Chapter 3. The Relational Model - Introduction to Database Systems

CHAPTER 3

THE RELATIONAL MODEL

After reading this chapter, the reader will understand:

- *The relational model and its three components, namely, structural component, manipulative component and integrity constraints.*
- *The relation schema and relation instance*
- *Attribute and its domain*
- *Unstructured and structured domain*
- *Degree and cardinality of a relation*
- *Characteristics of relations*
- *The relational database schema and relational database instance*
- *Keys including superkey, candidate key, primary key, alternate key and foreign key*
- *Various types of data integrity including domain integrity, entity integrity, referential integrity and semantic integrity*
- *Different types of integrity constraints*
- *Legal and illegal relation*
- *Constraint violation while insert, delete, and update operation*
- *Mapping of E-R model to relational model*
- *Representation of entity types to relation*
- *Representation of weak entity types, relationship types, composite and multivalued attributes, and Extended E-R features*

A database is a collection of interrelated data and the way data is related to each other depends upon the model being used. Generally databases are based on one of the three models: the *hierarchical* model, the *network* model or the *relational* model. A database based on the network or hierarchical model is a non-relational database and a database based

on the relational model is a relational database. A **relational database** is a collection of relations (or two-dimensional tables) having distinct names. It is a persistent storage mechanism that conforms to the relational model.

At the time when the relational model was proposed, hierarchical and network model were used to structure the data. Both these models are complex since they use pointers to relate the records. However, the relational model is simple as it uses the values of the records instead of pointers to relate the records. For example, in *Online Book* database, the common value from the publisher and book records such as publisher id is used to relate the publisher record to the book record, instead of using an internal pointer. In addition, it gives the freedom to change the way of storing and accessing the data without corresponding change in the way the user perceives the data.

The relational model is based on the branches of mathematics, namely, *set theory* and *predicate logic*. The key assumption is that the data is represented in the form of mathematical *n*-ary relations, which is a subset of the Cartesian product of *n* sets. Reasoning about such a data is done in two-valued predicate logic in mathematical model. In two-valued predicate logic, two possible evaluations for each proposition are possible, that is, either true or false. In addition to true or false, third value such as *null* is included in three-value predicate. Some consider only two-valued logic as crucial part of the relational model, whereas others include three-valued logic as crucial part of the relational model.

The relational model includes three components, namely, *structural component* (or data structures), *manipulative component* (or data manipulation), and *integrity constraints*. **Structural component** consists of two components, namely, entity types and their relationships, which act as a framework of the given situation. **Manipulative component** consists of a set of operations that are performed on a table or group of tables. **Integrity constraints** are a set of rules that governs the entity types and their relationships, and thereby, ensures the integrity of

database.

3.1 RELATIONAL MODEL CONCEPTS

The relational model represents both data and the relationships among those data using *relation*. A **relation** is used to represent information about any entity (such as book, publisher, author, etc.) and its relationship with other entities in the form of attributes (or columns) and tuples (or rows). A relation comprises of a *relation schema* and a *relation instance*. A relation schema (also termed as relation intension) depicts the attributes of the table, and a relation instance (also termed as relation extension) is a two-dimensional table with a time-varying set of tuples.

Relation Schema

A **relation schema** consists of a relation name R and a set of attributes (or fields) $A_1, A_2, \ldots, A_n$. It is represented by $R(A_1, A_2, \ldots, A_n)$ and is used to describe a relation R. To understand the concept of relation schema, consider the book, publisher, and author information stored in an *Online Book* database with the schema of the relations given as

```
BOOK(ISBN, Book_title, Category, Price, Copyright_date, Year,
Page_count, P_ID)

AUTHOR(A_ID, Aname, City, State, Zip, Phone, URL)

PUBLISHER(P_ID, Pname, Address, State, Phone, Email_id)
```

Learn More

Codd's model emerged as the dominant model that provides the basis for effectively using both the relational database software, such as Oracle, MS SQL Server, etc., and the personal database systems such as Access, FoxPro, etc. The main basis in developing relational model is that a database is a collection of unordered tables that can be modified through various operations that return a table or group of tables as their result.

In these statements, ISBN, Book_title, Category, Price, Copyright_date, Year, Page_count, and P_ID are the attributes of relation BOOK. A_ID, Aname, City, State, Zip, Phone, and URL are the attributes of relation AUTHOR. P_ID, Pname, Address, State, Phone, and Email_id are the attributes of relation PUBLISHER.

An attribute can be single-valued as well as multivalued attribute. Single-valued attributes represent single value of data, whereas multivalued attributes represent more than one value of data. Examples of single-valued attributes of relation PUBLISHER are P_ID, Pname, Address, and State, whereas Phone and Email_id are multivalued attributes of relation PUBLISHER.

An attribute consists of an attribute name and a value of one or more bytes from the set of permissible values of same data type. This set of permissible value of same data type for an attribute is termed as the **domain** D of that attribute. The relation schema of book information can be written using the data type of each attribute as

```
BOOK(ISBN: string, Book_title: string, Category: string,
Price: integer, Copyright_date: integer, Year: integer,
Page_count: integer, P_ID: string)
```

In this statement, the permissible value for the attribute ISBN consists of a set of character values. Thus, the domain for the attribute ISBN is string. Note that the data type or format for each domain is the **physical** definition of domains and the value of the domain is the **logical** definition of domains. Here, the data type string is the physical definition of domain and a set of character values is the logical definition of domain.

Several attributes may have same domain at the physical level but distinct domains at the logical level. For example, both the attributes Aname and Pname of relations AUTHOR and PUBLISHER, respectively, have same data type, that is, string. Hence, both the attributes have same domain at the physical level. However, both the attributes Aname and

`Pname` have different values, say, *Thomas William* and *Hills Publications*, respectively. Hence, both the attributes have distinct domains at the logical level.

Several attributes may have either same or distinct domains at both physical and logical level. For example, the attributes `State` of relations `AUTHOR` and `PUBLISHER` have same domain both at the physical level, that is `string`, and at the logical level, say, *New York*. On the other hand, the attributes `URL` and `Phone` of relations `AUTHOR` and `PUBLISHER`, respectively, ought to have distinct domains both at the physical level that is, `string` for `URL` and `integer` for `Phone`, and also at the logical level.

Note that domain can be unstructured or structured. **Unstructured (or atomic) domain** consists of the non-decomposable or indivisible values that cannot be further divided into sub-values. Atomic domains are also known as *application-independent domains* since it consists of the general sets that are independent of a particular application. Sets of strings, integers, real numbers, etc. are some of the examples of unstructured domain. On the other hand, **structured (or composite) domain** consists of non-atomic values that are decomposable or divisible values. The domain for the attribute `Address` of relation `PUBLISHER` consisting of house number, street name, and city is an example of structured domain. Structured domains are also known as *application-dependent domains* since these are the subset of standard data types and can be defined by specifying the constraints on the values that can be inserted into the application.

Relation Instance

A **relation instance** (or relation) `r` of the relation schema $R(A_1, A_2, \ldots, A_n)$ is a set of *n*-tuples `t`. It is also denoted by `r(R)`. A relation instance is an ordered set of attributes values `v` that belongs to the domain `D` and it can be denoted as $r = \{t_1, t_2, \ldots, t_m\}$ where, $t = \{v_1, \ldots, v_n\}$. In other words, it is a set of tuples (or records) having the same number of attributes as the relation schema. It can be specified as $t \in r$, which

denotes that tuple `t` is in relation `r`. A relation instance can be considered as a *table*, which is a collection of tuples having the same number of attributes.

Note that a relation with *n* attributes is a subset of a Cartesian product of *n* domains. In other words, a relation of *n* attributes is a subset of $D_1 \times D_2 \times D_3 \times \ldots \times D_n$. Consider the BOOK relation in which domains D_1, D_2, D_3, …, D_8 denote the set of all ISBN, Book_title, Price,...,P_ID, respectively. Now any tuple of BOOK relation must comprise of eight attributes $(A_1, A_2, \ldots, A_8)$, where the values of A_1(ISBN) belong to domain D_1 and similarly, the values of remaining attributes $A_2, A_3, \ldots, A_8$ belong to domains $D_2, D_3, \ldots, D_8$, respectively. Since a BOOK relation is a subset of all possible attributes values in a list of domains, it can be written as a subset of $D_1 \times D_2 \times D_3 \times \ldots \times D_8$.

Consider the BOOK relation of Figure 3.1, the instance B1 contains nine tuples and eight attributes. If the tuple `t` denotes the first tuple of the relation, the notation `t [ISBN]` refers to the value of `t` on the ISBN attribute. Hence, `t [ISBN]` = "001-354-921-1", `t [Book_title]` = "Ransack" and so on. Mathematically, it can also be written as `t [1]` = "001-354-921-1", `t [2]` = "Ransack" and so on. Here, `1` refers to the first attribute of a BOOK relation, `2` refers to the second attribute, and so on.

Attributes (or Fields or Columns)

Tuples (Records or rows)

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

Fig. 3.1 *Instance* `B1` *of the* `BOOK` *relation*

The number of attributes in a relation is known as **degree** or **arity**, and the number of tuples in a relation is known as **cardinality**. A relation of degree one is known as **unary relation**, of degree two is known as **binary relation**, of degree three is known as **ternary relation**, and of degree *n* is known as ***n*-ary relation**. For example, the degree of the relation in Figure 3.1 is eight and its cardinality is nine.

3.1.1 Characteristics of Relations

A relation has certain characteristics, which are given here.

Learn More

A relation of zero degree (or with no attributes) is known as nullary

relation and it is of the type `RELATION{}`. There are two relations of zero degree, namely, TABLE_DEE (or DEE) and TABLE_DUM (or DUM). DEE contains only one tuple, namely, 0-tuple and is of the type `RELATION{TUPLE {}}`. On the other hand, DUM is empty and contains no tuples. DEE and DUM are mainly used in the relational algebra and they represent TRUE (or YES) and FALSE (or NO), respectively.

- **Ordering of Tuples in a Relation:** Since a relation is a set of tuples and a set has no particular order among its elements, tuples in a relation do not have any specified order. However, tuples in a relation can be logically ordered by the values of various attributes. In that case, information in a relation remains the same, only the order of tuple varies. Hence, tuple ordering in a relation is irrelevant. For example, if the `BOOK` relation in Figure 3.1 is logically ordered by the values of attribute `Category`, the first tuple of `BOOK` relation in Figure 3.1 becomes the third tuple of `BOOK` relation in Figure 3.2. Similarly, second tuple becomes the fifth tuple in Figure 3.2. Thus, every tuple of `BOOK` relation in Figure 3.1 is the tuple of `BOOK` relation in Figure 3.2 and vice versa. Since both the relations contain the same set of tuples, they are the same.

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

Fig. 3.2 *The* `BOOK` *relation (sort by* `Category`*)*

- **Ordering of values within a Tuple:** As discussed earlier, *n*-tuple is an ordered set of attributes values that belongs to the domain `D`, so, the order in which the values appear in the tuples is significant. However, if a tuple is defined as a set of (<`attribute`>: <*value*>) pairs, the order in which attributes appear in a tuple is irrelevant. This is due to the reason that there is no preference for one attribute value over another. For example, consider these statements.

  `t =<ISBN:` *001-987-760-9*, `Book_title:` *C++*, `Category:` *Textbook*, `Price:` *40*, `Copyright_date:` *2003*, `Year:` *2005*, `Page_count:` *800*, `P_ID:` *P001* >

  `t =<Category:` *Textbook*, `ISBN:` *001-987-760-9*, `Price:` *40*, `P_ID:` *P001*, `Book_title:` *C++*, `Page_count:` *800*, `Copyright_date:` *2003*, `Year:` *2005* >

In these statements, since the attributes and values are represented as pairs, these two tuples are the same.

- **Values and Nulls in the Tuples:** Relational model is based on the assumption that the tuples in a relation contain only atomic values. In other words, each tuple in a relation contains a single value for each of its attribute. Hence, a relation does not allow composite and multivalued attributes. Moreover, it allows denoting the value of the attribute as *null*, if the value does not exist for that attribute or the value is unknown. For example, if a publisher does not have an email address, the *null* value can be specified, which signifies that the value does not exist (see Figure 3.3).

P_ID	Pname	Address	State	Phone	Email_id
P001	Hills Publications	12, Park street, Atlanta	Georgia	7134019	h_pub@ hills.com
P002	Sunshine Publishers Ltd.	45, Second street, Newark	New Jersey	6548909	null
P003	Bright Publications	123, Main street, Honolulu	Hawaii	7678985	bright@ bp.com
P004	Paramount Publishing House	789, Oak street, New York	New York	9254834	param_house @param.com
P005	Wesley Publications	456, First street, Las Vegas	Nevada	5683452	null

Fig. 3.3 *The* PUBLISHER *relation*

NOTE A relation that assigns a single value to each attribute is said to be in first normal form, which will be discussed in Chapter 06.

- **No two tuples are identical in a relation:** Since a relation is a set of tuples and a set does not have identical elements. Therefore, each tuple in a relation must be uniquely identified by its contents. In other words, two tuples with the same value for all the attributes (that is, duplicate tuples) cannot exist in a relation.
- **Interpretation of a Relation:** A relation can be used to interpret facts about entities as a type of assertion. For example, the PUBLISHER relation in Figure 3.3 asserts that an entity PUBLISHER has

P_ID, Pname, Address, State, Phone, and Email_id. Here, the first tuple asserts the fact that there is a publisher whose publisher id is *P001*, name is *Hills Publications*, and so on.

A relation can also be used to interpret facts about relationships. For example, consider the BOOK relation in Figure 3.1, the attribute P_ID relates the BOOK relation with the PUBLISHER relation.

Learn More

E.F. Codd designed a set of 13 rules (numbered 0 to 12), known as Codd's 12 rules, to define relational database system. These rules are so strict that almost all the popular RDBMSs fail to meet the criteria.

3.2 RELATIONAL DATABASE SCHEMA

A set of relation schemas $\{R_1, R_2, \ldots, R_m\}$ together with a set of integrity constraints in the database constitutes **relational database schema**. A relational database schema, say S, is represented as $S = \{R_1, R_2, \ldots, R_m)$.

For example, the relational database schema *Online Book* is a set of relation schemas, namely, BOOK, PUBLISHER, AUTHOR, AUTHOR_BOOK, and REVIEW, which is shown in Figure 3.4.

BOOK

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID

PUBLISHER

P_ID	Pname	Address	State	Phone	Email_id

AUTHOR

A_ID	Aname	City	State	Zip	Phone	URL

AUTHOR_BOOK

A_ID	ISBN

REVIEW

R_ID	ISBN	Rating

Fig. 3.4 *Relational database schema*

Hence, the relational database schema *Online Book* can be represented as

```
Online Book ={BOOK, PUBLISHER, AUTHOR, AUTHOR_BOOK, REVIEW}
```

NOTE To define a relational database schema, a non-procedural language, SQL, is required to communicate with relational database. SQL will be discussed in Chapter 05.

3.3 RELATIONAL DATABASE INSTANCE

A **relational database instance** s of relational database schema $S = \{R_1, R_2, \ldots, R_m\}$ is a set of relation instances $\{r_1, r_2, \ldots, r_m\}$ such that each relation instance r_i is a state of corresponding relation schema R_i. In addition, each relation instance must satisfy the integrity constraints specified in the relational database schema. For example, a relational database instance corresponding to the *Online Book* schema is shown in Figure 3.5.

Note that when relational model was developed, some of its versions were developed on the assumption that those attributes which represent same real-world concept would be given same attribute names in all relations. This assumption had a limitation that it would not be possible to represent attributes having same real-world concept with different meanings in the same relation. Nowadays, different names can be given to the attributes representing the same real-world concept with different meanings in the same relation. To elaborate this concept, consider the modified form of *Online Book* database in which the relations, namely, AUTHOR_BOOK and REVIEW are merged together in a single relation say, AUTHOR_REVIEWER. The resultant relation is shown in Figure 3.6.

In the AUTHOR_REVIEWER relation, the concept of author id appears twice, once as an author and another as a reviewer who is also an author. The different attribute names, that is, A_ID and R_ID are given to them to represent different meaning in the same relation.

Attributes representing the same real-world concept may have same or different names in different relations. For example, the attribute `P_ID` in both `PUBLISHER` and `BOOK` relations represents the same concept with same attribute name in different relations. However, the attribute `A_ID` in the `AUTHOR` relation, and the attribute `R_ID` in the `REVIEW` relation represent the same concept, but with different attribute name in different relations. On the other hand, attributes representing different real-world concept may have same name in different relations. For example, the attribute `Pname` in the `PUBLISHER` relation and the attribute `Aname` in the `AUTHOR` relation represents different real-world concept, that is, publisher name and author name. However, they can be given a same attribute name, say, `Name` in different relations, that is, `PUBLISHER` and `AUTHOR` relations.

BOOK

ISBN	Book_title	Category	Price	Copyright _date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

PUBLISHER

P_ID	Pname	Address	State	Phone	Email_id
P001	Hills Publications	12, Park street, Atlanta	Georgia	7134019	h_pub@ hills.com
P002	Sunshine Publishers Ltd.	45, Second street, Newark	New Jersey	6548909	null
P003	Bright Publications	123, Main street, Honolulu	Hawaii	7678985	bright@ bp.com
P004	Paramount Publishing House	789, Oak street, New York	New York	9254834	param_house @param.com
P005	Wesley Publications	456, First street, Las Vegas	Nevada	5683452	null

AUTHOR

A_ID	Aname	State	City	Zip	Phone	URL
A001	Thomas William	New York	New York	14998	923673	www.thomas.com
A002	James Erin	Georgia	Atlanta	31896	376045	www.ejames.com
A003	Charles Smith	California	Los Angeles	95031	419562	www.csmith.com
A004	Lewis Ian	Washington	Seattle	98012	578932	www.au_lewis.com
A005	Allen Ford	Alaska	Juneau	99502	581231	www.allenford.com
A006	Jones Martin	New York	Albany	14521	218161	www.martin.com
A007	John Stephen	Texas	Austin	75112	316292	www.john_stepen.com
A008	Annie George	Michigan	Detroit	48011	321313	www.annie.com
A009	Suzanne Hedley	Washington	Seattle	98001	785236	www.suzz.com
A010	Richard Flaming	Virginia	Virginia Beach	22111	456163	www.richard.com

AUTHOR_BOOK

A_ID	ISBN
A001	001-987-760-9
A002	001-678-980-4
A003	001-987-760-9
A004	003-456-433-6
A005	001-354-921-1
A006	002-678-880-2
A007	001-987-650-5
A008	002-678-980-4
A008	004-765-409-5
A009	004-765-359-3
A010	003-456-533-8

REVIEW

R_ID	ISBN	Rating
A001	002-678-980-4	2
A002	001-987-760-9	6
A003	002-678-980-4	5
A003	004-765-409-5	4
A004	003-456-533-8	9
A005	002-678-980-4	7
A006	001-354-921-1	7
A006	002-678-880-2	4
A007	004-765-359-3	3
A008	001-987-760-9	7
A009	001-987-650-5	8
A010	003-456-433-6	5

Fig. 3.5 *Relational database instance*

A_ID	ISBN	R_ID	Rating
A001	001-987-760-9	A002	6
A002	002-678-980-4	A001	2
A003	001-987-760-9	A002	6
A004	003-456-433-6	A010	5
A005	001-354-921-1	A006	7
A006	002-678-880-2	A005	4
A007	001-987-650-5	A009	8
A008	002-678-980-4	A001	2
A008	004-765-409-5	A003	4
A009	004-765-359-3	A007	3
A010	003-456-533-8	A004	9

Fig. 3.6 *The* `AUTHOR_REVIEWER` *relation*

3.4 KEYS

Key is one of the important concepts of relational database. It can be

superkey, candidate key, primary key, and *foreign key*.

Superkey

A superkey (`SK`) is a subset of attributes of a relation schema `R`, such that for any two distinct tuples t_1 and t_2 in the relation state `r` of `R`, we have $t_1[SK] \neq t_2[SK]$. It means that the combination of attributes in `SK` uniquely identify each tuple in `r`. In every relation schema `R`, we have at least one default superkey, that is, the set of all the attributes, since no two tuples can be identical in all of its attributes.

For example, consider a relation schema `BOOK` with three attributes `ISBN`, `Book_title`, and `Category`. As we know, the value of `ISBN` is unique for each tuple; hence, `{ISBN}` is a superkey. In addition, the combination of all the attributes, that is, `{ISBN, Book_title, Category}` is a default superkey for this relation schema.

Candidate Key

Generally, all the attributes of a superkey are not required to identify each tuple uniquely in a relation. Instead, only a subset of attributes of the superkey is sufficient to uniquely identify each tuple. Further, if any attribute is removed from this subset, the remaining set of attributes can no longer serve as a superkey. Such a minimal set of attributes, say `K`, is a **candidate key** (also known as **irreducible superkey**).

For example, the superkey `{ISBN, Book_title, Category}` is not a candidate key, since its subset `{ISBN}` is a minimal set of attributes that uniquely identify each tuple of `BOOK` relation. So, `ISBN` is a candidate key.

Note that each relation can have more than one candidate key, and all candidate keys are super-keys whereas all superkeys are not candidate keys. Further, if a superkey includes a single attribute, it is a candidate key. However, if a candidate key has multiple attributes, all these attributes must be needed to uniquely identify each tuple.

Primary Key

Since a relation can have more than one candidate key, one of these keys has to be chosen as the primary key to uniquely identify each tuple. A candidate key that is chosen to uniquely identify a tuple in a relation is known as **primary key** (PK). Other candidate keys that are not chosen as primary key are known as **alternate keys**. For example, in `BOOK` relation, there are two candidate keys, namely, `ISBN` and `Book_title` (considering no two books can have the same title). If the attribute `ISBN` is chosen as a primary key, the attribute `Book_title` becomes the alternate key.

Generally, the primary key comprises one attribute in a relation; however, more than one attribute can form the primary key. In that case, primary key is known as composite key.

The attribute whose value is never or rarely changed should be chosen as primary key. For example, in `PUBLISHER` relation, the attribute `Address` should not be chosen as a primary key since it can be changed often. However, the attribute `P_ID` can be chosen as a primary key since each publisher has a unique publisher id and generally, it does not change.

Foreign Key

Attribute of one relation can be accessed in another relation by enforcing a link between the attributes of two relations. This can be done by defining the attribute of one relation as the foreign key that refers to the primary key of another relation. For example, consider two relations, namely, `PUBLISHER` and `BOOK`. Here, all the books must be associated with a publisher that is already in the `PUBLISHER` relation. In this case, a foreign key will be defined on the `BOOK` relation, which will be related to the primary key of the `PUBLISHER` relation (see Figure 3.7). Thus, all books in the `BOOK` relation would be related to a publisher in the `PUBLISHER` relation.

Things to Remember

The candidate key with least number of attributes should be chosen as

the primary key of the relation.

A relation that references another relation is known as **referencing relation**, whereas a relation that is being referenced is known as **referenced relation**. For example, in Figure 3.7, the PUBLISHER relation is a referenced relation and the BOOK relation is a referencing relation.

A foreign key may refer to its own relation. Such a foreign key is known as **self-referencing** or **recursive** foreign key.

A relation can have zero or more foreign keys and each foreign key can refer to different referenced relations. For example, consider REVIEW relation that includes the details of ratings given by the reviewer to a particular book. Here, books and reviewers (or authors) must be associated with book and author that are already in the BOOK and AUTHOR relations, respectively. In this case, foreign keys will be defined on the REVIEW relation that will be related to the primary keys of the BOOK and AUTHOR relations (see Figure 3.8).

BOOK

Foreign key

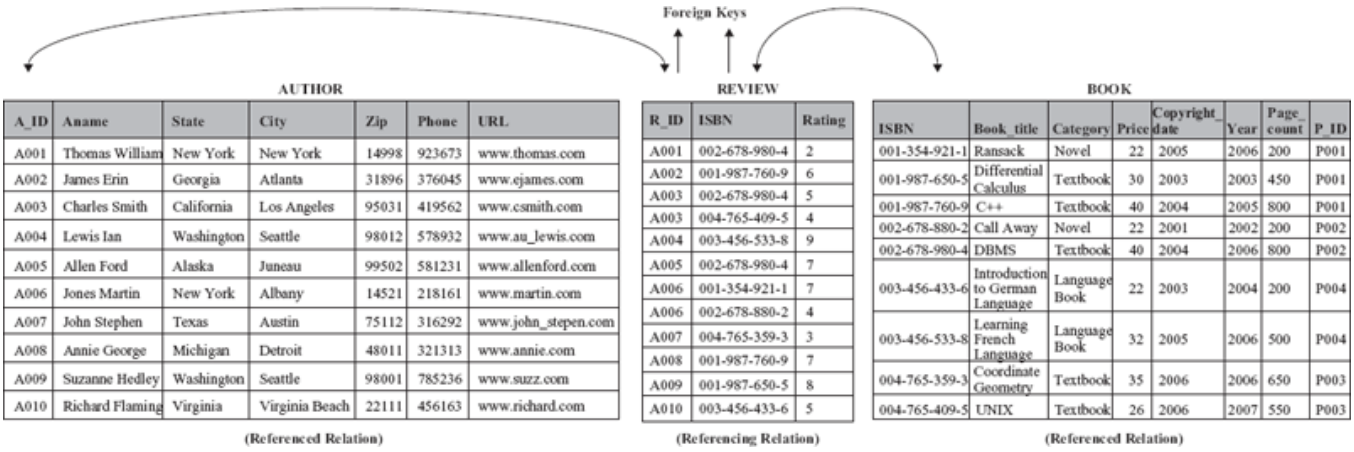
ISBN	Book_title	Category	Price	Copyright date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

Primary key

PUBLISHER

P_ID	Pname	Address	State	Phone	Email_id
P001	Hills Publications	12, Park street, Atlanta	Georgia	7134019	h_pub@hills.com
P002	Sunshine Publishers Ltd.	45, Second street, Newark	New Jersey	6548909	null
P003	Bright Publications	123, Main street, Honolulu	Hawaii	7678985	bright@bp.com
P004	Paramount Publishing House	789, Oak street, New York	New York	9254834	param_house@param.com
P005	Wesley Publications	456, First street, Las Vegas	Nevada	5683452	null

Fig. 3.7 *Foreign key*



A_ID	Aname	State	City	Zip	Phone	URL
A001	Thomas William	New York	New York	14998	923673	www.thomas.com
A002	James Erin	Georgia	Atlanta	31896	376045	www.ejames.com
A003	Charles Smith	California	Los Angeles	95031	419562	www.csmith.com
A004	Lewis Ian	Washington	Seattle	98012	578932	www.au_lewis.com
A005	Allen Ford	Alaska	Juneau	99502	581231	www.allenford.com
A006	Jones Martin	New York	Albany	14521	218161	www.martin.com
A007	John Stephen	Texas	Austin	75112	316292	www.john_stepen.com
A008	Annie George	Michigan	Detroit	48011	321313	www.annie.com
A009	Suzanne Hedley	Washington	Seattle	98001	785236	www.suzz.com
A010	Richard Flaming	Virginia	Virginia Beach	22111	456163	www.richard.com

R_ID	ISBN	Rating
A001	002-678-980-4	2
A002	001-987-760-9	6
A003	002-678-980-4	5
A003	004-765-409-5	4
A004	003-456-533-8	9
A005	002-678-980-4	7
A006	001-354-921-1	7
A006	002-678-880-2	4
A007	004-765-359-3	3
A008	001-987-760-9	7
A009	001-987-650-5	8
A010	003-456-433-6	5

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

Fig. 3.8 More than one foreign key in one referencing relation

A foreign key attribute in referencing relation may have a different name from that of the primary key attribute of referenced relation. However, the domain of both the attributes must be same. For example, the foreign key attribute `R_ID` of the `REVIEW` relation has a different name from the primary key attribute `A_ID` of the referenced relation `AUTHOR`. However, the data type of both the attributes is same, that is, `string`.

A foreign key in a referencing relation that matches a primary key in a referenced relation represents many-to-one relationship between referencing and referenced relation.

3.5 DATA INTEGRITY

The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity ensures that the data in the database is consistent, accurate, correct, and valid. It ascertains that the data adhere to the set of rules defined by the database administrator and hence, prevents the entry of the invalid information into database.

Data integrity is of four types, namely, *domain integrity, entity integrity, referential integrity*, and *semantic integrity*. Generally, domain integrity is applied on the attribute; entity integrity is applied on the tuple; referential integrity is applied on the relation; and semantic integrity ensures logical data in the database. This section discusses the data integrity in detail.

Domain Integrity

Domain integrity can be specified for each attribute by defining its specific range or domain. It requires that the attribute value must fall under a particular range in order to be valid. It ensures correct values for an attribute by defining the restrictions on the data type, format or range of possible values. For example, the domain of the attribute `Price` of the `BOOK` relation for the `Category`: *Textbook* may be between *$20* and *$200*. Similarly, the domain of the attribute `Category` may be *Textbook, Novel* or *Language Book*.

More formally, domain integrity ensures

$$A \text{ in } D$$

where, `A` denotes an attribute, which has the same set of values as that of the domain `D`. Note that a tuple with *n* attributes satisfying domain integrity can be written as

$$\{< A_1:V_1, A_2:V_2, \ldots, A_n:V_n > | V_1 \in D_1, V_2 \in D_2, \ldots, V_n \in D_n\}$$

where,

> $A_1, A_2, \ldots, A_n$ are attributes of a tuple
>
> $V_1, V_2, \ldots, V_n$ are the value set associated with the domain $D_1, D_2, \ldots, D_n$, respectively,
>
> These statements imply that the attribute value must belong to its associated domain values. For example, the first tuple of `PUBLISHER` relation can be written as

<P_ID: *P001*, Pname: *Hills Publications*, Address: *12, Park Street, Atlanta*, State: *Georgia*, Phone: *7134019*, Email_id: *h_pub@hills.com* >

Note that the domain integrity tests the value entered by the user into the database by ensuring that the attribute value entered by the user is in its

associated domain. The attribute values that do not violate any domain integrity rules are considered as valid values even if they are logically incorrect. For example, if the `Price` attribute of the `BOOK` relation is assigned a value *65* instead of *56*, this value is considered as valid since it belongs to the set of values defined in the domain, that is, between *$20* and *$200*.

Instead of defining a set of range for the attribute, domain integrity can be defined conditionally on the attribute values that can be validated according to the respective rules applied on them. If the attribute values entered by the user violate the domain integrity rules, the entered values are rejected. For example, the restriction on the domain of the attribute `Price` of `BOOK` relation can be applied as

> If `Category` is *Textbook,* `Price` must be between *$20* and *$200*
>
> If `Category` is *Language Book,* `Price` must be between *$20* and *$150*
>
> If `Category` is *Novel,* `Price` must be between *$20* and *$100*

Entity Integrity

Entity integrity assigns restriction on the tuples of a relation and ascertains the accuracy and consistency of the database. It ensures that each tuple in a relation is uniquely identified in order to retrieve each tuple separately. The primary key of a relation is used to identify each tuple of a relation uniquely.

Entity integrity is a rule, which ensures that the set of attribute(s) that participates in the primary key cannot have duplicate value or *null* value. This is because each tuple in a relation is uniquely identified by a primary key attribute and if the value of the primary key attribute is duplicate or null, it becomes difficult to identify such tuple uniquely. For example, if the attribute `ISBN` is declared as a primary key of `BOOK` relation, entity integrity ensures that any two tuples of `BOOK` relation must have unique value for the attribute `ISBN` and must not be *null*.

NOTE If an attribute is declared as a primary key, it adheres to entity integrity rule.

Entity integrity rule requires that various operations like `insert`, `delete`, and `update` on a relation maintains uniqueness and existence of a primary key. In other words, `insert`, `delete`, and `update` operations on a relation results in valid values, that is, unique and non-*null* values in primary key attribute. Any operation that provides either duplicate or *null* value in primary key attribute is rejected.

Note that it is not mandatory for each relation to have a primary key but it is a good practice to create a primary key for each relation

Referential Integrity

Referential integrity condition ensures that the domain for a given set of attributes, say `S1`, in a relation `r1` should be the values that appear in certain set of attributes, say `S2`, in another relation `r2`. Here, the set `S1` is foreign key for the referencing relation `r1` and `S2` is primary key for referenced relation `r2`. In other words, the value of foreign key in the referencing relation must exist in the primary key attribute of the referenced relation, or that the value of foreign key is *null*.

Referential integrity also ensures that the data type of the foreign key in referencing relation must match the data type of the primary key of referenced relation. For example, the data type of both foreign key attribute `P_ID` in `BOOK` relation and primary key attribute `P_ID` in `PUBLISHER` relation is same, that is, `string`.

Note that there may be some tuples in the referenced relation that do not exist in the referencing relation. For example, in Figure 3.9, the tuple with the attribute value *P005* in `PUBLISHER` relation does not exist in the `BOOK` relation.

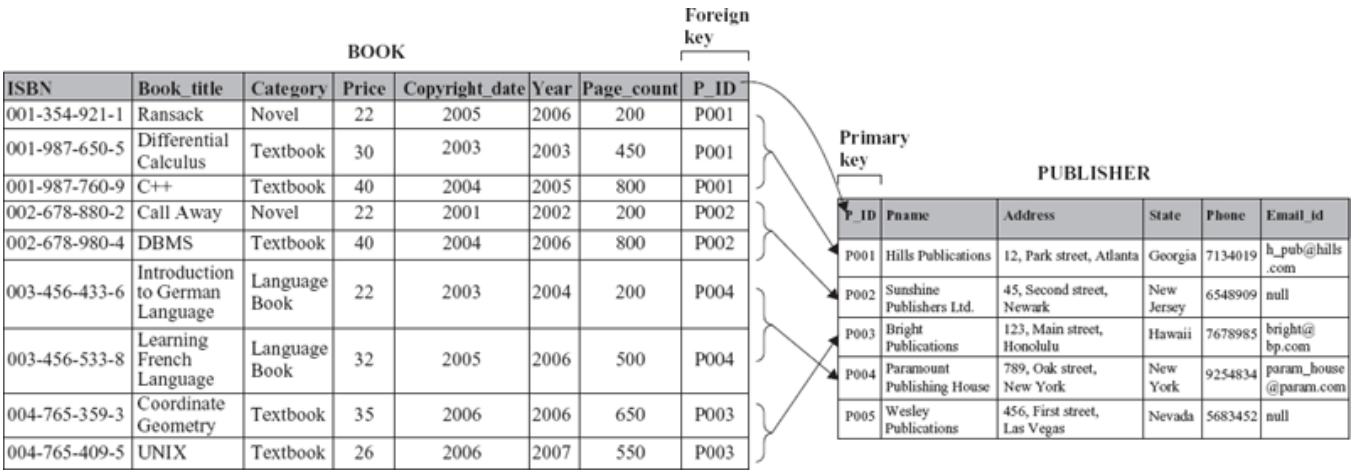


Fig. 3.9 *Referential integrity*

Semantic Integrity

To represent real world accurately and consistently, the business rules and logical rules (that are derived from our knowledge of the semantics of the real world) must be enforced on database. These rules are known as **semantic integrity**. Semantic integrity ensures that the data in the database is logically consistent and complete with respect to the real world. Examples of semantic integrity are:

- Number of pages of a book cannot be zero.
- One book can be published by one publisher only.
- Copyright date of a book cannot be later than its published date.
- An author cannot review his own book.

3.5.1 Enforcing Data Integrity

In order to maintain the accuracy and consistency of data in a database, integrity of the data is enforced through *integrity constraints*. **Integrity constraints** are the rules defined on a relational database schema and satisfied by all the instances of database in order to model the real-world concepts correctly. They ensure the consistency of database while the modifications are made by the users. For example, the integrity constraint ensures that the value of `ISBN` in `BOOK` relation cannot be duplicated or *null*.

A relation that conforms to all the integrity constraints of the schema is

known as **legal relation**. On the other hand, a relation that does not conform to all the integrity constraints of the schema is known as **illegal relation**.

Integrity constraints can be categorized into three parts, namely, *table-related constraints, domain constraints*, and *assertion constraints*.

- **Table-related constraints** are defined within a relation definition. This type of constraint can be specified either as a part of the attribute definition or as an element in the relation definition. When a constraint is specified as a part of the attribute definition, it is known as **column-level** constraint. Column-level constraints are checked when the constrained attribute is modified. On the other hand, when a constraint is specified as an element in the relation definition, it is known as **table-level** constraint. Table-level constraints are checked when any attribute of a relation is modified. Both column-level and table-level constraints hold different types of constraints (see Figure 3.10).

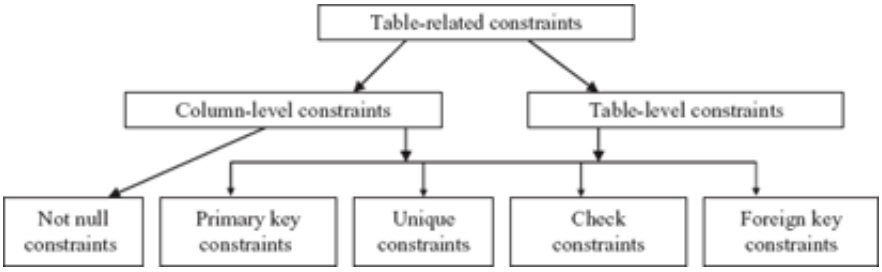


Fig. 3.10 *Table-related constraints*

- **Domain constraints** are specified within a domain definition. In other words, it is associated with any attribute that is defined within a particular domain. It supports not null and check constraints.
- **Assertion constraints** are specified within an assertion definition, which will be discussed in Chapter 05.

NOTE The syntax for enforcing various constraints is given in Chapter 05.

Primary Key Constraint

The **primary key** constraint ensures that the attributes, which are

declared as primary key must be unique and not *null*. Primary key constraint enforces *entity integrity* for the relation and constraints the relation in the following ways:

- The primary key attribute must have unique set of values for all the tuples in a relation.
- The primary key attribute cannot have *null* value.
- A relation can have only one primary key attribute.

Unique Constraint

The **unique** constraint ensures that a particular set of attributes contains unique values and hence, two tuples cannot have duplicate values in specified attributes. Like primary key, it also enforces *entity integrity*. For example, the unique constraint can be specified on `Pname` attribute of `PUBLISHER` relation to prevent tuples with duplicate publisher names.

Unique constraint is preferable over primary key constraint to enforce uniqueness of a non-primary key attribute. This is because

- Multiple unique constraints can be applied on a relation, whereas only one primary key constraint can be defined on a relation.
- Unlike primary key constraint, unique constraint allows the attribute value to be *null*.

Primary key and unique key constraints act as a parent key when applying a foreign key constraint. Parent key is the key that identifies the tuples of the relation uniquely.

Check Constraint

The **check** constraint can be used to restrict an attribute to have values in a given range, that is, for each tuple in a relation, the values in a certain attribute, must satisfy a given condition. One way to achieve this is by applying check constraint during the declaration of the relation. For example, if the check constraint `(PRICE>20)` is applied on the attribute `Price` of the `BOOK` relation, it ensures that the value entered into the

attribute `Price` must be greater than *$20*.

Check constraints can be applied on more than one attribute simultaneously. For example, in the attributes `Copyright_date` and `Published_date`, constraints can be applied in such a way that `Copyright_date` must be either less than or equal to the `Published_date`.

NOTE If the inserted value violates the check constraint, the insert or update operations will not be permitted.

The check constraint can also be applied during domain declaration. It allows specifying a condition that must be satisfied by any value assigned to an attribute whose type is the domain. For example, the check constraint can ensure that the domain, say `dom`, allows values between *20* and *200*. If domain `dom` is assigned to the attribute `Price` of a `BOOK` relation, it ensures that the attribute `Price` must have values between *20* and *200*. As a result, if a value that is not between *20* and *200* is inserted into the constrained attribute, an error occurs.

Not Null Constraint

Every attribute has a nullability characteristic that shows whether a particular attribute accepts a *null* value or not. By default, every attribute accepts *null* values but this default nullability characteristic can be overridden by using the **not null** constraint. The not null constraint ensures that the attribute must not accept *null* values. For example, consider a tuple in the `BOOK` relation where the attribute `ISBN` contains a *null* value. Such a tuple provides book information for an unknown `ISBN` and hence, it does not provide appropriate information. In such case, *null* value must not be allowed and this can be done by constraining the attribute `ISBN` using not null constraint. Here, the not null constraint prevents the *null* value to be inserted into the attribute `ISBN`. Any attempt to change the attribute value of `ISBN` to *null* value results in an error. Note that the *null* values are not allowed in the primary key attribute of a relation. Since `ISBN` is a primary key attribute of `BOOK` relation, it cannot

accept *null* values. Hence, it is not necessary to explicitly specify not null constraint for the attribute `ISBN`.

The not null constraint enforces *domain integrity* by ensuring that the attribute of a particular domain is not permitted to take *null* values. For example, a domain, say `dom2`, can be restricted to take non-*null* values. If the domain `dom2` is assigned to the attribute `Category` of a `BOOK` relation, it ensures that the attribute `Category` must have some values. As a result, if *null* value is inserted into the constrained attribute, it will not be accepted as it violates the not null constraint.

Foreign Key Constraint

A **foreign key** constraint allows certain attributes in one relation to refer to attributes in another relation. The relation on which foreign key constraint is defined contains the partial information. Its detailed information can be searched from another relation with a matching entry. In this case, a foreign key constraint will not allow the deletion of a tuple from the relation with detailed information if there is a tuple in the relation with partial information exist. In other words, a foreign key constraint not only controls the data that can be stored in the referencing relation but it also controls the changes made to the referenced relation.

For example, if the tuple for a publisher is deleted from the `PUBLISHER` relation, and the publisher's ID is used in the `BOOK` relation, the deleted publisher's ID becomes orphaned in the `BOOK` relation. This situation can be prevented by a foreign key constraint. This constraint ensures that changes made to the data in the `PUBLISHER` relation will not make the data in the `BOOK` relation orphan.

Foreign key constraint enforces *referential integrity* by ensuring that an attribute in a relation `S` whose value in each tuple either matches with the values of the primary key in another relation `R` or is *null*. For example, the foreign key attribute `R_ID` in the `REVIEW` relation must have either the same values as that of the primary key `A_ID` of the `AUTHOR` relation or must

be *null*.

Learn More

It is not necessary that the foreign key constraint can only be related with the primary key constraint in another relation. A foreign key constraint can also be defined to refer to the attributes of a unique constraint in another relation.

3.6 CONSTRAINT VIOLATION WHILE UPDATING DATA

In addition to storing the data, various operations such as add, remove, and modify can also be performed on the relation. These operations must be performed in such a way that various integrity constraints should not be violated. The update operations on the relation are *insert, delete*, and *update* (or *modify*).

3.6.1 The Insert Operation

The **insert** operation creates a new tuple or a set of tuples into a relation and provides attribute values to the inserted tuples. These inserted values must satisfy all the integrity constraints to maintain the consistency of the database. For example, the statement Insert <'*003-456-654-3*', '*Discrete Mathematics'*, '*Textbook*', *60, 2007, 2007, 650*, '*P002*' > satisfies all the integrity constraints and hence, these values can be inserted into the BOOK relation.

However, if the attribute values violate one or more integrity constraints, the attribute values are rejected and the insert operation cannot be performed. Consider some cases given here.

- If the attribute value is inserted into a tuple of BOOK relation, it must belong to the associated domain of possible values otherwise the domain integrity will be violated. For example, in the statement, Insert <'*002-678-999-5*', '*LINUX*', '*Textbook*', *300, 2006, 2007, 750*, '*P005*'> into BOOK, the price of the book does not belong to the specified domain, that is, between *$20* and *$200*. So, this statement

violates the domain integrity and hence, the values are rejected.

- If the attribute value is inserted into a primary key attribute of BOOK relation, it must be unique and non-*null* otherwise the entity integrity will be violated. For example, in the statement, Insert <'*003-456-654-3*', '*Software Engineering*', '*Textbook*', *75, 2005, 2006, 700*, '*P003*'> into BOOK, the ISBN of the book has the same value as the ISBN of the existing tuple. So, this statement violates the primary key constraint and hence, the values are rejected. Similarly, consider another statement, Insert < *null*, '*Software Engineering*', '*Textbook*', *75, 2005, 2006, 700*, '*P004*'> into BOOK. In this statement, the ISBN of the book does not have any value, so it violates the entity integrity and hence, the values are rejected.
- If the attribute value is inserted into a foreign key attribute of BOOK relation, it must exist in the PUBLISHER relation otherwise the referential integrity will be violated. For example, in the statement, Insert <'*004-765-558-5*', '*Introduction to Japanese Language*', '*Language Book*', *45, 2007, 2007, 550*, '*P007*'> into BOOK, the P_ID of the BOOK relation does not exist in the P_ID of the PUBLISHER relation. So, this statement violates the referential integrity and hence, the values are rejected.
- If the attribute value is inserted into a foreign key attribute of BOOK relation, it must be logically correct otherwise the semantic integrity will be violated. For example, in the statement, Insert <'*004-765-558-5*', '*Introduction to French Language*', '*Language Book*', *45, 2007, 2007, 0*, '*P003*'> into BOOK, the number of pages is zero, which is logically incorrect. So, this statement violates the semantic integrity and hence, the values are rejected.

3.6.2 The Delete Operation

A tuple or a set of tuples can be deleted from a relation using the **delete** operations. Special attention must be given to the tuples that are being referenced by other tuples in the database, since it can violate the referential integrity. If deletion violates the referential integrity then it is

either rejected or cascaded. For example, if an attempt is made to delete the tuple with P_ID = "P004" in PUBLISHER relation, the delete operation is rejected as it violates the referential integrity. On the other hand, if the tuple of PUBLISHER relation with P_ID = "P005" is deleted, the attribute values of this tuple will be deleted since this tuple is not referenced by any tuple of BOOK relation and does not violate the referential integrity.

Deletion can be cascaded by deleting the tuples in the referencing relation that references the tuple to be deleted in the referenced relation. For example, in order to delete P_ID = "P004" from PUBLISHER relation, the tuples with P_ID = "P004" of BOOK relation must be first deleted. In addition to rejecting and cascading the deletion, the tuples that reference the tuple to be deleted can be modified either to *null* value or to another existing tuple. Note that assigning the *null* value to the tuple must not violate the entity integrity. If it violates then the *null* value should not be assigned to the tuple.

Note that it is not possible to cascade the deletion in each case. To understand this concept, consider the modified form of *Online Book* database in which the relation BOOK is altered by adding an attribute A_ID. The resultant relation can be given as shown in Figure 3.11.

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID	A_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001	A005
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001	A007
001-987-760-9	C++	Textbook	40	2004	2005	800	P001	A001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001	A003
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002	A006
:								
:								

Fig. 3.11 *The modified* BOOK *relation*

If an attribute value *A001* has to be deleted from the AUTHOR relation, the corresponding tuple with the attribute value *A001* must also be deleted from the modified BOOK relation. If this is done, in that case the corresponding book as well as publisher information of that particular author will also be deleted. So in this case, the attribute value of the author (whose tuple has to be deleted) in the modified BOOK relation is updated to *null* value instead of deleting that tuple.

3.6.3 The Update Operation

The attribute values in a tuple or a set of tuples can be modified using the **update** operation. Update operation changes the existing value of the attribute in a tuple to the new and valid value. It ensures that the new value satisfies all the integrity constraints. For example, if the value of the attribute `Price` with `ISBN= "001-987-760-9"` is updated from *$25* to *$50*, the changes are reflected in a relation since it satisfies all the integrity constraints.

Things to Remember

If a referencing attribute is a part of the primary key and violates the referential integrity, we cannot set it to *null* value, since doing this would result in the violation of entity integrity.

Changing the value of attributes does not create any problem as long as the valid values are entered in a relation. However, changing the value of primary key attribute or foreign key attribute can be a critical issue. Since the primary key attribute identifies each tuple, changing this attribute value must satisfy all the integrity constraints. For example, if the value of the attribute `P_ID` in PUBLISHER relation is updated from *P001* to *P002*, the changes are reflected in a relation only if it satisfies all the integrity constraints.

Similarly, the foreign key attribute must be modified in such a manner,

which ensures that the new value either refers to the existing value in the referenced relation or is *null*. For example, if the P_ID of the BOOK relation with ISBN= "001-987-760-9" is updated to *null* or any other value that exists in the attribute P_ID of PUBLISHER relation, the value of the foreign key attribute is modified. However, if the new value of P_ID of the BOOK relation does not exist in the P_ID of the PUBLISHER relation, it violates the referential integrity and hence, the values are rejected.

3.7 MAPPING E-R MODEL TO RELATIONAL MODEL

E-R model provides a conceptual model of the real-world concepts, which is represented in a database. To represent the E-R database schema to the relation schema, the relational model is used. In other words, E-R model is mapped to the relational model by representing E-R database schema by a collection of relation schemas.

The mapping from E-R model to relational model is possible due to the reason that both the models represent the real world logically and follow similar design principles. The *Online Book* database is used as an example to illustrate the mapping from E-R database schema to a collection of relation schema. Each entity type and relationship type in the *Online Book* database is represented as a unique relation schema and relation to which the name of the corresponding entity type or relationship type is assigned. The E-R schema of *Online Book* database is shown in Figure 3.12. In this section, the steps to translate an E-R database schema into a collection of relation schema and relations are shown.

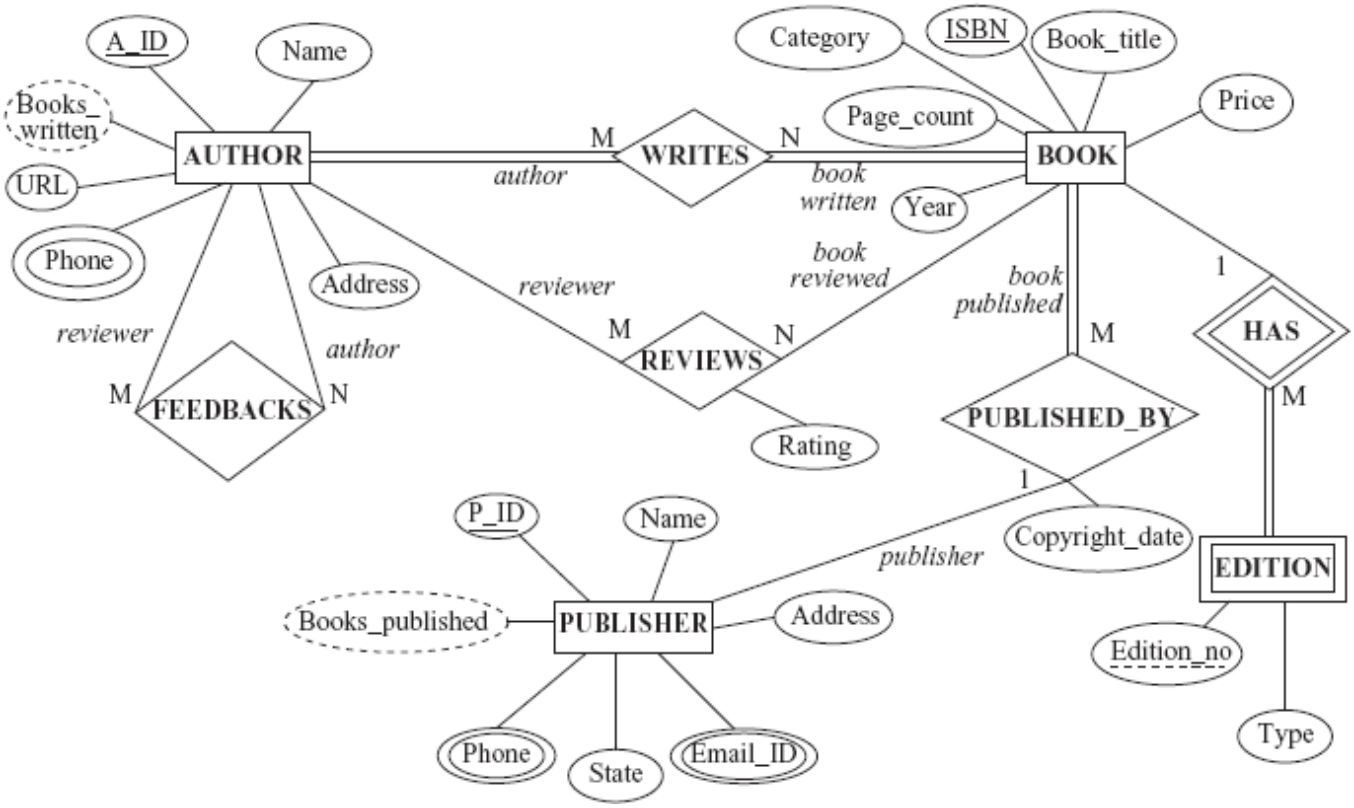


Fig. 3.12 *E-R schema for the Online Book database*

3.7.1 Representation of Entity Types to Relation

An entity type is mapped to a relation by representing different attributes $A_1, A_2, \ldots, A_n$ of an entity type `E` as attributes $A_1, A_2, \ldots, A_n$ of a relation `R`. Each entity instance of the entity type E represents a tuple in the relation `R`, and the primary key of the entity type `E` becomes the primary key of the relation `R`.

Consider the E-R diagram of the entity type `BOOK` in Figure 3.13. The entity type `BOOK` has six attributes, namely, `ISBN`, `Book_title`, `Category`, `Price`, `Year`, and `Page_count`.

Fig. 3.13 *The `BOOK` entity type*

The entity type `BOOK` can be represented by a relation schema `BOOK` with six attributes

```
BOOK(ISBN, Title, Category, Price, Year, Page_count)
```

As discussed earlier, the mapping from E-R model to relational model results in a relation. The resultant relation is known as **entity relation** since each entity instance of the entity type represents a tuple in a relation. A relation with six attributes of the BOOK schema is shown in Figure 3.14.

ISBN	Book_title	Category	Price	Year	Page_count
001-354-921-1	Ransack	Novel	22	2006	200
001-987-650-5	Differential Calculus	Textbook	30	2003	450
001-987-760-9	C++	Textbook	40	2005	800
002-678-880-2	Call Away	Novel	22	2002	200
002-678-980-4	DBMS	Textbook	40	2006	800
003-456-433-6	Introduction to German Language	Language Book	22	2004	200
003-456-533-8	Learning French Language	Language Book	32	2006	500
004-765-359-3	Coordinate Geometry	Textbook	35	2006	650
004-765-409-5	UNIX	Textbook	26	2007	550

Fig. 3.14 *The BOOK relation*

In this figure, the tuple <*'001-354-921-1'*, *'Ransack'*, *'Novel'*, *22, 2006, 200* > represents that the book with ISBN *'001-354-921-1'* has a book title *'Ransack'* under the category *'Novel'* with *200* pages, its price is *$22*, and it is published in the year *2006*.

NOTE The attributes `P_ID` and `Copyright_date` will be added into the BOOK relation in later steps.

3.7.2 Representation of Weak Entity Types

Representation of weak entity type with attributes $a_1, a_2, \ldots, a_m$ depends on the strong entity type with the attributes $A_1, A_2, \ldots, A_n$. Weak entity type must have total participation in the relationship type and must participate in one-to-many relationship with the strong entity type that defines it (that is, one strong entity type with many weak entity types). Weak entity type can be represented by the relation schema R with one attribute for each member of the set as

$$\{a_1, a_2, \ldots, a_m\} \cup \{A_1, A_2, \ldots, A_n\}$$

The primary key of the weak entity type consists of the primary key of the strong entity type as foreign key and its own partial key. In addition to the primary key, foreign key constraints are applied on the weak entity type as the primary key of the weak entity type refers to the primary key of the strong entity type.

Whenever a strong entity type is deleted, the related weak entity types must also be deleted..

For example, consider the E-R diagram of entity type BOOK having weak entity type EDITION shown in Figure 3.15. The weak entity type EDITION has two attributes, namely, Edition_no and Type.

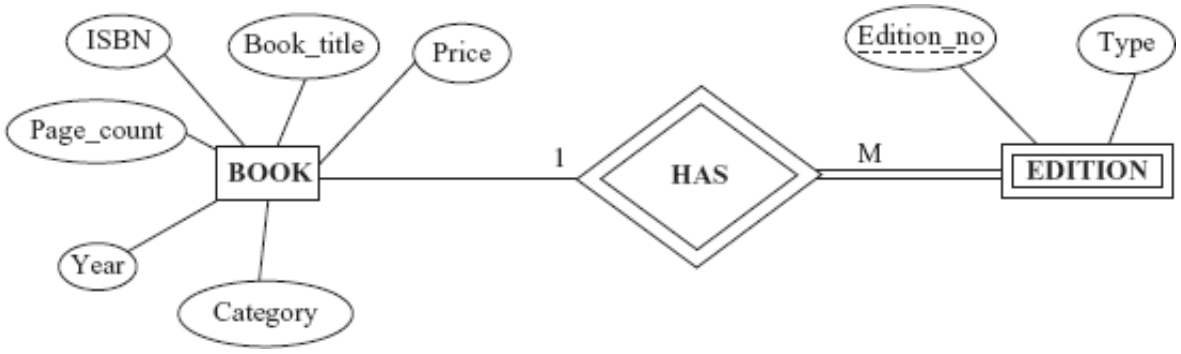


Fig. 3.15 *The entity type BOOK with weak entity type EDITION*

The primary key of the entity type BOOK, on which weak entity type EDITION depends, is ISBN. Thus, the relation schema of the weak entity type EDITION can be represented as

```
EDITION= (ISBN, Edition_no, Type)
```

In this statement, the primary key consists of the primary key of BOOK, along with the partial key of Edition, which is Edition_no. The foreign key constraint can be created on the Edition schema with the attribute ISBN referencing the primary key attribute of the BOOK relation and Edition_no is its own partial key. The corresponding EDITION relation is shown in Figure 3.16.

3.7.3 Representation of Relationship Types

The relationship type R is mapped to a relation, which includes

- Primary key of each participating entity type as a foreign key
- The set of descriptive attributes (if any) of R

Generally, primary key attributes of the participating entity types are chosen to identify a specific tuple in a binary relationship types. The primary key is chosen as shown in Table 3.1.

Consider the relationship type REVIEWS in the E-R diagram of Figure 3.17. Each author can review one or more books and similarly, one book can be reviewed by one or more authors.

For example, the relationship type REVIEWS has two entity types, namely, AUTHOR with the primary key A_ID, and BOOK with the primary key ISBN. Since the relationship type has one descriptive attribute, namely, Rating, the REVIEW schema of the relationship type REVIEWS has three attributes that can be shown as

```
REVIEW(R_ID, ISBN, Rating)
```

As stated earlier, the same concept of author id can be given a different name in different relation in order to distinguish the role of a reviewer

from the author. Hence, the attribute A_ID of the AUTHOR relation is given a different name, that is, R_ID in the REVIEW schema.

ISBN	Edition_no	Type
001-354-921-1	First	International
001-987-650-5	Second	Limited
001-987-760-9	Third	Limited
002-678-880-2	First	International
002-678-980-4	Fourth	International
003-456-433-6	First	Limited
003-456-533-8	Second	Limited
004-765-359-3	Second	International
004-765-409-5	Third	International

Fig. 3.16 *The EDITION relation*

Since the relationship type is many-to-many, the primary key for the REVIEW relation is the union of the primary key of entity type AUTHOR, that is, R_ID and primary key of entity type BOOK, that is, ISBN. In addition, R_ID and ISBN cannot be *null* since these keys are primary keys and hence, not null constraint and unique constraint are implicit for these fields.

In addition to the primary key, foreign key constraints are also created on the REVIEW relation with attribute A_ID referencing the primary key of AUTHOR relation and attribute ISBN referencing the primary key of BOOK relation.

Table 3.1 *Identifying primary key for binary relationship types*

Relationship Type	Primary Key for Relation
one-to-one	Primary key of either entity type participating in the relationship type
one-to-many or many-to-one	Primary key of entity type on the ‘many’ side of the relationship type
many-to-many	Union of primary key of each participating

	entity type
n-ary without arrow on its edges	Union of primary key attributes from the participating entity types
n-ary with arrow on one of its edges	Primary key of the entity type not on the 'arrow' side of relationship type

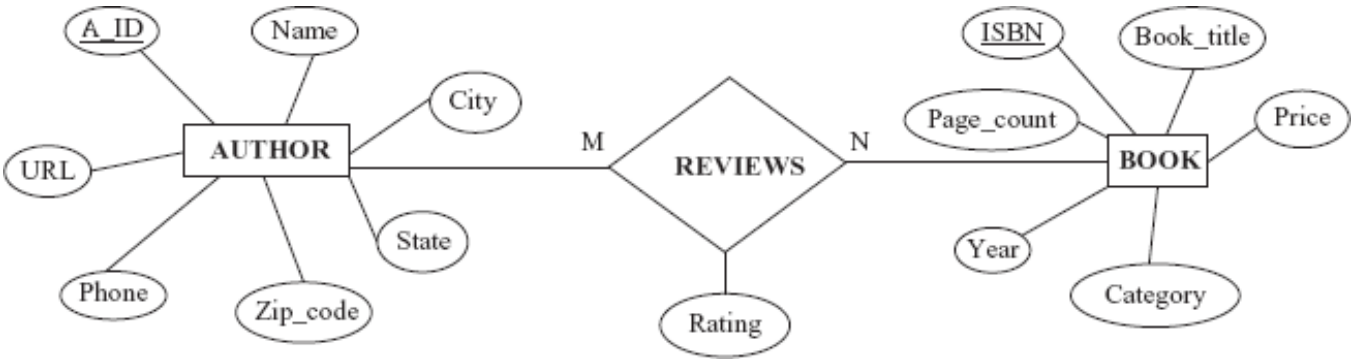


Fig. 3.17 *A relationship type* `REVIEWS`

The corresponding relation `REVIEW` of the relation schema is shown in the Figure 3.18.

Note that in the `AUTHOR` relation, no reference to the `BOOK` relation is given. Similarly, reference of `AUTHOR` relation is not given in the `BOOK` relation. This is because, in case of many-to-many relationship, this is logically implemented by including a separate relation. In this case, a `REVIEW` relation is included that relates `AUTHOR` and `BOOK` relations. The `REVIEW` relation links the primary key of `AUTHOR` relation with the primary key of the `BOOK` relation.

NOTE A relationship type relating a weak entity type to the corresponding strong entity type is not required in a relational database since it creates redundancy.

R_ID	ISBN	Rating
A001	002-678-980-4	2
A002	001-987-760-9	6
A003	002-678-980-4	5
A003	004-765-409-5	4
A004	003-456-533-8	9

A005	002-678-980-4	7
A006	001-354-921-1	7
A006	002-678-880-2	4
A007	004-765-359-3	3
A008	001-987-760-9	7
A009	001-987-650-5	8
A010	003-456-433-6	5

Fig. 3.18 *The* `REVIEW` *relation*

So far, we map each relationship type `R` to a different schema. Alternatively, the schema of `R` can be merged with the schema of one of the entity type that participates in `R`. Consider a many-to-one relationship type `PUBLISHED_BY` shown in Figure 3.19.

The double line in the Figure 3.19 indicates that the participation of `BOOK` in the relationship type `PUBLISHED_BY` is total. Hence, the book cannot exist without being associated with a particular publisher. Thus, we can combine the schema for `PUBLISHED_BY` with the schema of `BOOK`, and we require only two schemas, which are given here.

```
BOOK(ISBN, Book_title, Category, Price, Copy-right_date, Year,
Page_count, P_ID) PUBLISHER(P_ID, Pname, Address, State,
Phone, Email_id)
```

The corresponding relation `BOOK` can be given as shown in Figure 3.20.

The primary key of entity type, into whose schema the relationship type schema is combined, becomes the primary key of combined schema. So, the primary key of the combined schema is the primary key of `BOOK`, that is, `ISBN`.

The schema representing the relationship type would have foreign key constraints that refer to each of the participating entity type in the relationship type. Here, the foreign key constraint of the entity type, into

whose schema the relationship type schema is combined, is dropped and foreign key constraints of the remaining entity types are added to the combined schema. For example, the foreign key constraint that refers to the BOOK is dropped, whereas the foreign key constraint with P_ID that refers to PUBLISHER is retained as a constraint on the combined schema BOOK.

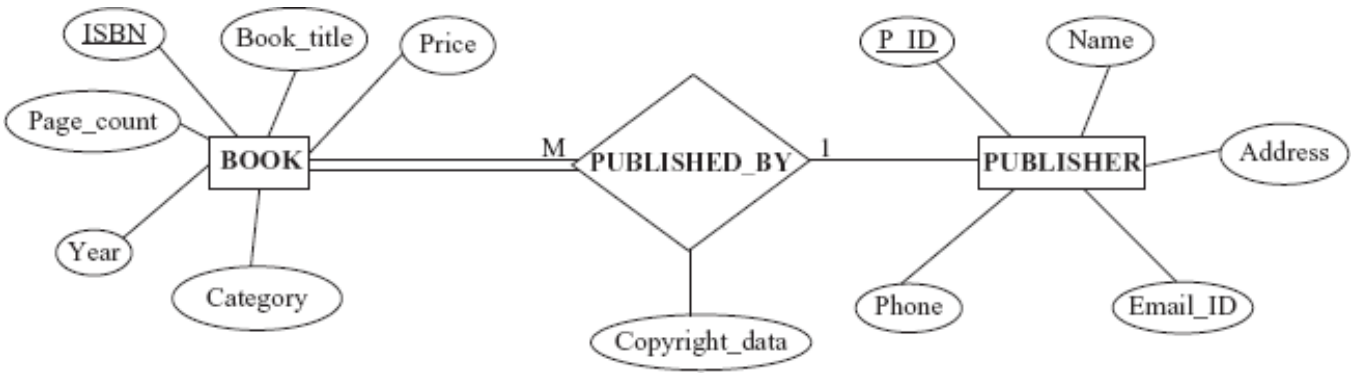


Fig. 3.19 *E-R diagram of relationship type* PUBLISHED_BY

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID
001-354-921-1	Ransack	Novel	22	2005	2006	200	P001
001-987-650-5	Differential Calculus	Textbook	30	2003	2003	450	P001
001-987-760-9	C++	Textbook	40	2004	2005	800	P001
002-678-880-2	Call Away	Novel	22	2001	2002	200	P002
002-678-980-4	DBMS	Textbook	40	2004	2006	800	P002
003-456-433-6	Introduction to German Language	Language Book	22	2003	2004	200	P004
003-456-533-8	Learning French Language	Language Book	32	2005	2006	500	P004
004-765-359-3	Coordinate Geometry	Textbook	35	2006	2006	650	P003
004-765-409-5	UNIX	Textbook	26	2006	2007	550	P003

Fig. 3.20 *The* BOOK *relation*

NOTE Even if the participation of the entity type in the relationship type is partial, the schemas can be combined by using null values.

For one-to-one relationship type, the schema of the relationship type can be merged with the relation schema of either of the entity types.

3.7.4 Representation of Composite and Multivalued Attributes

As discussed in Section 3.6.1, different attributes of an entity type become the attributes of a relation. If an entity type has composite attributes, no separate attribute is created for the composite attribute itself. Instead, separate attributes for each of its component attributes are created.

For example, consider the E-R diagram of the entity type `AUTHOR` with the composite attribute `Address` shown in Figure 3.21.

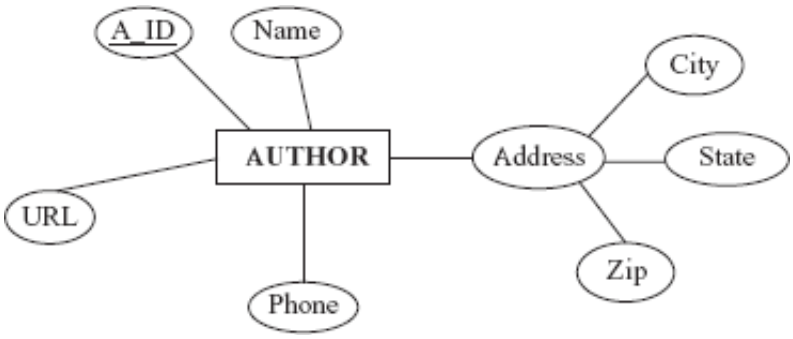


Fig. 3.21 *E-R diagram of entity type* `AUTHOR`

The attribute `Address` of the entity type `AUTHOR` is a composite attribute with the component attributes, namely, `City`, `State`, and `Zip`. The separate attributes are created for these component attributes and the schema of the `AUTHOR` relation is given here.

```
AUTHOR(A_ID, Aname, State, City, Zip, Phone, URL)
```

For the multivalued attribute, a separate relation is created with the primary key of entity type and with an attribute corresponding to the multivalued attribute of the entity type. For example, the attribute `Phone` of entity type `AUTHOR` is a multivalued attribute. Consider the E-R diagram of multivalued attribute `Phone` of entity type `AUTHOR` in Figure 3.22.

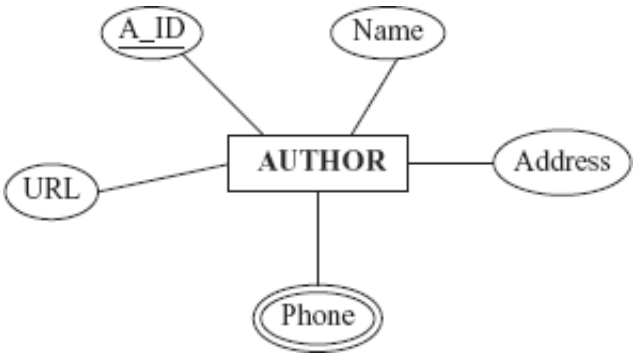


Fig. 3.22 *The entity type* AUTHOR *with multivalued attributes*

Multivalued attribute Phone of entity type AUTHOR can be represented by the relation schema as

```
AUTHOR_PHONE(A_ID,Phone)
```

And the corresponding relation, say AUTHOR_PHONE, can be given as shown in Figure 3.23.

A_ID	Phone
A001	923673
A001	923743
A002	376045
A002	376123
A003	419456
A003	419562
A004	578123
A004	578932

Fig. 3.23 *The* AUTHOR_PHONE *relation*

Note that each value of the multivalued attribute Phone is represented in a separate tuple of the AUTHOR_PHONE relation. For example, an author with A_ID as *A001* and Phone as *923673* and *92374* is represented in two tuples, namely, {*A001, 923673*} and {*A001, 923743*} in the AUTHOR_PHONE relation.

If the multivalued attribute is an alternate key of the entity type, the primary key of new relation will be the combination of the primary key and the multivalued attribute of the entity type.

3.7.5 Representation of Extended E-R Features

So far, we have discussed the representation of basic features of E-R model. In this section we discuss the representation of extended E-R features, which include generalization/specialization and aggregation. The representation of generalization involves two different approaches of designing the relation. The first approach creates separate relation for the higher-level entity type as well as for all the lower-level entity types. The relation for higher-level entity type includes its primary key attributes and all other attributes. Each relation for lower-level entity type includes the primary key of higher-level entity type and its own attributes. The primary key attribute of lower-level entity type refers to the primary key attribute of the higher-level entity type and hence, creates the foreign key constraint.

For example, consider the EER diagram of entity type BOOK representing generalization in Figure 3.24.

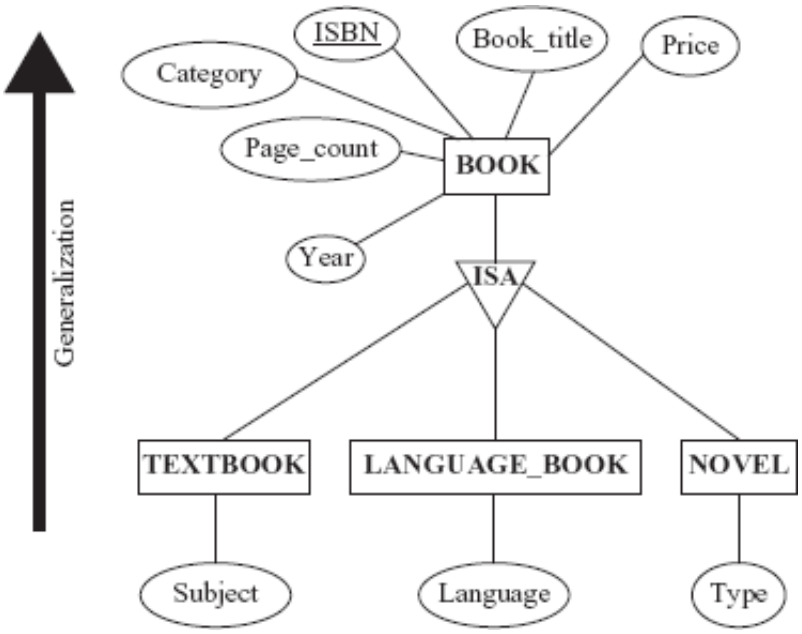


Fig. 3.24 *EER diagram representing generalization*

In this figure, the higher-level entity type is BOOK and the lower-level entity

types are TEXTBOOK, LANGUAGE_BOOK, and NOVEL. The relation schemas for the higher-level and lower-level entity types can be given as

```
BOOK(ISBN, Book_title, Category, Price, Year, Page_count)

TEXTBOOK(ISBN, Subject)

LANGUAGE_BOOK(ISBN, Language)

NOVEL(ISBN, Type)
```

The corresponding relation BOOK is shown in Figure 3.14 and corresponding relations TEXTBOOK, LANGUAGE_BOOK, and NOVEL are shown in Figure 3.25.

In this figure, the primary key attribute ISBN of relations of lower-level entity types TEXTBOOK, LANGUAGE_BOOK, and NOVEL refers to the primary key attribute of the relation of higher-level entity type BOOK and hence, creates the foreign key constraints.

The second approach creates separate relations only for all the lower-level entity types. Each relation of lower-level entity type includes all the attributes of higher-level entity type and its own attributes. This approach is possible when the generalization is disjoint and complete.

TEXTBOOK

ISBN	Subject
001-987-650-5	Maths
001-987-760-9	Computer
002-678-980-4	Computer
004-765-359-3	Maths
004-765-409-5	Computer

LANGUAGE_BOOK

ISBN	Language
003-456-433-6	German
003-456-533-8	French

NOVEL

ISBN	Type
001-354-921-1	Fiction
002-678-880-2	Non-Fiction

Fig. 3.25 *The TEXTBOOK, LANGUAGE_BOOK, and NOVEL relations*

For example, the relation schemas of the lower-level entity types can be given as

```
TEXTBOOK(ISBN, Book_title, Category, Price, Year, Page_count,
Subject)

LANGUAGE_BOOK(ISBN, Book_title, Category, Price, Year,
Page_count, Language)

NOVEL(ISBN, Book_title, Category, Price, Year, Page_count,
Type)
```

The second approach has a limitation that the foreign key constraint cannot be created on the relations for lower-level entity type. This is because there is no relation existing to which a foreign key constraint on relations for lower-level entity type can refer. To overcome this limitation, a relation has to be created that contains at least the primary key attribute of the relation for higher-level entity type.

Note that the second approach is not used for overlapping generalization as it may lead to redundancy. For example, if a book is both a textbook as well as a language book, the attributes `Book_title`, `Price`, `Year`, and `Page_count` have to be stored twice. In addition, the second approach is not used for generalization that is not complete. For example, if a book is neither a textbook nor a language book then another relation, that is, `BOOK` has to be used to represent such books.

An EER diagram containing aggregation expresses relationship between the relationships and a collection of entities/entity types. When EER diagram with aggregation is transformed in the relation, the primary key of the relation includes the primary key of the entity type and the primary key of the relationship type. The descriptive attributes of the entity type are also included in the relation.

In addition to the primary key, foreign key constraint can be applied on the relationship types involving aggregation. The primary key of the

aggregation refers to the primary keys of the entity type and the relationship type.

For example, consider the EER diagram representing aggregation shown in Figure 3.26.

Since the primary key for the relationship type is the union of the primary keys of the entity types involved, the primary key for the relationship type WRITES is the union of the primary keys of the entity type BOOK and AUTHOR. Hence, the schema for the relationship type WRITES can be written as

```
{ISBN,A_ID}
```

The primary key of the relation, say COPYRIGHT, includes the primary key of the entity type PUBLISHER and the primary key of the relationship type WRITES. The descriptive attribute COPYRIGHT_

DATE of the relation COPYRIGHT is also included. Now, the schema for the relation COPYRIGHT can be represented as `{ISBN,A_ID,P_ID,Copyright_date}` and the relation is shown in Figure 3.27.

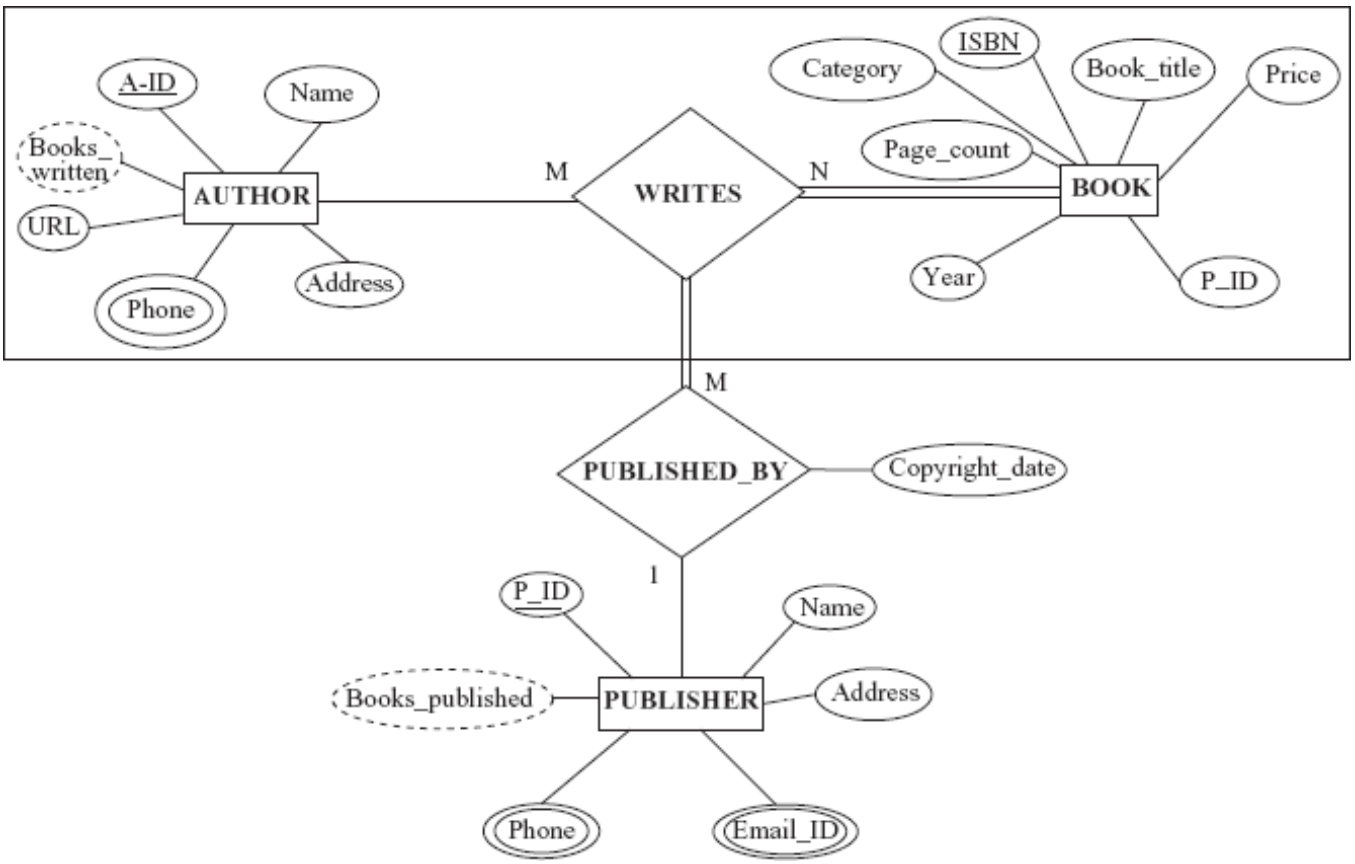


Fig. 3.26 *EER diagram with aggregation*

P_ID	ISBN	A_ID	Copyright_date
P001	001-987-760-9	A001	2004
P001	001-987-760-9	A003	2004
P001	001-354-921-1	A005	2005
P002	002-678-980-4	A002	2005
P002	002-678-980-4	A008	2005
P003	004-765-409-5	A008	2006
P003	004-765-359-3	A009	2006
P004	003-456-433-6	A004	2003
P001	001-987-650-5	A007	2003
P002	002-678-880-2	A006	2001
P004	003-456-533-8	A010	2005

Fig. 3.27 *The* `COPYRIGHT` *relation*

SUMMARY

1. A relational database is a collection of relations having distinct names. A relation is used to represent information about an entity and its relationship with other entities in the form of attributes (or columns) and tuples (or rows).
2. The relational model includes three components, namely, structural component (or data structures), manipulative component (or data manipulation), and integrity constraints.
3. Structural component consists of two components, namely, entity types and their relationships, which act as a framework of the given situation.
4. Manipulative component consists of a set of operations that are performed on a table or group of tables.
5. Integrity constraints are a set of rules that governs the entity types and their relationships, and thereby, ensures the integrity of the database.
6. A relation comprises of a relation schema and a relation instance. A relation schema depicts the attributes of the table and a relation instance is a two-dimensional table with a time-varying set of tuples.
7. A relation has certain characteristics. The first is that the tuples in a relation do not have any specified order. The second is that the order in which the values appear in the tuples is relevant. The third is that each tuple in a relation contains a single value for each of its attribute. Finally, each tuple in a relation must be uniquely identified by its contents.
8. The fundamental function of the DBMS is to maintain the integrity of the data. Data integrity ensures that the data in the database is consistent, accurate, correct, and valid.
9. Data integrity is of four types, namely, domain integrity, entity integrity, referential integrity, and semantic integrity.
10. Domain integrity can be specified for each attribute by defining its specific range or domain.
11. Entity integrity assigns restriction on the tuples of a relation and ascertains the accuracy and consistency of the database.

12. Referential integrity ensures that the value of foreign key in the referencing relation must exist in the primary key attribute of the referenced relation, or that the value of the foreign key is null.
13. Semantic integrity ensures that the data in the database is logically consistent and complete with respect to the real world.
14. Integrity constraints are the rules defined on a relational database schema and satisfied by all the instances of a database in order to model the real-world concepts correctly.
15. The primary key constraint ensures that the attributes, which are declared as primary key must be unique and not null.
16. The unique constraint ensures that a particular set of attributes contains unique values and hence, two tuples cannot have duplicate values in specified attributes.
17. The check constraint can be used to restrict an attribute to have values in a given range, that is, for each tuple in a relation, the values in a certain attribute must satisfy a given condition. One way to achieve this is by applying check constraint during the declaration of the relation.
18. The check constraint can also be applied during domain declaration. It allows specifying a condition that must be satisfied by any value assigned to an attribute whose type is the domain.
19. The not null constraint ensures that the attribute must not accept null values.
20. A foreign key constraint allows certain attributes in one relation to refer to attributes in another relation.

KEY TERMS

- Referencing relation
- Relational model
- Relational database
- Relation
- Relation schema
- Domain
- Unstructured domain

- Structured domain
- Relation instance
- Attributes and Degree
- Tuples and Cardinality
- Unary relation
- Binary relation
- Ternary relation
- Relational database schema
- Relational database instance
- Superkey
- Candidate key
- Primary key
- Alternate key
- Foreign key
- Referenced relation
- Self-referencing foreign key
- Domain integrity
- Entity integrity
- Referential integrity
- Semantic integrity
- Integrity constraints
- Legal and illegal relations
- Table-related constraints
- Column-level constraint
- Table-level constraint
- Domain constraints
- Assertion constraints
- Primary key constraint
- Unique constraint
- Check constraint
- Not null constraint
- Foreign key constraint
- Insert, Delete, and Update operations

EXERCISES

A. Multiple Choice Questions

1. The relational model is based on the branches of _____.
    1. Science
    2. Engineering
    3. Mathematics
    4. None of these
2. The relational model includes:
    1. Structural components
    2. Manipulative components
    3. Integrity constraints
    4. All of these
3. Which of the following is not a constituent of a relation schema?
    1. Tuples
    2. Set of attributes
    3. Relation name
    4. Both (b) and (c)
4. Candidate key is also known as _____.
    1. Primary key
    2. Irreducible superkey
    3. Superkey
    4. Foreign key
5. In case more than one attribute forms the primary key, the key is known as _____.
    1. Foreign key
    2. Alternate key
    3. Candidate key
    4. Composite key
6. Which of the following integrity is applied on the tuple?
    1. Domain integrity
    2. Entity integrity
    3. Referential integrity
    4. Semantic integrity
7. Integrity constraints can be categorized into _____ parts.
    1. One

    2. Two
    3. Three
    4. Four
8. Which of the following can only be defined as column-level constraints?
    1. Not null constraint
    2. Foreign key constraint
    3. Primary key constraint
    4. Check constraint
9. Which of the following is not true with primary key constraint
    1. Unique key constraint is preferable
    2. Primary key attribute cannot have *null* value over primary key constraint
    3. Primary key constraint enforces entity integrity
    4. Multiple primary key constraints can be defined on a relation
10. While mapping relationship types to a relation, which of the following relationship types allows us to choose primary key of either participating entity type as the primary key of the relation?
    1. One-to-one
    2. One-to-many
    3. Many-to-many
    4. All of these

B. Fill in the Blanks

1. The relational model represents both the data and the relationships among data using _____.
2. The number of attributes in a relation is known as _____ or _____.
3. In case, foreign key refers to its own relation, it is known as _____ foreign key.
4. Entity integrity assigns restriction on the _____ of a relation and ascertains the accuracy and consistency of the database.
5. The update operations on the relation are _____, delete, and update (or modify).

6. If deletion violates the _____ integrity, then it is either rejected or _____.
7. The mapping from E-R to relational model results in a _____.
8. Whenever a strong entity type is deleted, the _____ must also be deleted.
9. If an entity type has composite attribute, no _____ is created for the composite attribute itself.
10. The E in EER stands for _____.

C. Answer the Questions

1. Define a relational database, and discuss the various components of a relational model.
2. State the difference between the following terms with the help of suitable examples.
    1. Relation schema and relation instance
    2. Relation database schema and relational database instance
    3. Primary key and foreign key
3. Define the following terms:
    1. Degree
    2. Relation
    3. Cardinality
    4. Unary relation
    5. Binary relation
    6. Ternary relation
    7. Alternate key
4. What are the characteristics of relations?
5. All candidate keys are superkeys, whereas all superkeys are not candidate keys. Justify this statement with a suitable example.
6. What do you understand by the term data integrity? What are its four types? Elaborate entity integrity and referential integrity. Mention their differences and discuss the importance of each.
7. What are integrity constraints? Why are they important? Mention the various categories of integrity constraints with examples. Discuss how you enforce integrity constraints.

8. What is the difference between primary key constraint and unique constraint? Why is unique constraint preferred over primary key constraint?
9. Describe check constraint with the help of an example.
10. What is foreign key constraint? Explain its importance.
11. How does a not null constraint enforce domain integrity?
12. Discuss the various update operations that can be performed on a relation.

D. Practical Questions

1. Consider the following update operations performed on the database instance shown in Figure 3.5. Discuss the integrity constraints violated by each operation.
    1. Insert< *'002-678-999-6', 'Operating System', 'Textbook', 500, 2006, 2007, 700, 'P001'*> into `BOOK` relation.
    2. Insert< *'P003', 'Express Publications', '100, Second street, Atlanta', 'Georgia', 7134000, 'Express@publications.com'*> into `PUBLISHER` relation.
    3. Insert< *NULL, 'Richard Martin', 'Washington', 'Seattle', 98012570732, 'www.rmartin.com'*> into `AUTHOR` relation.
    4. Insert< *'004-765-200-1', 'VC++', 'Textbook', 40, 2007, 2008, 550, 'P010'*> into `BOOK` relation.
    5. Insert< *'002-876-680-2', 'Sunlight', 'Novel', 25, 2003, 2001, 250, 'P002'*> into `BOOK` relation.
    6. Insert< *'A001', '001-987-760-9', 8* > into `REVIEW` relation.
    7. Delete the tuple with `P_ID= "P004"` from `PUBLISHER` relation.
    8. Delete the tuple with `A_ID= "A003"` from `AUTHOR` relation.
    9. Delete the tuple with `ISBN= "004-765-409-5"` from `BOOK` relation.
    10. Modify the `Rating` attribute of the `REVIEW` relation with `R_ID= "A001"` and `ISBN= "002-678-980-4"` to `"6"`.
    11. Modify the `P_ID` attribute of the `PUBLISHER` relation to *'P002'*.
2. Consider the relation `STUDENT (Enroll#, SemRoll#, Stud_name, Gender, Course, Semester, Address, Phone_no)`. Specify the

candidate keys for this schema and the constraints under which each candidate key would be valid. What other constraints (if any) can be specified on this relation?

3. Consider the following relation schema for a UNIVERSITY database:

```
PROFESSOR(PID, P_Name, Dept_ID)
COURSE(Code, Dept_ID, C_Name, Syllabus)
PROF_COURSE(PID, Code, Semester)
```

   Identify the primary keys and foreign keys for this schema. Now populate these relations with some tuples, and then give an example of insertion in the PROF_COURSE relation that violates the referential integrity constraints and of another that does not. Identify what you think should be the various candidate keys for this schema. Make any assumption, wherever necessary.

4. Consider the following relation schema for the SALES database:

```
CUSTOMER(CustNo, Cust_Name, Address)
ORDER(OrderNo, Order_date, Cust_no, Qty, Amount)
PRODUCT(ProdNo, Price, Order_no)
```

   Specify the foreign key constraints for the SALES database. Also insert some tuples in the relations and show some examples of deletion of tuples that violate referential integrity constraints. Make any assumption, wherever necessary.

5. Construct appropriate relation schemas for the E-R diagrams given in Practical questions 1 and 5 in Chapter 02. Make appropriate assumptions, if necessary.
6. Construct appropriate relation schemas for the COMPANY, BANK databases given in Practical questions 2 and 6, respectively, in Chapter 02. Make appropriate assumptions, wherever necessary.