1 Introduction

Welcome to the UML Database Modeling Workbook! This book can teach you database modeling through concise explanation, practice with exercises, and a series of self-assessment tests. The book is targeted at software professionals, such as the following:

- **Application architects**. Their guidance for applications often revolves about models and data models.
- **Enterprise architects**. They reach beyond a single application and address the needs of an enterprise. There is no better way to harmonize applications than by modeling them deeply and aligning the models.
- **Data modelers**. The book can teach basic skills to novice data modelers. Experienced data modelers can pick up a new notation (the UML) by leveraging their existing knowledge.
- **Students**. The book is suited for commercial data modeling courses as well as university courses.

1.1 Data and Databases

Data is a corporate resource. It's the lifeblood of most organizations and critical to their endeavors. Data provides an organization's memory — its customers, competitors, products, orders, employees, equipment, locations, objectives, plans, sales, and expenses.

Data is found in many forms. A simple technique is to store data in files; this can work well for small quantities. A more sophisticated approach is to store data in a database. Database software carefully manages data, offering multi-user access while protecting against accidental loss. Database software can also enforce constraints that boost the quality of data, reducing corruption from data entry mistakes and programming errors. Databases provide flexible data access that reduces development cost while achieving fast performance.

The dominant database paradigm today is the relational database. A relational database presents data in the form of tables. Normally each table has a primary key that uniquely identifies individual records. Tables connect via foreign keys, a reference to a unique identifier, usually the primary key. A major feature of relational databases is referential integrity — developers can declare foreign key references and system software keeps the references valid.

Relational databases are certainly powerful, but they can be difficult to use. The notion of a table seems simple, but a database is anything but simple. Hundreds or thousands of tables interconnect via foreign keys to form a complex graph. This is why data modeling is important. Data modeling lets stakeholders visualize the structure of a database, understand the critical concepts, consider alternate representations, and then summarize the results in a meaningful diagram.

1.2 Models and Data Models

A ***model*** is a representation of some aspect of a problem that lets you thoroughly understand it. A ***data model*** is a model that describes how data is stored and accessed, usually for a database. Developers must understand a problem before attempting a solution. Most database models are expressed as graphical diagrams and, by their form, appeal to human intuition. There are many reasons for constructing models:

- **Better quality**. Your application can be no better than the underlying thought. ACM Turing award winner Fred Brooks contends "that conceptual integrity is the most important consideration in system design." [Brooks-1995]
- **Reduced cost**. You can shift your activities towards the relatively inexpensive front end of software development and away from costly debugging and maintenance.
- **Faster time to market**. It takes less time to deal with difficulties at the conceptual stage than to deal with them when software has been cast into code.
- **Better performance**. A sound model simplifies database tuning.
- **Communication**. Models reduce misunderstandings and promote consensus among developers, customers, and other stakeholders.
- **Fewer mistakes**. Rigorous modeling improves the quality of the data. You can weave constraints into the fabric of a model and the resulting database.

The bottom line is that models provide the means for building quality software in a predictable manner.

This book uses the Unified Modeling Language (UML) as its primary notation. In addition, we use Information Engineering (IE) to clarify the meaning of UML constructs and illustrate aspects of database design.

1.3 The Unified Modeling Language (UML)

The ***Unified Modeling Language*** (***UML***) [Blaha-2005] [Rumbaugh-2005] [OMG – Object Management Group] is a graphical language for modeling software. The UML has a variety of notations of which one (the class model) concerns databases. The UML ***class model*** specifies classes (entity types) and their relationship types. The resulting model sets the scope and level of abstraction for subsequent development. There are several benefits of using the UML for database modeling:

- **Communication**. The UML provides a means for communicating with customers. The UML is more concise than traditional database notations and defers database details. Thus the UML suppresses database gore such as primary keys, foreign keys, indexes, and referential integrity that is important for implementation but is not needed for specifying a problem. This helps customers focus on requirements and scope.

- **Abstraction**. The UML lets modelers focus on the essence of a problem — the key concepts and how they relate — and avoid distracting notational clutter. The resulting concise models help developers envision modeling alternatives and explicitly consider their options, such as different data modeling patterns [Blaha-2010].
- **Precision**. The UML class model has several helpful features (most notably association classes and qualifiers) not found in most other database notations.

The UML is popular with programmers, but is used less often by database developers — this is the primary drawback of the UML. One reason is technical — the UML creators loaded the UML with many programming details that are extraneous to databases. Another reason is that few UML tools support database design. The irony is that the programming jargon is superficial, and in reality, the UML has much to offer for database applications.

1.4 Information Engineering (IE)

Information engineering (***IE***)* is a prominent database modeling notation that has been in use for many years. IE was popularized by James Martin and Clive Finkelstein in the 1980s and is oriented towards database design. IE focuses on details such as tables, keys, and indexes. This book uses IE as a secondary notation because it is familiar to many database practitioners. Furthermore, IE's attention to database detail is helpful for explaining nuances of the UML.

IE lacks a standard notation. Rather, there are several variants that express the same underlying concepts. This book uses the ERwin database modeling tool as the arbiter of IE notation.

1.5 Using UML and IE Together

The database literature traditionally distinguishes among three kinds of models:

- ***Conceptual model*** — focuses on major entity types and relationship types. Provides a high-level overview. Has no attributes.
- ***Logical model*** — fleshes out the conceptual model with attributes and lesser entity types.
- ***Physical model*** — converts the logical model into a database design. The emphasis is on physical constructs such as tables, keys, indexes, and constraints.

The UML is effective for conceptual and logical data models. IE is effective for physical data models.† The notations complement each other's strengths and are effective to use together.

Our development process uses both notations. We start with the UML to conceive the abstractions of an application and converge with customers on content and scope. Then we translate the ideas into IE and add database details. From IE, we generate database code. The

* Please note: In this book, IE is an acronym for Information Engineering (and is not referring to Internet Explorer).

† Strictly speaking, UML and IE are equally good for conceptual modeling. But the UML is usually better for logical modeling, which comes after conceptual modeling.

UML is good for abstract modeling but not for database design. IE is good for database design but not for abstract modeling. Both notations are useful, but each has its place. We maintain both models as revisions occur, and use agile development to build an application as a series of iterations.

1.6 Chapter Summary

A model is an abstraction of a problem that lets you thoroughly understand it. A data model is a model that describes how data is stored and accessed, usually for a database.

This book uses the UML class model as its primary notation, and IE as a secondary notation. The UML is effective for conceptual and logical data models. IE is effective for physical data models. The notations complement each other's strengths and are effective to use together.

Bibliographic Notes

The UML class model is one of a score of approaches descended from the seminal entity-relationship notation of [Chen-1976]. The class model is essentially just another Chen dialect, but one that has the backing of a standard (sponsored by the Object Management Group). [Elmasri-2011] is a good general database reference. Hernandez, Hoberman, and Teorey are good data modeling references. [Burbank-2011] has helpful advice for using the ERwin database modeling tool.

References

[Blaha-2005] Michael Blaha and James Rumbaugh. *Object-Oriented Modeling and Design with UML, Second Edition*. Upper Saddle River, NJ: Prentice Hall, 2005.

[Blaha-2010] Michael Blaha. *Patterns of Data Modeling*. New York: CRC Press, 2010.

[Brooks-1995] Frederick P. Brooks, Jr. *The Mythical Man-Month, Anniversary Edition*. Reading, Massachusetts: Addison-Wesley, 1995.

[Burbank-2011] Donna Burbank and Steve Hoberman. *Data Modeling Made Simple with CA ERwin Data Modeler*. Bradley Beach, New Jersey: Technics Publications, 2011.

[Chen-1976] PPS Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems 1*, 1 March 1976.

[Elmasri-2011] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, Sixth Edition*. Boston: Addison-Wesley, 2011.

[Hernandez-2013] Michael J. Hernandez. *Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design, Third Edition*. Boston: Addison-Wesley, 2013.

[Hoberman-2009] Steve Hoberman. *Data Modeling Made Simple, Second Edition*. Bradley Beach, New Jersey: Technics Publications, 2009.

[OMG] The Object Management Group. www.omg.org

[Rumbaugh-2005] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Boston: Addison-Wesley, 2005.

[Teorey-2011] Toby Teorey, Sam Lightstone, Tom Nadeau, and H.V. Jagadish. *Database Modeling and Design: Logical Design, Fifth Edition*. New York: Morgan Kaufmann, 2011.

2

A Data Modeling Example

This chapter presents a sample data model and explains its meaning. Subsequent chapters discuss UML and IE* modeling constructs and use portions of the example for illustration.

2.1 UML Data Model

Figure 2.1 shows a sample UML data model for an online retail Web site. Online retail is targeted at *Customers*. Each *Customer* has a name (*customerName*) as well as a *loginName* and password (*encryptedPassword*) for identification. A *Customer* can place many *Orders* and an *Order* is specific to a *Customer*.

Each *Order* has a unique *orderNumber*, as well as an *orderDateTime*, *shippingMethod* (post office, UPS, Fed Ex), *taxAmount*, *shippingHandlingAmount*, and *totalAmount*. An *Order* may have a different *shipping Address* and *billing Address*. There can be multiple *Payments* for an *Order*; each *Payment* can be by a credit card, a check, money order, or some other *paymentType*. An *Order* may consist of multiple *OrderItems*, each of which is identified by an *itemNumber* within an *Order*. Each *OrderItem* has a *quantity* of one or more.†

An *OrderItem* pertains to a *VendedProduct*. A *VendedProduct* is a *Product* sold by a particular *Merchant*. The *price* of a *Product* can vary by *Merchant* and *condition* (such as new, good condition, and worn).

Retail Web sites support a wide range of *Products*, of which the model shows three kinds: *Book*, *Electronics*, and *Automotive*. Many additional kinds of *Product* are possible (*OtherProduct*). The model shows some details for *Book* — the *ISBN*, *publishedDate*, *format* (hardcover or softcover), *publisherName*, and *language*. A *Book* can have multiple *Authors*,

* Please note: In this book IE is an acronym for Information Engineering (and not Internet Explorer).

† The relationship type between *Order* and *OrderItem* is a qualified association. The qualifier is *itemNumber*. The qualified association indicates that the combination of an *Order* and an *itemNumber* denotes a single *OrderItem*. See Chapter 7 for further explanation.

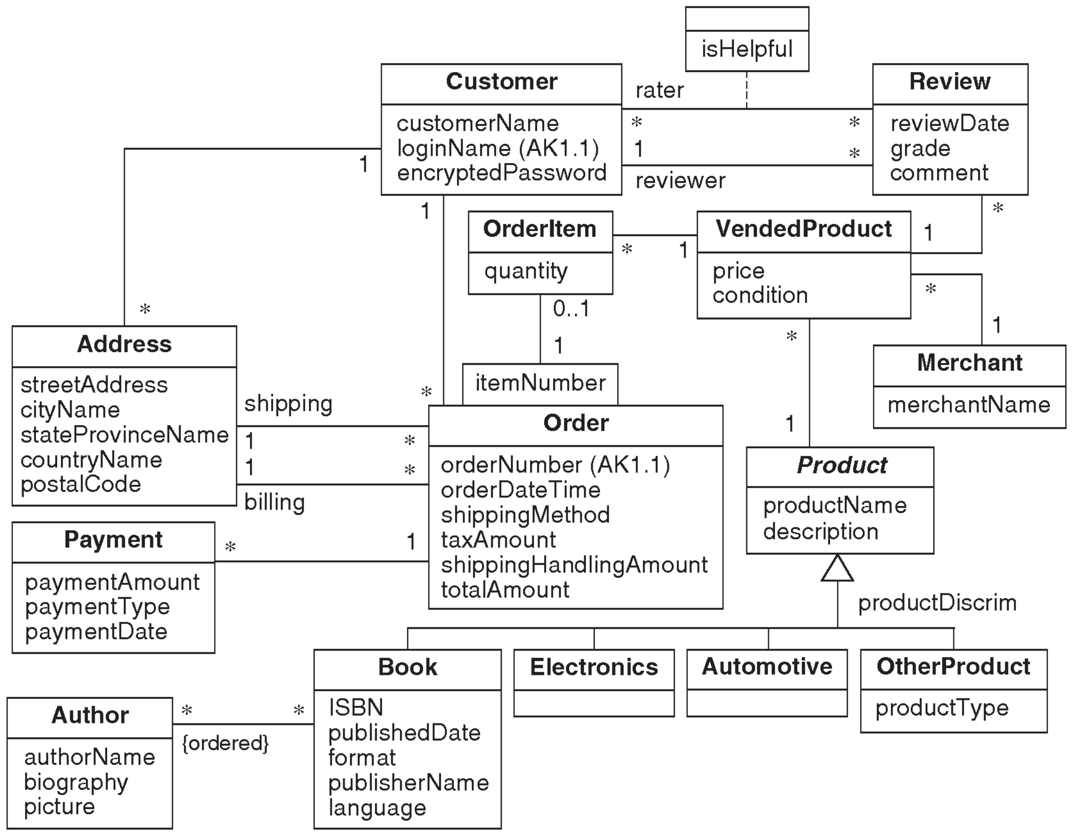

Figure 2.1 An example of a UML logical data model. This model provides the basis for most UML examples throughout the book.

ordered by their significance. *Authors* can post information about themselves, including their name (*authorName*), *biography*, and *picture*.

Customers can not only place *Orders*, but they can also submit *Reviews* for *VendedProducts*. In addition a *Customer* can rate someone else's *Review* and indicate if it *isHelpful.*

In business discussions, we often use the UML notation as we have here, without prior explanation. Most business staff find the UML to be intuitive and understandable. During interactive sessions, they guide the content of a model with a modeling expert leading the way. The modeling sessions have the side benefit of causing different departments to speak to each other and agree on application requirements and scope. See [Blaha-2011] for a demonstration of interactive UML data modeling.

2.2 IE Data Model

Figure 2.2 is the counterpart to Figure 2.1 and shows an IE data model for online retail. This is just one of several IE models that correspond to the UML model. That is because an IE model incorporates decisions about primary keys, of which there are multiple choices. In contrast, a UML model defers database details to later development stages.

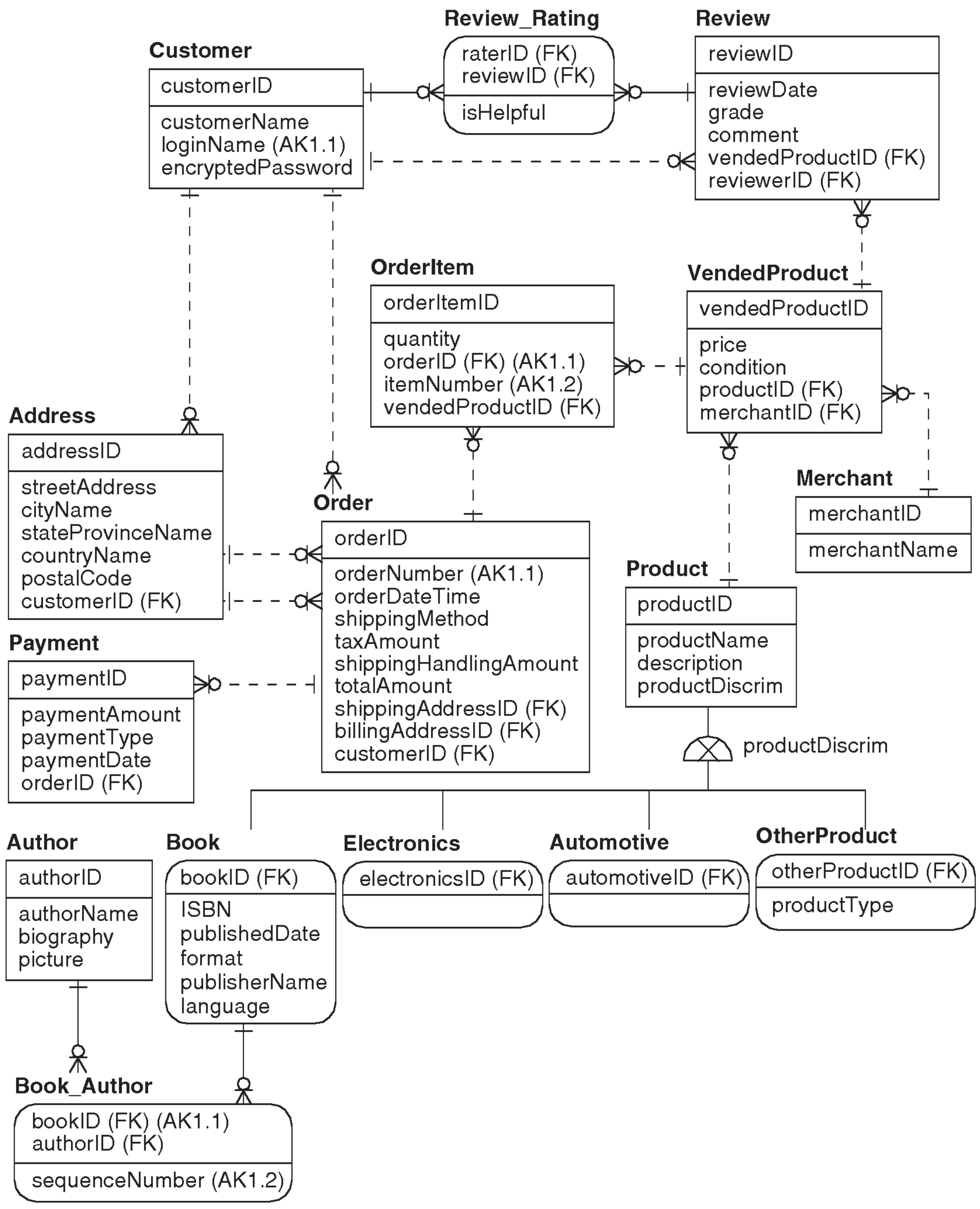

Figure 2.2 An example of an IE logical data model. This model provides the basis for most IE examples throughout the book.

In the IE model, we named an entity type *Review_Rating*. The name for this entity type was not needed in the UML model, but we had to create a name for it in the IE model.

2.3 Chapter Summary

A data model of customer orders for an online retail Web site provides an example of UML and IE constructs for subsequent chapters.

Bibliographic Notes

[Blaha–2005], [Fowler–2003], [Rumbaugh–2005], and [OMG] have further information about the UML notation. The UML class model is a derivative of Peter Chen's seminal ER notation and is essentially an ER dialect. The original IE books [Information Engineering] are dated now, but information about IE is available on the Web.

References

[Blaha-2005] Michael Blaha and James Rumbaugh. *Object-Oriented Modeling and Design with UML, Second Edition*. Upper Saddle River, NJ: Prentice Hall, 2005.

[Blaha-2011] A series of YouTube videos show how to construct UML data models using agile techniques. http://www.youtube.com/view_play_list?p=EE77921A75E846EB

[Chen-1976] P.P.S. Chen. The Entity-Relationship model—toward a unified view of data. *ACM Transactions on Database Systems 1*, 1 (March 1976), 9–36.

[Fowler-2003] Martin Fowler. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition*. Boston: Addison-Wesley, 2003.

[Information Engineering] Information Engineering (IE) is a database notation that was popularized by James Martin and Clive Finkelstein in the 1980s.

[OMG] The Object Management Group. www.omg.org

[Rumbaugh-2005] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Boston: Addison-Wesley, 2005.