Modeling Persistent Data

30

The objective of this chapter is to introduce different types of design models used to express the structure of data in persistent data stores.

30.1 Preconditions

The following should be true prior to starting this chapter.

- You understand five program design criteria: separation of concerns, design for reuse, design only what is needed, performance, and security.
- You understand the six characteristics of good software design: simplicity, coupling, cohesion, information hiding, performance, and security.
- You understand the notion of abstraction as a design process. Abstraction is the ability to generalize a concept by removing details that are not needed to properly convey the design perspective being emphasized.
- You have created and/or modified models that described an object-oriented or structured software design.
- You understand how to apply the Model–View–Controller architectural pattern to modify an existing design into these three components.
- You understand how to apply the Model–View–Controller architectural pattern to create a high-level design that accurately reflects the domain requirements while satisfying the design constraints inherent in MVC.

D. P. Voorhees, *Guide to Efficient Software Design*, Texts in Computer Science,
https://doi.org/10.1007/978-3-030-28501-2_30

30.2 Concepts and Context

Up to this point, the Address Book case study has used very simple data relationships and a memory-based data structure to non-persistently store the data. This chapter will discuss more complex data relationships and how these are represented in design models. Chapter 31 will then discuss two persistent storage technologies.

30.2.1 Externally Versus Internally Stored Data

Storing data internal to the software program requires the use of memory-based data structures. The ABA designs have been using data structures (e.g., lists, dictionaries, trees) to store nonpersistent data. In designing these solutions, our focus has been on the way in which the software needs to use the data and determine the type of data structure to use based on its performance characteristics.

Storing data in a persistent data store also requires an understanding of the way in which the software needs to access the stored data and the performance characteristics of the external file format. In addition, external file formats provide rules for how data elements are stored in relation to each other. Currently, our simple ABA case study has a name, along with a single phone number and email address, for each contact person. These simple data requirements present very few design options as we transition to a persistent data store (i.e., external file format). In Chaps. 32 and 33, we'll add more data requirements to the ABA, allowing us to explore more interesting data designs.

30.2.2 Logical Data Modeling

As we begin to focus on persistent data storage we need to think about data from two different perspectives: a logical view of the data and a physical view of the data. In a logical data model, we are focused on how the different data elements are related to each other. In a physical data model, we are concerned about how the data is physically stored within the external file format.

30.2.2.1 Entity Relationship Diagram

An entity relationship diagram (ERD) is a type of logical data model. An ERD is a high-level modeling technique that logically describes how different collections of data elements relate to each other. This modeling technique uses the notations shown in Fig. 30.1.

An *entity* represents a thing or object that consists of a collection of logically related data elements. The entity notation is a rectangle shape that has a noun or noun phrase to name the entity. A *relationship* describes how two entities are logically related to each other. The relationship notation is a diamond shape that has a verb or

verb phrase to describe the relationship. A *connector* is a line that connects an entity notation to a relationship notation.

30.2.2.2 Fully-Attributed Logical Data Model

A Fully-attributed logical data model (LDM) is another type of logical data model. A LDM is a more detailed modeling technique that logically shows how data elements are related to entities, and how entities are related to each other. This modeling technique uses the notations shown in Fig. 30.2.

The Entity, relationship, and connector notations are the same as the ERD. A LDM also includes notation to represent a *weak entity* and an *attribute*. A weak entity is similar to an entity, except a weak entity must include data from another entity in order to uniquely identify data instances of the weak entity. (The examples below will illustrate the differences between an entity and a weak entity.) The weak entity notation is a rectangle that has two lines that form its four sides. An *attribute* represents a single data element (i.e., data field) that is logically related to an entity. The attribute notation is an oval with a noun or noun phrase to name the attribute.

30.2.2.3 ERD and LDM Example

Let's say we want to model the data for a digital library that will store books and movies. We want to use this digital library to keep track of all of the titles we own. This digital library is intended for use by a single person.

The information we want to keep for a book might include the title, author, and publisher. The information we want to keep for a movie might include the title and director. To develop an ERD we need to think about these nouns (e.g., author, book, director, movie, publisher, and title) and determine which of these are entities and which would be attributes associated with an entity.

If we consider each of these nouns to be an entity we end up with an ERD similar to Fig. 30.3. Here the information for a book is represented in four entities. The book entity has a relationship with three other entities. Note the use of *cardinality* symbols (e.g., "1" and "*") for each relationship. The "written-by" relationship is read two ways: (1) *A book is written-by many authors*; and (2) *An author writes a book*. Below is a list of all of the data relationship rules for the ERD in Fig. 30.3.

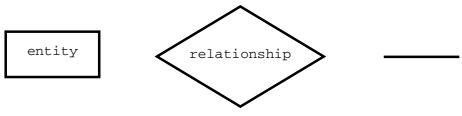


Fig. 30.1 Entity Relationship Diagram (ERD) notations

Fig. 30.2 Logical Data Model (LDM) notations

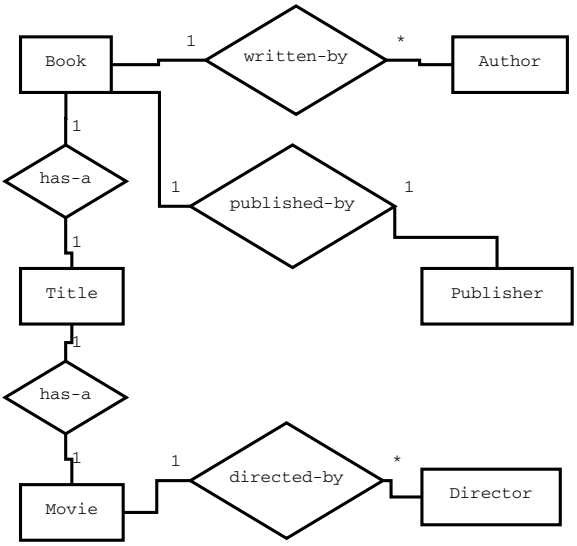


Fig. 30.3 Digital library ERD Version 1

- A book is written-by one or more authors. An author writes a book.
- A book is published by one publisher. A publisher publishes one book.
- A book has a title. A title is for one book.
- A movie is directed by one or more directors. A director directs a movie.
- A movie has a title. A title is for one movie.

There are (at least) two problems with this ERD. First, any additional information we want to store for a book or movie (e.g., year published) would result in another entity. This seems to contradict the purpose of an entity as a data object containing a collection of logically related data elements.

Second, what data attributes would we associate with each entity in the ERD? Figure 30.4 shows a LDM with an attribute associated with the author, title, publisher, and director entities. Note the book and movie entities do not have any attributes while the other three entities have only one attribute each. Again, this contradicts the purpose of an entity as a data object which contains a collection of logically related data elements.

Figure 30.5 shows a second attempt at an ERD for this digital library. This ERD does not show any relationship between the book and movie entities—we simply have a bunch of books and movies in the library where none of the books have any relationship to any of the movies. The other observation regarding the version 2 ERD is that all of the one-to-one relationships shown in the version 1 ERD have been eliminated, while the two one-to-many relationships are still shown in version 2. Below is a list of all of the data relationship rules for the ERD in Fig. 30.5.

- A book is written-by one or more authors. An author writes a book.
- A movie is directed by one or more directors. A director directs a movie.

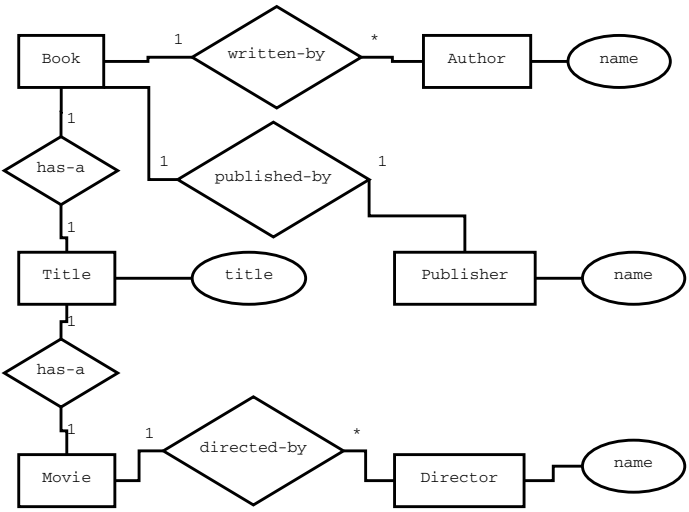
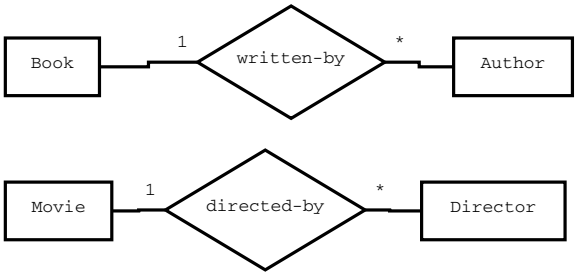


Fig. 30.4 Digital library LDM Version 1

Fig. 30.5 Digital library ERD Version 2



The LDM in Fig. 30.6 now makes more sense since each entity has a list of attributes associated with it. Each attribute represents a data value that helps to describe the entity. The isbn attribute is underlined for a book while the title and year published attributes are underlined for a movie. These attributes represent the *primary key* for the appropriate entity. That is, each book stored in the digital library will have a unique isbn value while each movie stored in the library will have a unique value for the combination of title and year published.

The version 2 LDM also shows the author and director as weak entities. This says an author of a book cannot be identified unless we know the specific book. From a data modeling perspective, the Author weak entity and its weak key attribute—name—requires the primary key value from the book instance (i.e., the isbn value) in order to correctly identify the authors for a book. A similar arrangement exists between movie and the director weak entity.

One last observation regarding Fig. 30.6—both the book and movie entities have attributes named title and year published. Can we simplify the LDM by showing one title and one year published attribute, with a relationship showing a connection from

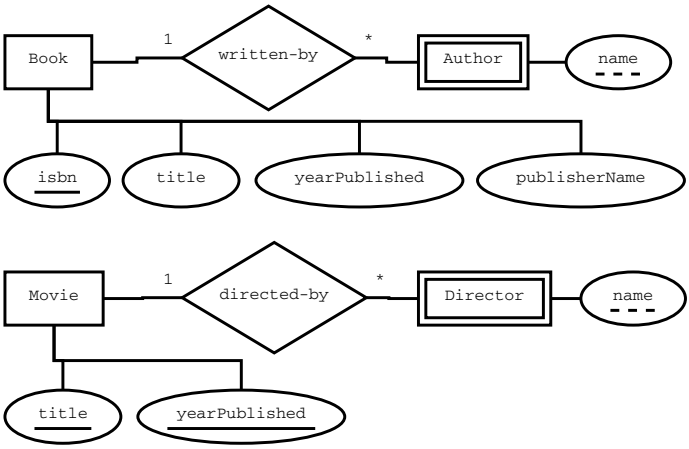
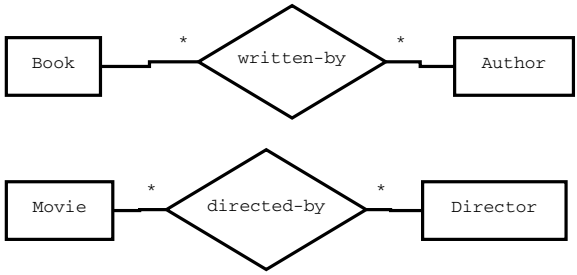


Fig. 30.6 Digital library LDM Version 2

Fig. 30.7 Digital library ERD Version 3



each of these attributes to both the book and movie entities? The answer is no, for two reasons. First, when an LDM shows one attribute being associated with two distinct entities, this is saying that the same attribute instance (i.e., value) is used in both entity instances. In the case of the digital library, it is not true that a title for a book will always exactly match the title for a movie. Second, the title and year published attributes associated with the movie entity are primary key attributes. However, these two attributes are not part of the primary key for the book entity.

Figure 30.7 shows a third attempt at an ERD for the digital library. The difference between the second and third ERDs are the cardinality rules. In version 3, we are showing a many-to-many relationship between book and author, and between movie and director. Below is a list of all of the data relationship rules for the ERD in Fig. 30.7.

- A book is written-by *one or more* authors. An author writes *one or more* books.
- A movie is directed by *one or more* directors. A director directs *one or more* movies.

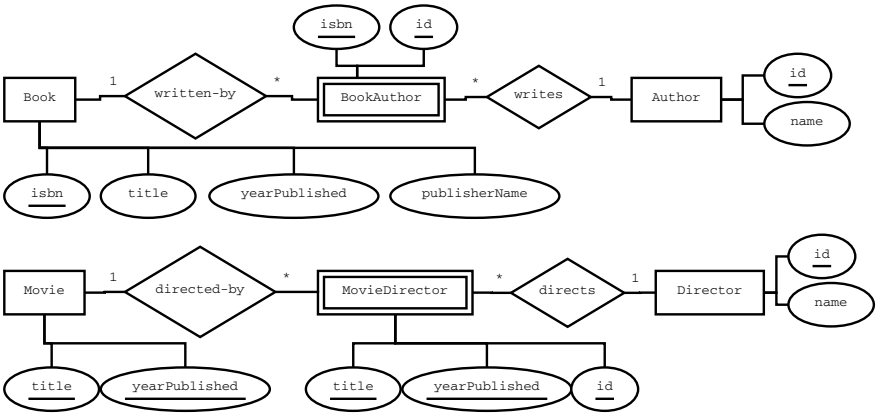


Fig. 30.8 Digital library LDM Version 3

The LDM in Fig. 30.8 has four relationships (written by, writes, directed by, and directs), four strong entities (book, author, movie, and director) and two weak entities (book author and movie director). Each weak entity and its two one-to-many relationships replace the corresponding many-to-many relationship shown in the ERD of Fig. 30.7. When converting a many-to-many relationship into two one-to-many relationships, the weak entity will contain, at a minimum, the primary key attributes from both strong entities they are related to. Below is a list of all of the data relationship rules for the LDM in Fig. 30.8. The first four rules are associated with cardinality and the last six rules describe the attributes associated with each entity.

- A book is written by *one or more* book authors. A book author writes *one* book.
- An author writes *one or more* book authors. A book author is written by *one* author.
- A movie is directed by *one or more* movie directors. A movie director directs *one* movie.
- A director directs *one or more* movie directors. A movie director is directed by *one* director.
- A book is uniquely identified by its isbn and is described by its title, year published, and publisher name.
- A book author is uniquely identified by its isbn and (author) id.
- An author is uniquely identified by its id (an integer value guaranteeing uniqueness) and is described by its name.
- A movie is uniquely identified by its title and year published.
- A movie director is uniquely identified by its title, year published, and (director) id.
- A director is uniquely identified by its id (an integer value guaranteeing uniqueness) and is described by its name.

Translating a many-to-many relationship in an ERD into two one-to-many relationships in an LDM is common practice when developing data models. This is done to make it easier to translate a logical data model into a physical data model based on the type of persistent storage technology to be used. This will be further discussed in Chap. 31.

30.2.3 Physical Data Modeling

Once we have a logical data model, the next step is to translate this into a physical representation. Unfortunately, there is not one approach to modeling the physical representation of data. This is because the type of physical storage to be used plays a significant role in how we decide to model the format of physical data. Two physical data formats—text-based XML files and relational databases—are described in detail in Chap. 31. A relatively simple physical data format is introduced in this chapter.

30.2.3.1 Text Files

When a text file is used to store data, we need to describe the order in which the data values are stored within the file. With this type of data storage, the most common approach is to store the data for a single entity instance on a line of text within the file. A common format for this type of text file is to use a delimiter character to separate each data value on a text line. Common character values to use as the delimiter are a comma or tab.

Listing 30.1 shows a text file where each logical data entity has its own *section* within the text file. The following observations are important:

- The special text lines {Book} and {Movie} are used to denote the start of text lines containing data for that logical data entity.
- Each data value is delimited with quotes, denoting a character (i.e., string) value. While an isbn value is a series of digits, we do not expect to do arithmetic on these values, and so we treat this data as characters.
- The data values on each text line are separated with commas.
- The book text lines list the author values last. This is because there may be many number of authors for a book.

Listing 30.1 Text File Example 1

```
{Book}
"isbn","title","edition","publisher","yearPublished","authors"
"0441172717","Dune","","Chilton Books","1965","Frank Herbert"
"0151008116","Life of Pi","","Harcourt","2001","Yann Martel"
"0137135599","Blown to Bits","Addison Wesley","2008","Hal
"0137135599","Abelson","Ken Ledeen","Harry Lewis"
{Movie}
"title","yearPublished","directors"
"The Shawshank Redemption","1994","Frank Darabont"
```

Listing 30.2 shows an alternative text file format where the name of a logical data entity is the first value found in each line of text.

Listing 30.2 Text File Example 2

```
"book","isbn","title","edition","publisher","yearPublished","authors"
"book","0441172717","Dune","","Chilton Books","1965","Frank Herbert"
"book","0151008116","Life of Pi","","Harcourt","2001","Yann Martel"
"book","0137135599","Blown to Bits","Addison Wesley","2008","Hal Abelson","Ken Ledeen","Harry Lewis"
"movie","title","yearPublished","directors"
"movie","The Shawshank Redemption","1994","Frank Darabont"
```

Both of these text file formats require the software code to be written with specific knowledge about the exact order of data values found on a text line. The code must know this order for each logical data entity whose data is stored in the text file. Another option would be to have a separate text file for each logical data entity. In this case, the text file would not need the special section text lines (that are shown in Listing 30.1) and each text line would be formatted with the same number of data values (like the three text lines shown inside the `Book` section).

30.2.4 Normalization

As a logical data model is created, one of our goals is to avoid unnecessary redundancy in the data elements. When redundant data elements exist in the physical format, any change to a data value would need to be updated in all of its redundant physical locations. Another goal when creating a logical data model is to keep the model as simple as possible while maintaining the logical relationships between the elements. To achieve these goals we apply a process called *normalization* [1]. This process analyzes data relations based on their functional dependencies and primary keys.

functional dependency: Denoted using the notation $A \rightarrow B, C$. This says that attributes B and C are functionally dependent on A. That is, a specific value for A will determine the values for B and C. For example, the functional dependency $studentID \rightarrow firstName, lastName, SSN$ says that once a specific student ID value is known, we can determine the first name, last name, and SSN for that student.

primary key: A primary key is one or more attributes that will uniquely identify each entity instance. Each attribute or set of attributes on the left side of a functional dependency are candidate primary keys.

relation: A text-based notation to express an entity and its associated attributes. For example, Student = (studentID, firstName, lastName, SSN) is a relation where *Student* is the name of the entity and the names inside the parentheses are the list of associated attributes. The underlined attribute(s) identify the primary key for the relation.

tuple: A text-based notation to express a list of attributes. For example, (studentID, firstName, lastName, SSN) is a tuple containing four attributes where one of the attributes (e.g., studentID) is the primary key.

The normalization process is described as a series of levels called forms.

30.2.4.1 First Normal Form (1NF)

A relation is in 1NF when *each attribute value is an atomic (or indivisible) value.* That is, a relation should have no multivalued attributes or nested relations [1].

For example, the following relation is not in 1NF.

```
Employee = (employeeID, employeeName, dependentName, address)
```

Why is this employee relation not in 1NF?

- The employee name is not atomic (e.g., first, middle, last).
- The dependent name is a multivalued attribute (i.e., an employee may have many dependents).
- The address is not atomic (e.g., street, city, state, zip).

30.2.4.2 Second Normal Form (2NF)

A relation is in 2NF when *it is in 1NF and each non-key attribute is fully functionally dependent on the primary key (PK).* That is, when a PK has multiple attributes, no non-PK attribute should be functionally dependent on a part of the PK [1].

For example, the following relation is not in 2NF.

```
EmployeeProject = (ssn, projID, hours, emplID, projName, projLocation)
```

First, this relation is in 1NF since each attribute is an atomic value. That is, each of the attributes listed in the relation are atomic and have a single value (for one employee project) instance. Why is this employee project relation not in 2NF?

- The functional dependency $ssn \rightarrow emplID$ shows a non-PK attribute (emplID) is functionally dependent on only a portion of the PK (ssn).
- The functional dependency $projID \rightarrow projName, projLocation$ shows two non-PK attributes (projName and projLocation) are functionally dependent on only a portion of the PK (projID).

One of the non-PK attributes—hours—is functionally dependent on the entire PK.

30.2.5 Third Normal Form (3NF)

A relation is in 3NF when *it is in 2NF and no non-PK attribute is transitively dependent on the PK*. That is, a relation should not have a non-key attribute functionally dependent on another non-key attribute [1].

For example, the following relation is not in 3NF.

```
EmployeeDepartment = (ssn, birthDate, deptID, deptName, deptMgrSSN)
```

First, this relation is in 2NF since the functional dependency below is true. $ssn \rightarrow birthDate, deptID, deptName, deptMgrSSN$ The reason why this is not in 3NF is that two of the attributes are transitively dependent on the PK as shown below:

- We can use the ssn to identify the attributes birth date and dept ID. Or, using functional dependency notation $ssn \rightarrow birthDate, deptID$. That is, knowing an employee's ssn allows us to get the employee's birth date and the department ID of where they work.
- We can use the dept ID to identify the attributes dept name and dept mgr ssn. Or, using functional dependency notation $deptID \rightarrow deptName, deptMgrSSN$. That is, knowing the department (deptID) allows us to get the name of the department and the department manager's ssn.
- Thus, each ssn (the PK) will identify a dept ID (a non-PK attribute), and each dept ID will identify two other non-PK attributes. This gives us a transitive dependency we want to avoid for 3NF.

30.2.6 Boyce-Codd Normal Form (BCNF)

A relation is in BCNF when *every non-trivial functional dependency in the relation is a dependency on a superkey*. That is, a relation is in BCNF if it eliminates all redundancy that can be discovered based on functional dependencies [1].

A *superkey* is any set of attributes that will uniquely identify each tuple in the relation. For example, the tuple (studentID, ssn, birthDate, firstName, lastName, graduationDate) has many valid functional dependencies, including:

- $studentID \rightarrow ssn, birthDate, firstName, lastName, graduationDate$
- $studentID, ssn \rightarrow birthDate, firstName, lastName, graduationDate$
- $studentID, ssn, birthDate \rightarrow firstName, lastName, graduationDate$
- $studentID, ssn, birthDate, firstName \rightarrow lastName, graduationDate$
- $studentID, ssn, birthDate, firstName, lastName \rightarrow graduationDate$
- $studentID, birthDate \rightarrow ssn, firstName, lastName, graduationDate$
- $studentID, birthDate, firstName \rightarrow ssn, lastName, graduationDate$
- etc.

All of these superkeys are candidate primary keys. The primary key is the minimal superkey (i.e., the minimal number of attributes in a relation that will uniquely identify each tuple in the relation). In this case, the first functional dependency in the above list shows the minimal superkey, resulting in the primary key being student ID.

For example, if we assume an employee can be on many projects, the following relation is not in BCNF.

```
EmployeeProject = (ssn, projID, projName, projHours)
```

The functional dependency $projID \rightarrow projName$ is true. But proj ID by itself is not a superkey (i.e., the combination of ssn and projID is the minimal superkey and the primary key).

30.2.7 Fourth Normal Form (4NF)

A relation is in 4NF when *every non-trivial multivalued dependency in the table is a dependency on a superkey*. To be in 4NF, a relation cannot have a multivalued dependency on a subset of the superkey [1].

For example, the following relation about a loan customer is not in 4NF.

```
LoanCustomer = (loanNumber, custID, custStreet, custCity)
```

First, this relation is in 3NF since no non-key attribute is functionally dependent on another non-key attribute. That is, all of the functional dependencies for this relation, listed below, show the non-key attributes are functionally dependent only on key attributes.

- $loanNumber, custID \rightarrow custStreet, custCity$
- $custID \rightarrow custStreet, custCity$

The second functional dependency above results in custStreet and custCity values being repeated for each customer loan, since only the custID is needed to uniquely identify these two attributes. This is why the above relation is not in 4NF.

30.2.7.1 Additional Levels of Normalization

Researchers have also defined a fifth (5NF) and a sixth normal form (6NF). These are not covered in this book as database designers tend to normalize up to either 3NF, BCNF, or 4NF. Furthermore, there are situations where a database designer will not normalize to 3NF to improve the performance of the database. This scenario is also outside the scope of this book.

30.2.8 Apply Normalization

We'll apply the normalization process to the digital library example presented earlier in this chapter.

Table 30.1 Relations and functional dependencies for LDM Example 2

```
Book = (isbn, title, yearPublished, publisherName)
Author = (isbn, authorName)
Movie = (title, yearPublished)
Director = (title, yearPublished, directorName)
```

- $isbn \rightarrow title, yearPublished, publisher$
- $isbn, authorName \rightarrow \varnothing$
- $title, yearPublished, directorName \rightarrow \varnothing$

30.2.8.1 Check 1NF

Is the ABA version 2 LDM in Fig. 30.6 in 1NF? i.e., does each attribute represent an atomic (or indivisible) value?

The relations and functional dependencies for the logical data model are shown in Table 30.1.

Each of the attributes now represents an atomic (or indivisible) value. (For this analysis, we are ignoring an author and director likely having a first and last name. We are considering their entire name to be atomic.) As shown in the functional dependencies, the multivalued author name and director name are part of the superkey for their respective relations. Given this, the logical data model in Fig. 30.6 is in 1NF.

30.2.8.2 Check 2NF

Is the ABA version 2 LDM in Fig. 30.6 in 2NF? i.e., are the relations in 1NF and is each non-key attribute fully functionally dependent on the primary key?

Given the relations and functional dependencies listed in Table 30.1:

- For a Book, knowing the isbn allows us to identify the title, yearPublished, publisher, and the authorNames for the book.
- For a Movie, knowing the title and year published allows us to identify the director names for the movie.

Thus, each of the non-key attributes is fully functionally dependent on the primary key. The logical data model in Fig. 30.6 is in 2NF.

30.2.8.3 Check 3NF

Is the ABA version 2 LDM in Fig. 30.6 in 3NF? i.e., are the relations in 2NF and is there no non-PK attribute that is transitively dependent on the PK?

Given the relations and functional dependencies listed in Table 30.1:

- Can a new edition of a book have the same isbn as a previous edition? No, each new edition has a different isbn value. So any change in the authors or publisher in this new edition will be identified given the new isbn value.

- Since the Movie and Director relations have no non-key attributes, these relations and their functional dependencies must be in 3NF.

Thus, each of the non-key attributes is fully functionally dependent on the primary key. The logical data model in Fig. 30.6 is in 3NF.

30.2.8.4 Check BCNF

Is the ABA version 2 LDM in Fig. 30.6 in BCNF? i.e., is every non-trivial functional dependency in a relation a dependency on a superkey?

Given the relations and functional dependencies listed in Table 30.1:

- Any functional dependency that specifies a superkey for the Book relation will have isbn as part of the superkey. Thus, all of the functional dependencies one could create in this relation are dependent on a superkey.
- Since the Movie and Director relations have no non-key attributes, these relations and their functional dependencies must be in BCNF.

Thus, each of the non-trivial functional dependencies is dependent on a superkey. The logical data model in Fig. 30.6 is in BCNF.

30.2.8.5 Check 4NF

Is the ABA version 2 LDM in Fig. 30.6 in 4NF? i.e., is every non-trivial multivalued dependency in a relation a dependency on a superkey?

Given the relations and functional dependencies listed in Table 30.1:

- The one multivalued dependency for the book relation has isbn and author as the superkey.
- All of the functional dependencies for the Movie relation have multivalued dependencies and the left side of each of these dependencies is a superkey.

Thus, each of the non-trivial multivalued dependencies is dependent on a superkey. The logical data model in Fig. 30.6 is in 4NF.

30.3 Post-conditions

The following should have been learned when completing this chapter.

- You understand how to develop data models, including entity relationship diagrams and fully-attributed logical data models. You also understand that translating an LDM to a physical data model needs to consider the limitations of the physical storage format.

- You understand the process of normalization and have applied normalization forms to a data design.

Exercises

Discussion Questions

1. In your own words, explain each of the following concepts.

   a. ERD
   b. LDM
   c. Physical data model
   d. Normalization
   e. 1NF, 2NF, 3NF, BCNF, 4NF

Hands-on Exercises

1. Apply the normalization forms to the Digital Library version 3 LDM. Which level of normalization does this data model reach?
2. The Digital Library version 3 LDM includes the attribute publisherName associated with the Book entity. It is logically correct to say a publisher will publish many books. Given this, create a new ERD and LDM to show a Publisher entity along with its corresponding attributes and relationships.
3. Use an existing design solution you've developed, develop data models in preparation for persistently storing data.
4. Using your data models from the first hands-on exercise, evaluate your data models using the normalization forms. Which level of normalization does your data models reach?

Reference

1. Silberschatz A, Korth HF, Sudarshan S (2006) Database System Concepts, 5th edn. McGraw-Hill, New York