

Teaching Data Modeling: Process and Patterns

Paul Wagner

Department of Computer Science

University of Wisconsin-Eau Claire

Eau Claire, WI 54701, USA

wagnerpj@uwec.edu

ABSTRACT

While competition for scarce space in a Database Systems course curriculum increases, the amount of time spent in many such courses on data modeling decreases. We instead recommend increasing the amount of time spent in the study of data modeling and encourage data model study beyond formalism syntax. We do this in an attempt to help computer science students better understand complex data domains and to help develop higher-level skills that serve them well in a job market threatened by the increased outsourcing of lower level programming jobs. We further recommend the study of process skills as part of data modeling, and develop the idea of data patterns to assist students in the development of advanced data modeling skills.

Categories and Subject Descriptions

K.3 [Computers & Education]: Computer & Information Science Education - *Computer Science Education*.

General Terms

Design, Management.

Keywords

Database, Systems, Data, Modeling, Teaching.

1 INTRODUCTION

The computer science education community has recently struggled with the idea of what material to teach in a Database Systems course [1]. This issue is made more difficult by several factors. First, the lasting importance of traditional topics (including data modeling, querying a relational database using Structured Query Language (SQL), and other long-standing topics such as concurrency and transaction processing) together with the emergence of new topics such as web-database applications,

object-oriented database management systems (DBMSs) and eXtended Markup Language (XML) means that the list of possible topics is quite large. Second, there has been more of a push toward “application oriented” approaches (e.g. [8]), which add in more practical and technical content relating to particular DBMSs and developing larger applications. Third, many four-year universities can only find room in the curriculum to teach one Database Systems course, and are not able to move some of these topics into an advanced Database Systems course.

One of the tendencies we have noted from the emergence of these new areas plus the pressure to fit more into the first Database Systems course is that the topic of data modeling increasingly is not given sufficient coverage in such courses. Students and faculty alike sometimes view the topic as “soft” (not really technical computer science) and perhaps more fitted to management information systems. Instructors who take such a view end up with a few weeks of data modeling discussion that focuses mostly on the syntax of common data modeling approaches such as Entity-Relationship (ER) or Extended ER (EER) modeling. Several traditional database textbooks reflect this approach by placing data modeling as one or two chapters in the midst of fifteen to twenty chapters (e.g. [4],[10]).

We see several significant problems with this approach. First, the study of data modeling builds the foundation for many other database system issues, including querying, creating database systems from a model and database administration. For example, students cannot be successful in querying large and complex databases if they don’t understand how to develop and read this structure. Second, in an era where many basic programming jobs are being offshore outsourced, we see data modeling as an important component of a broader computer science education that allows our graduates to find a niche in a competitive and, at least in some areas, narrowing job market. Data modeling is one of the important aspects of database systems that is still a practical skill but goes beyond the basic programming skills that are increasingly outsourced.

We thus suggest teaching a Database Systems course with a larger focus on data modeling. However, an instructor taking this approach cannot just increase the amount of time spent on the syntax of a particular data modeling formalism. We suggest expanding the coverage of the *process* of data modeling, and as part of this process we suggest the study of data *patterns* in this process. We have done this over many years of teaching database systems with good success.

2 BACKGROUND

Most Database Systems instructors and textbooks cover data modeling to some degree, but a variety of data modeling formalisms are used in this coverage of data modeling. While

most modern database textbooks use ER or EER (e.g. [5], [11], [12]), others use such formalisms as Logical Data Structures (LDS) [3] or semantic object formalisms (e.g. [7]). Also, UML is increasingly being used as a data modeling formalism (e.g. [9]).

At the same time, a variety of levels of process coverage are used. Many textbooks limit their coverage of data modeling process to discussion of one medium size example (e.g. a partially-simplified employee database). Such discussions generally cover the application of syntax within the model to show different syntactic features without discussing the process used to generate the model.

There are a few notable exceptions to this approach. One text suggests studying not only the syntax of data modeling, but also focusing on such topics as developing the skills necessary for reading data models, the need for and the process involved in conversations with users to extract information and resolve conflicts, and the overall process needed to develop a complete and accurate data model [3]. It is primarily based on these ideas that our approach has evolved.

We also have taken inspiration from the design pattern community, most prominently developed for software development in general through [6]. We find that teaching students to recognize patterns in data assists them in the process of making data models in new domains, as they see similar shapes evolving across old and new data domains.

3 OUR APPROACH

As noted above, the approach we use to teach data modeling focuses on a combination of teaching the process by which one develops a data model along with illustrating the common patterns that data modelers and other database system workers see across many different data domains. We focus on the details of these two major ideas below.

While some may say that coverage of the actual process of working with users is best left for another course such as software engineering or analysis and design, we think that it is essential for students working with data to develop process skills in the context of working specifically with data. It's important for students to see the connection between a good process for data modeling and the likelihood of generating a good data model.

4 PROCESS

Teaching students about the process involved in data modeling involves work with several subtopics:

- developing a set of process rules to ensure the development of an accurate and complete model
- practice of the above with larger, realistic problem domains and (preferably) real users

We address each of these in separate sections below.

4.1 Process Rules

A good set of data modeling process rules can help ensure that a data model is both complete and accurate. While students at first may think that they can just ask a single knowledgeable user to “tell me everything you know about this area”, they quickly realize that life as a data modeler is not this easy. The main problems that a non-structured approach encounters are:

- incomplete answers from users

- conflicting answers from multiple users
- too much / irrelevant information from users
- inability of the data modeling team to synthesize the information received into a complete and accurate data model

We thus strive to show them that a structured approach to data modeling is important. This involves the development of the following rules related to the above issues:

- Data modelers need to talk at length to multiple users with differing perspectives to make sure that their understanding of the data domain is complete.
- Data modelers need to be able to resolve conflicts by talking to multiple users and helping those users (not the data modeler) come up with a consensus answer.
- Data modelers need to ask specific questions of users to make sure that the users focus on the data issues (not process, system or performance issues) and that those users answer questions in a way that helps the data modeler improve the data model.
- Data modelers need to have a methodology that supports evolution of the data as additional data is received.

We deal with the first three issues/points primarily through discussion and looking at case studies where similar problems have occurred (e.g. see the conversation chapter and story interludes in [3]). Helping the students see the need for a data modeling methodology and teaching the steps for this is something that takes more time, and is discussed in more detail below.

4.1.1 Methodology Details

There are a number of possible methodologies for approaching data modeling. Some are part of larger software engineering methodologies (e.g. [2]). Others are specifically focused on data modeling (e.g. see Ch. 18 in [3]). We approach this along the lines of the latter, in that we focus on teaching a controlled evolution of our data model through structured communication with domain users. Important steps in any approach include:

- generation of a base set of entities or classes
 - Example: A registration system may start with a Student entity, a Course entity, an Instructor entity, and a Room entity
- establishing a proper identifier attribute (or set of attributes) for each entity/class
 - Example: A Student cannot be identified by their name as there may be two students with the same name; an arbitrary Student ID number or string is needed.
- thinking completely and expansively about possible non-identifying attributes, and evolving complex attributes into their own entity/class
 - Example: A Student apparently has a telephone number and an address. On second examination, the student actually has multiple telephone numbers (e.g. land, cell) and multiple addresses (e.g. local, permanent). TelephoneNumber and Address should evolve into separate

entities/classes. Data modelers should think expansively

- looking for fundamental relationships between two (or possibly more) of those entities/classes
 - Example: Student is related to Course, Course is related to Instructor, Course is also related to Room
- establishing the cardinality for each end of a relationship
 - Example: Many instances of Student are related to one instance of Course; Many instances of Course are related to one instance of Student
- determining if the relationship actually evolves to a new entity/class, based on the complexity of the data attached to that relationship
 - Example: The M-M relationship above can be better represented as a Registration entity, where one Student has many instances of Registration associated with it, and one Course has many instances of Registration. This allows the storage of, for example, a Grade attribute and a RegisterTimeStamp attribute.
- looking for relationships between any new entities/classes created and other previously-created entities
 - Example: The newly created Registration entity has a relationship with a Term entity
- ensuring that attributes are positioned in the correct entity/class (this essentially leads to their data model having at least 3rd Normal Form).
 - Example: Course should not contain the attribute NumberOfCredits if a variable number of credits is possible. Data modelers should look toward another entity/class (e.g. Registration) where the NumberOfCredits attribute can be appropriately located.
- looking for secondary relationships between entities/classes
 - Example: While a Student is indirectly related to an Instructor through Course, there is a more direct relationship for advising that must be added.

We teach students to iteratively apply the above methodology steps (in somewhat more detail) until all possible entity relationships have been considered. While the above seems fairly intuitive, getting student data modelers to consistently and thoroughly apply these steps is difficult and takes practice.

Several common problems occur during this process, mostly related to the details and the subtleties of a given domain. For example, student data modelers miss attributes and relationships. They also misjudge cardinalities (often based on users who state that “there’s USUALLY one of entity X per entity Y”). Student data modelers often don’t properly evolve complex relationships into new entities/classes. Any of these mistakes can cause the database they eventually implement to not be able to support all of the needs of the users. It is for this reason that we think that students need to practice data modeling numerous times, on successively larger problem domains.

4.2 Practice in Larger, Realistic Domains

In classrooms where data modeling is a one- to two-week topic, students do not normally get more than a few opportunities to practice data modeling. We have found that increased data modeling practice increases the quality of the final class project (which also contains a data modeling component), and presumably increases the quality of future data modeling work.

We ask the students to practice data modeling each class session – both in class and with a series of increasingly complex homework assignments. Over several weeks the students get to develop four to five data models. The last and largest data modeling homework goes through two iterations; i.e. the students get an opportunity to update/improve their first submission. This additional practice and the opportunity for refinement help the student data modelers see that more subtle data modeling issues exist and that first attempts are often incomplete. We spend a significant amount of class time reviewing issues raised by the first submissions, showing how assumptions can lead to omissions, and how lack of disciplined refinement can lead to incomplete or incorrect models.

We use a last homework assignment with approximately 30 or more entities (e.g. a basic airline route/reservation system). While not large by industrial standards, a model of this size indicates to the student data modelers the level of complexity found in the real world, and illustrates all of the process issues previously discussed.

For a final team project, we also have the students start by developing a data model, and again give them an opportunity to revise it in the second phase of this project. By this time, they are developing significant skill as data modelers, and the models they create generally do not require significant modifications during the implementation phase, indicating some level of success.

Several times we have brought in real-world users for these projects. Where possible, this is very helpful, as students see the difficulty of the information gathering process. We would like to get multiple users to present their perspectives, but unfortunately this is not often practical.

To supplement our discussion of process, we illustrate how the data modeling process can be made easier by recognizing patterns in data that exist across data domains. We have found that the recognition of patterns in data is the second major component that supports student data modelers in their transition to becoming professionals in this area.

5 DATA PATTERNS

While some authors have directly applied software design patterns to data modeling by focusing on particular domains and looking more toward the eventual database application (e.g. application of Singleton, Flyweight, Composite and other design patterns in [9]), we instead try to look for the general patterns of data that commonly occur in many different data domains (though some of the Structural data patterns (e.g. Composite) from [9] fit well with a general data pattern view.) Thinking of data modeling in terms of repeatedly applying a number of commonly occurring data patterns makes data modeling significantly more accessible to students, and we find that they are better able generate high quality data models as a result.

While a complete enumeration of all possible data patterns is beyond the scope of this paper (a much more complete description is found in [3], using the terminology of “shapes”), we can

enumerate a number of major patterns that are of significant importance to student data modelers. We instruct students to watch constantly for these patterns and apply them in new domains.

The data patterns we discuss are based on the number of entities involved, the relationships between those entities, and the cardinalities on those relationships. We use the LDS notation to illustrate examples of each pattern below, noting parallels with UML language in parentheses; however, similar models could be constructed in ER and other modeling formalisms that illustrate the same data patterns. In LDS notation, bars under attributes or on relationship lines indicate that the attribute or relationship contributes to the identifier for that entity, and the “chicken foot” on an end of a relationship indicates “Many” for cardinality.

- Aggregation/Containment (HAS-A) – 2 or more entities, 1-M relationships at each level, “Many” cardinality always on same side. In Figure 1, a Company has multiple Divisions, and a Division has multiple departments. This pattern thus generalizes over N levels to represent an instance hierarchy.

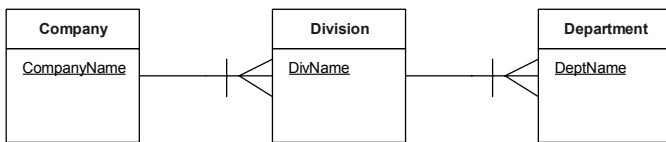


Figure 1

- Intersection – 3 entities, two 1-M relationships, “Many” cardinality on the inside. In Figure 2, Capability is the entity holding information about an Employee/Skill pair, similar to a relationship in ER modeling terms.

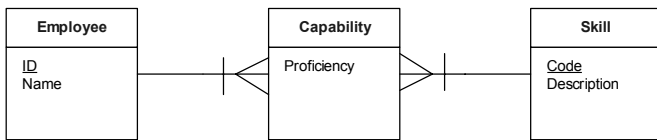


Figure 2

- Differing Aspects of Same – 3 entities, two 1-M relationships, “Many” cardinality on the outside. In Figure 3, a Student has multiple phones and multiple addresses, but there is no direct relationship between phones and addresses.

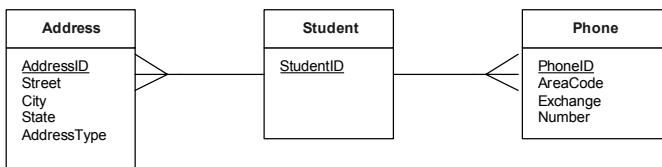


Figure 3

- Subordinate or Subtype (IS-A, or inheritance/generalization) – 2 or more entities, 1-1 relationships as subtypes (indicated in LDS by an identifier bar on the subtype relationship descriptor (representing the subordinate relationship) and the “be” labels (representing that the entities are two different representations of the same concept). In Figure 4, with the “subordinates out”, a Full-Time Employee is one subtype of an Employee, and a Part-Time employee is another subtype of an Employee at the same level as Full-Time Employee.

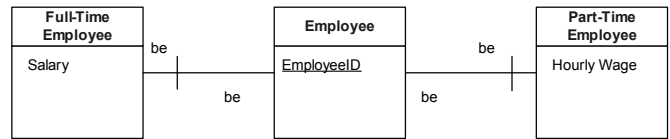


Figure 4

In Figure 5, the pattern is “subordinates across” – i.e. the subordinate descriptor is always on the same side of the relationship, and we have an N-level inheritance hierarchy; here, a University Person has one subtype Employee (could have others, such as Instructor and Student), and Employee in turn has one subtype Full-Time Employee (and, as shown above, could have others such as Part-Time Employee).

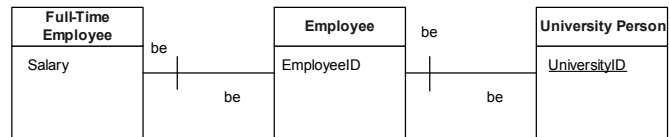


Figure 5

- General Association – 2 entities, 1-1 relationship, not a “be” relationship, not a subordinate/subtype relationship. In Figure 6, one Employee manages one Department. In this case, the association might be considered Aggregation (HAS-A) in UML. If the entities were changed to Simulator and RandomNumberGenerator (used by the Simulator to generate data), the association is more general.

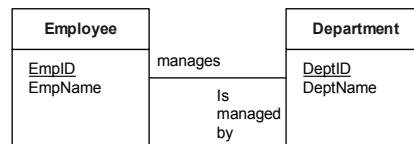


Figure 6

- Reflexive – 1 entity, various cardinalities. In Figure 7, an Employee is related (as a manager) to another Employee – each Employee has one managing Employee, but one managing Employee can manage multiple instances of Employee.

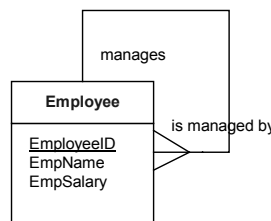


Figure 7

While the data patterns illustrated above are content neutral, there are also non-content neutral data patterns (“recipes” in [3]) for common data structures. We also encourage our students to watch for these data patterns as indicators of structural characteristics of the data; e.g. is a particular collection best regarded as a set, a multiset, a sequence, or a graph/network? While there are multiple data patterns that indicate variants on the same data structure, student data modelers can begin to see how common data structures appear in various data domains. Several examples follow.

- Set – in Figure 8, assuming an arbitrary set identifier, this is a concrete example of the Intersection data pattern above. A work team and a group of people are concrete examples.

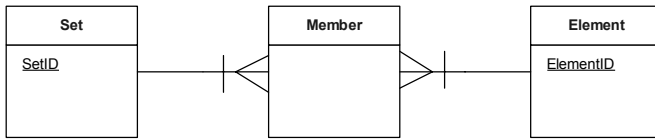


Figure 8

- List or Sequence – in Figure 9, the addition of a Position attribute in the center Entity gives us ordering of elements. A protein sequence and a printer queue are examples.

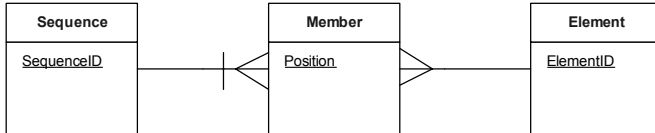


Figure 9

- Tree – in Figure 10, the content of an N-ary tree of unique values is represented reflexively (and recursively.) An employee management chart and game trees are examples.

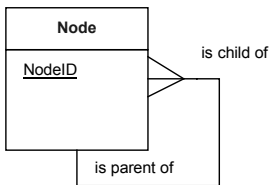


Figure 10

- Graph/Network – in Figure 11, the content of a Network is represented. With the removal of the Edge weight, this would be a graph. The internet and airline networks are examples.

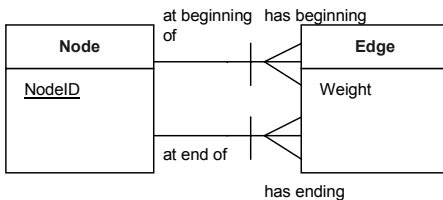


Figure 11

Teaching students to note such patterns in data, and recognize that they may indicate a certain type of structure, helps them to develop more accurate data models and better understand the structure inherent in complex data domains.

6 IMPLEMENTATION

We have found that detailed study of the process of data modeling can be accomplished in a first Database Systems course in approximately four to five weeks. As such, we were also faced with what material to remove. Personally, we have eliminated such topics as detailed discussion of indexing strategies, and we reduced our discussion of theoretical normal forms (finding that

high quality data models generated with the above process and patterns automatically produce models with at least 3rd Normal Form). Other instructors may make different decisions, but we find that we can increase the amount of time we spend on data modeling while including the topics we consider most important.

7 CONCLUSIONS

We have used this approach over approximately ten offerings of introductory Database Systems courses at two different universities. While we have not done formal studies of the results of our process, we have found that the quality of later data models and final projects has increased by in turn increasing the quality and quantity of time we spend on data modeling. Alumni report back that this is a very practical and useful course that helps prepare them for real-world software development. Overall, students appear to have a better understanding of data systems.

8 ACKNOWLEDGEMENT

The pedagogical strategy and many of the teaching tactics put forth in this paper are based on and evolved from the accumulated work of John V. Carlis. We acknowledge the large contribution he has made to the development and teaching of data modeling, and his work has inspired us to use and develop these ideas.

9 REFERENCES

- [1] Adams, Elizabeth, Granger, Mary, Goelman, Don, Ricardo, Catherine, Panel on “Managing the Introductory Database Course: What Goes In and What Comes Out?”; Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Educations, March 2004.
- [2] Bruegge, Bernd, and Dutoit, Allen, “Object-Oriented Software Engineering – Conquering Complex and Changing Systems”, Prentice Hall, 2000.
- [3] Carlis, John V. and Maguire, Joseph, “Mastering Data Modeling: A User-Driven Approach, Addison-Wesley, 2001.
- [4] Date, C.J., “An Introduction to Database Systems”, 7th Edition, Addison Wesley, 2000.
- [5] Elmasri, Ramez, and Navathe, Shamkant, “Fundamentals of Database Systems”, 3rd Edition, Addison Wesley, 2000.
- [6] Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John, “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995.
- [7] Kroenke, David, “Database Processing: Fundaments, Design and Implementation”, 7th Edition, Prentice Hall, 2000.
- [8] Lewis, Philip, Bernstein, Arthur, and Kifer, Michael, “Databases and Transaction Processing: An Application-Oriented Approach”, Addison Wesley, 2002.
- [9] Muller, Robert, “Database Design for Smarties: Using UML for Data Modeling”, Morgan Kaufman, 1999.
- [10] O’Neil, Patrick and O’Neil, Elizabeth, “Database Principles, Programming and Performance”, Morgan Kaufman, 2001.
- [11] Ramakrishnan, Raghu and Gehrke, Johannes, “Database Management Systems”, 3rd Edition, McrGraw Hill, 2003.
- [12] Sillberschatz, Abraham, Korth, Henry, and Sudarshan, S., “Database System Concepts”, 4th Edition, McGraw Hill, 2002.