different phone numbers. For example, members of the same family often share the same landline. A similar argument applies to emails.
3. Notice that we cannot store a name, phone number, or email that is independent of any contact. It's not clear why this would be useful, so it has been forbidden in the model.

Common Errors:
Model:
1. Same name used for two sets or two relations
2. The subsetting arrow is named
3. The subsetting arrow has a multiplicity
4. Model is blurry, illegible, or has cropped out names
5. Immutability or multiplicity symbol is in the MIDDLE of the arrow
6. Relation or set does not have a name

Names:
7. Uses undescriptive name
8. Uses “has” or “is” for relation names
9. Uses one set to represent two kinds of data without explanation (e.g. one set representing both email and phone number)

Generalization:
10. Model uses two relations that do the same thing and could have used subsetting instead and does not explain why this wasn't done (e.g. if possible, you should make Phone and Email a subset of a common superset)
11. Subsets don’t make sense (e.g. A is a subset of B, but an A is not a B).
12. Relation does not serve a purpose or is unexplained (e.g. it doesn’t make sense for a contact’s name to be associated with a location)
13. Subsetting breaks model/query requirements (e.g. If you say that a Name is either a First Name or a Last Name and that each Contact has exactly one Name, then that means that the Contact cannot have both a first name and a last name - this is bad subsetting)
14. Subsetting specified as textual constraint

Multiplicity:
16. Contact can have many first names (or many last names or names).
17. Two people can’t have same first name (or same last name or name)
18. Location type uniquely determines email (or phone). This does not make sense. If I tell you “Home”, how can you determine a single corresponding email (or phone number). After all, many emails (phone numbers) could be associated with the “Home” location
19. First name (or last name or name) uniquely identifies contact. This is not reasonable because many people have the same first name (or last name or name)
20. Contact can have either a first name or last name, but not both.

Model and Query Functionality:
21. One of the required queries is unsupported
22. One of the aspects of the system that we asked you to model is left unmodeled (e.g. an email should be allowed to have a location) or incorrectly modeled

Textual Constraints:
23. Textual constraint does not actually constrain the set of instances allowed by the model
24. Textual constraint reflects something already shown in the model
25. Used textual constraint instead of showing in model (e.g. don’t say that “a contact has exactly one name”, indicate it in your model)
26. Textual constraint contradicts model
27. Missing a necessary textual constraint

Explanation/Insights:
28. No explanation for unclear set
29. No insights
30. Explanation or insight is incorrect
Problem A4: Quiz

Example Model:

![Diagram of Quiz model]

Example Textual Constraints:
1. All answers in a response must be valid answers of one of that quiz’s questions.

Possible Insights:
1. I allowed multiple responses from a single user because the responses themselves are immutable. This is the same as what 6.170 does for quizzes, and it closely parallels what happens when a student physically submits a quiz; they are no longer allowed to take it back and change it. However, depending on the teacher, it might be okay to submit another quiz. Then, it’s up to the grader what to do with multiple submissions (e.g. only score the latest one).
2. In my model, Questions can be shared by multiple Quizzes. Currently, Google Forms does not allow this, but I think it could be useful if a teacher wants to repeat a question across quizzes. For this to work, we’d have to separate out the Question ID from the actual content of the Question, because Answers are associated with different Questions.
3. My mutability markings for Question-Answer, and Answer-Correct Answer show the state of the quiz after it has been created. We’re assuming that the quiz has been created in one atomic step, and that changes cannot occur to it later. It might make sense to leave these markings out if the creator of the quiz wants to add answers, and/or mark new ones as correct (Google Forms lets you do this). However, this seems like a bad idea if students have already submitted a quiz.
Common Errors:

Model:
1. Having an entity called “Score” without any indication of Correct Answer. Your model needs to contain all the information necessary to score the quiz. Simply having a box with “Score” does not do this.
2. Using a boolean to see if an answer is correct. This is a bad smell, and keeping a subset is the proper approach.
3. Separating Answers into “Selected” and “Unselected” subsets. This implies that, of all the answers, some are selected and some are not. It does not show how some users can select different answers.
4. Separating Quiz out into subsets “Template” and “Submission”. This does not show how submissions are linked to a template.

Multiplicity:
5. If Responses only include Answers, then Answer must have exactly 1 Question. Otherwise, we don’t know which Question was answered.
   (If Response had both Question and Answer, this multiplicity is no longer necessary)
6. Similarly, Response should belong to 1 Quiz, otherwise you don’t know which Quiz that response was for.

Textual Constraints:
7. It was common to miss the textual constraint. In general, you should check if you need a textual constraint if you see a loop somewhere in your model.
8. Instead of a subset, some students had two relations from Question to Answer: 1 representing “choices” and the other with “correct choices”. If this is the case, then you need a textual constraint explaining that correct choices are a subset of choices.
9. It’s unnecessary to add details like how you’d go about scoring a quiz. Those are implementation details.

Insight:
10. Saying something like “every answer must have 1 question” is not an insight. It’s already implied by your model.

Problem A5: Conference Call

Example Model:
Example Textual Constraints:
1. A connection can't be in both active and hold.
2. A party can't be in both initiator and participants.

Example Explanations:
1. The relation participants maps a connection to the parties that joined the connection.
2. The parties mapped by participants does not include the initiator.

Possible Insights:
1. Active and hold are not subsets of connection since they are from the point of view of the party. The same connection could be active for one party, but on hold for another party.
2. One way a connection can be started is by having one party call other parties. For example, person 1 (the initiator) can call person 2 on a cell phone and then person 1 can decide to call more people to add them to the connection. Another way a connection can be started is to have parties dial in to some central number that is provided by the initiator. This is common in conference calls for meetings at work.

Common Errors:
Diagram:
1. Showing active/hold as subsets of connection rather than relating them somehow to the party.
2. Multiplicity errors.
3. Missing names for all relations, poor choice of names for relations.

Query:
1. Missing phone number.
2. Missing some way to track connection time.
3. Able to get the connections associated with a party, but not the parties within a connection.

Textual Constraints:
1. Textual constraint is repetitive - already shown in the model.
2. Textual constraint is unnecessary - could have been shown in the model.
3. Textual constraint contradicts the model.

Explanation/Insights:
1. No explanation for unclear set.
2. No insights.
3. Incorrect explanation/insight.

Problem A6: Prerequisites

**Example Model:**

```
+ requires +
*                *

Class ←→ Prerequisite

+ fulfilled_by +

name !
```

**Note:** It was also possible to model this problem as an AST with expressions in which sub-expressions were combined with AND or OR.

**Example Textual Constraints:**
A class cannot be required by any itself or any of its prerequisites (must be acyclic).

**Example Explanations:**
The name is the course number of the class.

A class requires all its prerequisites to be fulfilled (AND). A prerequisite is satisfied by any one of the classes it can be fulfilled by (OR). This data model is CNF (AND of ORs).

**Possible Insights:**
1. There must be classes with no pre-requisites (i.e. introductory classes, to serve as the “base case”). A student enters MIT without having taken any classes, so there must be classes in the system that the students can take to start fulfilling prerequisites of higher level classes. Furthermore, there cannot be any cycles. If there were any cycles, it would be impossible to fulfill a class’s pre-requisites.
2. This data model does not distinguish between co-requisites and prerequisites and does not model “Permission of Instructor” as a possible pre-requisite. This could be modeled by create Pre-requisite “Types” that included the different requisite types.
3. There are some classes with multiple names (6.046 is the same class as 18.410), however the same name cannot belong to multiple classes. If a course has multiple names (course numbers), then its prerequisites should be the same. A name (course number) cannot belong to many classes, as different classes have different prerequisites.
Common Errors:
Model:
1. Using a “Class” object with multiple cyclic relations to model “requires” and “fulfilled_by”.
Mutability:
2. Missing an immutability marking from Class → Name.
Multiplicity:
3. Classes might not have pre-requisites.
4. Missing multiplicity marking on an interchangeable set / group of prerequisites (“satisfies one of”).
Textual Constraints:
5. Missing textual constraint that prerequisites graph must be acyclic.
Insights:
6. Insights such as “This problem was easier/harder than expected” or “This problem was interesting” were not accepted.
7. Anything obviously inferred/derived from your model was not accepted.

Problem A7: Voting ballots

Example Model:

Example Textual Constraints:
None necessary

Example Explanations:
None necessary except for clarification purposes. Sometimes useful to explain the difference between a ballot and a vote.

Possible Insights:

1. Once a ballot is cast, it cannot be changed.
2. Allows for write-in candidates.
3. We need the vote-to-office relation, because a candidate might run for multiple offices.

Common Errors:
Model:
1. Ballots are not connected to offices. We need the vote-to-office relation, because a candidate might run for multiple offices.
2. Offices, candidates, template missing from diagram, so the diagram is incomplete.
3. “Vote” stored as a count rather than tied to office/candidate
Query:
4. Not taking into account that a candidate can run for multiple offices, so just linking a voter/vote with the candidate is insufficient.
5. Missing the concept of a single “vote” in the diagram--without votes, cannot perform queries.

Problem 8: Java Types

Example Model:

**Note:** Mutability markings are based on a Java program during runtime.

**Example Textual Constraints:**
A class cannot extend itself. An interface cannot extend itself (no cycles). The runtime type must be in the ancestry of the declared type. Note: this rule is established statically.

**Example Explanations:**
Runtime type is based on the constructor of the object that the variable holds. The declared type determines what methods the variable has at execution.

**Possible Insights:**
1. Objects can be casted to another type during runtime, however the cast may fail and cause a runtime error. For example, you can cast a float to an int, but you can’t cast a float to a boolean.
2. Interfaces cannot be instantiated. Interfaces help define methods used by multiple classes. If there is an interface and no class that implements it, it is “useless.”

Common Errors:
Model:
1. Missing the notion of runtime type and declared type.
2. Missing the notion of object and variable.
3. Missing the extends/implements relation.
Mutability:
5. Missing an immutability marking on the runtime type of an object.
6. Missing an immutability marking on the name of a variable.
7. Missing markings on the extends/implements relations.
Multiplicity:
8. Missing that a variable (optionally) holds an object.
9. Restricting a class/interface to only extending one other interfaces.
10. Allowing classes to extend more than one class.
11. Requiring classes to implement interfaces.
Textual Constraints:
12. Missing the constraint that classes and interfaces (Types) cannot extend themselves.
13. Saying “Declared type must equal the runtime type”. Consider: List myList = new ArrayList<String, String>().
14. Having textual constraints that should be in the model instead (i.e. runtime type cannot be an interface).
Insight:
16. “This taught me a lot about Java” or “This refreshed my memory of Java” was not accepted.
17. Need to be non-obvious and not obviously inferred by from your diagram.

Problem A9: MIT Buildings and Rooms

Example Data Model:
Example Textual Constraints:
1. A room should be associated with a building B if and only if the room’s number indicates building B’s building number
2. A building should be associated with an area X if and only if the building’s number indicates area X

Example Definitions:
1. A Room Number is the full <optional area>building-room designation of a room. For example, 32-G825, W85ABC-400, and E34-400 could be Room Numbers.
2. A Building Number is the full <optional area>building designation of a building. For example, 32, E34, 34, and W85ABC, could be Building Numbers. Notice that a Building Number is fundamentally a different kind of entity than a Room Number, so we cannot combine them into the same set. Otherwise, our model would be incorrect. For example, it would allow a building number to contain a hyphen, which is not possible.
3. Name represents an informal name for a place. For example, “Green Building”, “Memorial Lobby”, and “Kirsch Auditorium” could be elements of Name, while 54, 10, and 32 cannot.

Possible Insights:
1. The area is optional because some buildings don’t have areas. For example, building E34 is in area E, but building 34 is not in any area.
2. Notice that we require an area to have at least one building and a building to have at least one room. So, it would be impossible, for example, for MIT to create a new building with no rooms.
3. New buildings can be constructed/demolished (e.g. Bexley) in an area and new rooms can be built in a building (e.g. EECS undergrad lounge in building 36).
4. It is not possible for a building to move to a different area or for a room to move to another building.
5. Places at MIT can have multiple names and can be renamed (e.g. Memorial Lobby and Lobby 10)

Common Errors:
Names:
1. Uses one set to represent two kinds of data without explanation.

Generalization:
2. Model uses two relations that do the same thing and could have used subsetting instead (e.g. if possible, you should make Building and Room a subset of a common superset. If not, you should explain why it can’t be done)
3. Relation does not serve a purpose or is unexplained (e.g. it doesn’t make sense for a Room to be associated with an Area, unless you justify)
4. Impossible room & building combination. For example, if your model allows 34-401 to be in building 32, that’s an error.

Multiplicity & Mutability:
5. Room can be in more than one building (and no justification)
6. Building can be in both West and East (and no justification)
7. Can't have both building 34 and E34
8. Can't have both room 34-401 and 33-401
9. Rooms or buildings can have multiple numbers
10. Room can change buildings (and no justification provided for why or how this is possible)

Model and Query Functionality:
11. One of the required queries is unsupported (e.g. get room by name)
12. One of the aspects of the system that we asked you to model is left unmodeled (e.g. no notion of areas, no relation between buildings and rooms) or incorrectly modeled

Textual Constraints:
13. Used textual constraint instead of showing in model (e.g. don't say that “a building has exactly one name”, indicate it in your model)
14. Missing textual constraint

Problem 10: Shopping Cart

Example Model:

![Shopping Cart Diagram]

Example Textual Constraints:
1. No two orders in the same cart can have the same item

Possible Insights:
1. For a company like Amazon, the same product can be sold by multiple sources, so the distinction between two Items is subtle. You would need to have distinct IDs for an item different suppliers.
2. A cart may not have a user because the user might not exist (i.e. the person hasn't logged-in to the site). However, once a user logs in, that User must have exactly 1 cart. This closely mimics functionality that Amazon has: Once a user creates an account, that user always has a cart, whether or not they are currently on the site. If they put something in the cart, and come back to the site in a few weeks, that cart is still there.

Common Errors:
Model/Generalization:
1. Nearly all students left out the “Order” entity. Instead, they had a cart with multiple Items, and each Item with a Quantity and Name. **This does not work.**
   a. If Item has exactly 1 Quantity, you don’t consider the case where two users can order different quantities of the same item. You also cannot explain that you’d just create a new item every time there’s a new quantity. *(That’s a dependency that needs a tertiary entity like “Order”)*
   b. If Order has 1 or more Quantities, you cannot figure out who wants which quantity. For example, if I have an item in my cart, and that item has quantities \{3, 4, 5\}, which one of those is my desired quantity?
2. Creating subsets of User (e.g. Guest and Member) is okay, but there shouldn’t be a relation between the two. If the subset relation is mutable, it already implies that a Guest can turn into a Member. But having a relation implies “every guest has a member” which makes no sense.
3. All arrows represent relations between entities. There should not be arrows for actions like “log in”, “add”, “remove”, etc.

Textual Constraints:
4. Explaining what happens when a user logs in or logs out is not a textual constraint. The model is complete without this information.

Insights:
5. Your insight cannot just be “you can still have a cart if you’re not logged-in”. This is not an insight, because we told you this was part of the problem statement.

**Problem B1: Tagging in Facebook**

**Example Model:**

```
image_tagged \! → Image

Tag

posted \!

visible_to +

User

user_tagged, tagged_by \!

friends
```

**Example Textual Constraints:**

1. A user can view an image if he/she:
   a. posted the image.
   b. is tagged in the image.
c. is friends with the user who posted the image.
d. is friends with any user tagged in the image.

2. A user can’t be friends with himself/herself.
3. A user can only be tagged once in the same image.

Possible Insights:
1. That who does the tagging has no effect on visibility.
2. A user does not have to be friends with another user to tag that user.
3. People who are not friends with a user tagged in an image can view the image.

Common Errors:
Diagram:
1. Depicting friendship by having a friends list set and some kind of relation from user to the friends list.
2. Depicting friendship by having a relation from user to some subset of user, such as friend.
3. Having subsets of users such as “TaggedUser”, “Friend”, “ViewableUser”, etc. This is fundamentally broken because a subset is with respect to a context (a user), which should not be confused with a relational state between users, tags, and images.
4. Multiplicity errors.
5. Missing names for all relations, poor choice of names for relations.

Textual Constraints:
1. Textual constraint doesn’t explain how visibility is derived from the components of the model.
2. Textual constraint is repetitive - already shown in the model.
3. Textual constraint is unnecessary - could have been shown in the model.
4. Textual constraint contradicts the model.

Explanation/Insights:
1. No explanation for unclear set.
2. No insights.
3. Incorrect explanation/insight.
Problem B2: Scheduling App Comparison

Example Models:

Doodle:

Congregar:
Example Textual Constraints:
1. Doodle
   a. Each participant can only specify one response per option.
2. Congregar
   a. Again, each participant can only specify one response per option.

Example Explanations:
1. Congregar
   a. The options for a poll can either be a specific date where time is not specified, or any arbitrary user defined string.
2. When2Meet
   a. ‘Date’ represents a specific date, and ‘generic day of week’ is a day of the week with no specific date.

Possible Insights:
1. Specifying options for polls in the different apps:
   a. When2meet allows users to make polls for weekly timeslots as opposed to polls for specific dates/times.
   b. Doodle only allows polls for specific dates and times.
   c. Congregar only allows the creator of the poll to offer specific dates as options, not times, or arbitrary string options. However, the usage of arbitrary strings can allow creators to make polls about any topic, not just scheduling.
2. How users respond to polls:
   a. Users respond to a when2meet poll by selecting a range of times with half hour granularity.
   b. Doodle allows users to respond to each date and time with a yes or no
   c. Congregar gives users the option to respond with preferred, neutral, or not preferred.
3. The uniqueness of how participants in a poll identify themselves and whether they can edit their responses:
   a. When2meet forces distinct names and allows users to change their response given that they know their password if one was set
   b. On a doodle, users can have the same name
   c. On congregar, users can have the same name
4. Doodle allows flexibility with different types of polls, such as polls with hidden responses, or ones where a user can only agree to one timeslot.

Common Errors:
Models:
   1. Failed to name all relations
   2. Many data models represented ‘creator’ and ‘participant’ as disjoint and immutable subsets of ‘user’. A user can be a creator of one poll but a regular participant in another poll, so there should not be a mutability marking.
   3. Some data models did not include the notion of events/polls/doodles, thus it was unclear that options or timeslots belong to a single event or poll.
   4. Other data models did not describe users as a set, making it unclear who responds to polls.
   5. Models generally did not show much differentiation of behavior between the three apps.
Insights
   6. Many insights described the applications as serving a similar purpose but differing in UI. Although true, this does not show a deeper understanding of any of the systems or that you’ve picked up on key differences in how each of the apps function.

Problem C: Designing an App

Example Model:
Example Textual Constraints:
   1. The “before” relation is specific to a party

Example Explanations:
   1. The before relation relates a request to the one preceding it

Possible Insights:
   1. The party id allows unique identification of each party
   2. Pending is the initial state of a request before it gets approved/rejected

Example Key Decisions & Alternatives:
1- Discussion of your design:

   Example of things you can discuss:
       1. what is a request , what does it contain, what are the possible states, why did you choose these states.
       2. what is a song, how do you represent it

   The discussion should be focused on the application’s behavior. An excerpt from your discussion can be the following :
“I introduced the Reject set to represent rejected suggestions, so that the app can remember which songs were rejected and automatically reject them if they are suggested again”
“I also introduced the pending state to represent the state of a song before it gets approved/rejected”
“The before relation reflects the relative order of a request and will help determine the order in which the songs are played/ added to the playlist”
“I separated all available songs from playlist because I want the playlist to only include songs approved by the host”

2- Comprehensive discussion of at least one alternative, including a detailed analysis of advantages and drawbacks.
   Example:
   1. Discuss a design where the host can suggest songs :
      Example: “I chose this design over a design where the host can suggest songs, because it simplifies matter. If I had allowed host to suggest songs, I would need to introduce a third entity ( eg : dj) to approve/reject them
   2. Discuss a design that does not include an order for songs
      “Example: Although a design without request orders could make things simpler, it would reduce the app’s functionality”

Common Errors:
Model:
   1. Not Labeling relations
   2. Having Redundant relations
   3. Not using subsets when needed
   4. Not showing the possible states of a request
   5. Not including requests/suggestions in your model
   6. Not distinguishing host from guest
   7. Not showing that a playlist includes accepted songs only
   8. Not showing the order of the song in the request
   9. Not including at least 2 insights

Design Analysis:
   10. Not discussing your design
   11. Not discussing design alternatives
   12. Not justifying design decisions

Problem D: Summarizing Your Fritter Design

Example Data Model:
Example Textual Constraints:
1. User cannot follow himself or retweet own tweets (also gave credit if other reasonable constraints on these issues were mentioned)

Example Explanations:
None were necessary for most models, some used slightly unclear wording so I looked for explanations for those sets

Possible Insights:
1. A possible insight arises from how retweeting was implemented. There were two ways this was mostly done: either a tweet keeps track of its retweeters, or a new tweet is created for every retweet (and some variations of this). Thus, the insight would be to explain whether when a user deletes a tweet, all retweets of that tweet are/are not deleted as well.

Example Database Description & Alternative:
User = { _id=ObjectId, username=String, password=String, follows=[User.ObjectId]}
Tweet = { _id=ObjectId, text=String, author=User.ObjectId, retweeter=User.ObjectId, creationTime=Date}
Alternative:
User = { _id=ObjectId, username=String, password=String, follows=[User.ObjectId], tweets=[{ text=String, retweeter=User.ObjectId, creationTime=Date}]}

Common Errors:
1. Missing markings for mutability or multiplicity (e.g. if the user who originally posted a tweet was not marked immutable, or if it wasn’t made clear that each tweet is associated with exactly one original poster).
2. Unnamed relations/missing relations
3. Many people included an alternate data model, instead of an alternate db representation.
4. Some people did not include insights or included “insights” that were restatements of something already presented in the data model (e.g. usernames are immutable)