It turns out that this kind of redundancy corresponds to another kind of constraint, called a *MultiValued Dependency* (or *MVD*). If we desired, we could revise our normalization criteria to deal with MVDs. The resulting schema would be in a form called *Fourth Normal Form* (*4NF*).

Unfortunately, there are other examples of redundancy that are not caught by MVDs. In the early years of the relational model, researchers developed increasingly more complex dependencies and normal forms, with each new normal form handling a situation that the previous normal forms could not handle. The goal was to be able to have a definition of "normal form" that always produced well-designed schemas.

That goal has been largely abandoned. Database design is still more of an art than a science. In the end, it is up to the database designer to craft an appropriate class diagram. Constructs such as FDs can help, but they cannot substitute for intelligent design and careful analysis.

3.7 Chapter Summary

- The information in a database is often vital to the functioning of an organization. A poorly designed database can be difficult to use, run slowly, and contain the wrong information.
- A good way to design a database is to construct a *class diagram*. A class diagram is a high-level representation of the database that focuses on what information the tables should hold and how they should relate to each other.
- A class diagram is built out of *classes* and *relationships*. A class denotes a table, and is represented in the diagram by a box. A relationship denotes (to some extent) a foreign key connection between two tables, and is represented by a line connecting two boxes.
- Each side of a relationship has a *cardinality* annotation that indicates, for a given record, how many other records can be related to it. An annotation may be a "1", which indicates exactly one related record; a "*", which indicates any number of related records; or a "0..1", which indicates at most one related record. A relationship is said to be *many-many, many-one,* or *one-one,* depending on the annotations on each side.
- The "1" annotation is called a *strong annotation,* because it asserts that an entity cannot exist without a related entity. The other annotations are *weak annotations*. A foreign key corresponds to a weak-strong relationship.
- The algorithm of Figure 3-4 generates tables from a class diagram. This algorithm generates one table per class. The table contains a field for each of the class's fields, and a foreign key field for each relationship where the class is on the weak side.
- A relationship with two weak sides is called a *weak-weak* relationship. One example of such a relationship is a many-many relationship. The transformation algorithm does not generate tables for weak-weak relationships. Instead, we must first *reify* the relationship. We replace the relationship by a new class, which has weak-strong rela-

tionships to the other classes in the relationship. This new class can be thought of as "splitting" the weak-weak relationship. It improves the class diagram by making explicit a concept that was previously implicit.

- A *strong-strong* relationship has "1" annotations on both sides. Such a relationship denotes a single conceptual record that has been split up between the two classes, and can be removed by combining the two classes into a single class.
- A well-designed class diagram should contain no inadequate relationships, and no redundant relationships. The presence of an inadequate relationship means that the database will not be able to hold some desired data. The presence of a redundant relationship means that the database will contain the same fact in multiple places. Such redundancy can allow the database to become inconsistent.
- A *functional dependency (FD)* is a constraint that generalizes a superkey. An FD asserts that two records having the same LHS values must also have the same RHS value. An FD is *key-based* if its LHS values form a key.
- A table that contains a non-key-based FD violates the properties of a well-designed database. If all FDs are key-based, then the table is in *Boyce-Codd Normal Form (BCNF)*.
- Removing a redundant field can make it impossible for a table to enforce a many-one constraint. If we want the database system to be able to enforce all such constraints, we can use *Third Normal Form (3NF)* instead of BCNF. 3NF is less restrictive than BCNF: A table in 3NF can have a non-key-based FD, provided that its RHS field belongs to some key.
- Database design is more of an art than a science. Normalization helps to find and fix problems in the class diagram, but it cannot guarantee a good diagram. In the end, the database designer needs to apply intelligent design and careful analysis.

3.8 Suggested Reading

The task of designing a database has three stages: First we construct the requirements document, then we model it using diagrams, and finally we map the diagrams to tables. This chapter focuses on the latter two issues. The first issue is the purview of Systems Analysis, and involves topics such as developing use-cases, interviewing users, consolidating and standardizing terminology, etc. Hernandez [2003] provides an excellent introduction to this aspect of design. In addition, Hay [1996] develops example diagrams for many common business situations, so that a designer can see how others have designed similar databases.

Kent [2000] is a thought-provoking philosophical examination of the various ways that people conceptualize data. It argues that any given database can have more than one "correct" design, and makes it clear that database design is indeed far more difficult than we think.

This chapter uses UML as its diagramming language. Historically, designers have used many different diagramming languages, most notably variations on what are called *Entity-Relationship Diagrams*. Teorey [1999] gives a comprehensive introduction to

database design using these diagrams. The UML language was developed as a reaction to the proliferations of these language variants. UML stands for "Universal Modeling Language," and the hope was that it would become the standard design language, both for databases and for software. Of course, such high aspirations lead to feature creep, and UML has become very large. The diagrams used in this chapter are known as *class diagrams,* and comprise just one out of thirteen possible UML diagrams. A short introduction to all of UML appears in Fowler and Scott [2003], as well as online at *www.agilemodeling.com/essays/umlDiagrams.htm.*

Our treatment of class diagrams in this chapter considers only the most common kinds of relationship. Other kinds of relationship can occur, such as subtypes and weak entities. These issues are considered in Teorey [1999] and Fowler and Scott [2003].

This chapter takes a somewhat narrow approach to normalization, primarily because it is our belief that most of normalization theory is irrelevant to modern design practice. A good overview of all of the normal forms and their purpose appears in Jannert [2003].

3.9 Exercises

CONCEPTUAL EXERCISES

3.1 Verify that there are no redundant relationships in Figure 3-12.

3.2 Consider the online bookseller tables in Exercise 2.7. Draw a class diagram that corresponds to these tables.

3.3 Consider a relational schema. Suppose you use the algorithms of Figure 3-3 and Figure 3-4 to translate it to a class diagram and then back to a relational schema. Do you get the schema that you started with? If not, what is different?

3.4 Consider a class diagram, all of whose relationships are weak-strong. Suppose you use the algorithms of Figure 3-3 and Figure 3-4 to translate it to a relational schema and then back to a class diagram. Do you get the diagram that you started with? If not, what is different?

3.5 Consider the class diagram of Figure 3-11. Show that the relationship between STUDENT and GRADE_ASSIGNMENT is not redundant.

3.6 Transform the class diagram of Figure 3-12 to a relational schema (that is, without choosing attributes). What fields do the tables have? How does your schema compare to the schema of Figure 2-1?

3.7 Suppose we want the university database to contain the following information: The room and time when each section meets, the semester of each section (not just its year), the faculty advisor of a student, and the department to which each professor is assigned, and whether or not the student graduated.

a) Modify the class diagram in Figure 3-12 appropriately. Reify any weak-weak or multi-way relationships.

b) Choose appropriate attributes for the classes, and transform the resulting diagram to a relational schema.

b) Your answer to part (a) should have a redundant relationship that is necessary for the enforcement of a many-one constraint. Which relationship is it?

c) Without removing that redundant relationship, choose attributes for your class diagram and transform it to a relational schema.

d) Explain why your relational schema is in 3NF but not in BCNF.

e) Do you think that the 3NF schema is a good design, or would it be better to remove the redundancy and get a BCNF schema? Explain.

3.21 The database design of Figure 3-23 is really bad. Improve it.

PROJECT-BASED EXERCISES

The following exercises ask you to design databases for different applications. For each exercise, you should follow the design methodology completely: Create a class diagram, generate a relational schema from it, determine applicable FDs, and verify that the schema is in normal form.

3.22 Design a database to hold the CD and song information described in Section 3.1.

3.23 An insurance company wants to maintain a database containing the following information relating to its automobile policies:

- Each policy applies to exactly one car, but several people may be listed as drivers under the policy. A person may be listed under several policies.
- People are identified by their social-security number, and have names and addresses; cars are identified by their registration number, and have model names and years.
- Each policy has an identifying number and a list of coverage types (such as collision, comprehensive, damage to others' property, etc.); for each coverage type the policy has a maximum amount of coverage and a deductible.
- The database also contains information about accidents covered by each policy. For each accident, the company needs to know the car and driver involved, as well as the date of the accident and the total amount of money paid out.

3.24 A video-rental store wants to maintain a database having two major components: The first component lists general information about movies; the second component lists information about which movie videos the store carries, the store's customers, and what movies have been rented by those customers. The specifics are as follows:

- Each movie has a title, year, rating, running time, and director name. There may be several movies in the same year having the same title. A movie can be assigned to zero or more genres (such as "comedy," "drama," "family," etc.). Each actor has a name, birth year, and sex. An actor may appear in several movies, but will play at most one role in any movie.
- The store carries videos for some of these movies. For each video it carries, the store keeps track of how many copies it owns. A customer cannot rent a video without being a member. The store keeps track of the name, address, and membership number of each customer. The store also keeps track of every

movie ever rented by each customer. For each rental, the store records the rental date and the return date. (The return date is null if the video is still being rented.)

- Customers can rate movies on a scale from 1 to 10. The ratings for each movie should be available, identified by which customer gave what rating. Similarly, the history of ratings for each customer should be available. (Should customers be able to rate only movies that they have rented? You decide.)

3.25 A hospital keeps a database on patients and their stays in the hospital. The specifics are:

- Patients are assigned a specific doctor that handles all of their hospital stays.
- Each doctor has a name and an office phone number.
- For each patient visit to the hospital, the database records the date the patient entered, the date the patient left, and the room that the patient is staying in. If the patient is still in the hospital, then the "date left" field is null.
- The patient is charged per day spent in the hospital. Each room has a price per night.

3.26 The commissioner of a professional sports league has asked you to design a database to hold information on the teams in the league, the players who have played for the teams, the position that they played, and what the result of each game was. This database will hold information over many years, possibly back to the beginning of the league. The following constraints hold:

- A player can play only one position, and plays that position throughout his/her career.
- A player may play for several teams over his/her career. A player may change uniform numbers when traded to a new team, but keeps that number while with that team.
- Each game has a home team and an away team. There are no ties.