CHAPTER 1

SQL AND DATA

Chapter Objectives

In this chapter, you will learn about:

What is SQL? SQL (pronounced *sequel*) is an acronym for *Structured Query Language,* a standardized language used to access and manipulate data. The history of SQL corresponds closely with the development of relational databases concepts published in a paper by Dr. E. F. Codd at IBM in 1970. He applied mathematical concepts to the specification of a method for data storage and access; this specification, which became the basis for relational databases, was intended to overcome the physical dependencies of the then-available database systems. The SQL language (originally called "System R" in the prototype and later called "SEQUEL") was developed by the IBM Research Laboratory as a standard language to use with relational databases. In 1979 Oracle, then called Relational Software, Inc., introduced the first commercially available implementation of a relational database incorporating the SQL language. The SQL language evolved with many additional syntax expansions incorporated into the American National Standards Institute (ANSI) SQL standards developed since. Individual database vendors continuously added extensions to the language, which eventually found their way into the latest ANSI standards used by relational databases today. Large-scale commercial implementations of relational database applications started to appear in the mid to late 1980s as early implementations were

hampered by poor performance. Since then, relational databases and the SQL language have continuously evolved and improved.

Before you begin to use SQL, however, you must know about data, databases, and relational databases. What is a database? A *database* is an organized collection of data. A *database management system* (DBMS) is software that allows the creation, retrieval, and manipulation of data. You use such systems to maintain patient data in a hospital, bank accounts in a bank, or inventory in a warehouse. A *relational database management system* (RDBMS) provides this functionality within the context of the relational database theory and the rules defined for relational databases by Codd. These rules, called "Codd's Twelve Rules," later expanded to include additional rules, describe goals for database management systems to cope with ever-challenging and demanding database requirements. Compliance with Codd's Rules has been a major challenge for database vendors and early versions of relational databases and many desktop databases complied with only a handful of the rules.

Today, SQL is accepted as the universal standard database access language. Databases using the SQL language are entrusted with managing critical information affecting many aspects of our daily lives. Most applications developed today use a relational database and Oracle continues to be one of the largest and most popular database vendors. Although relational databases and the SQL language are already over 30 years old, there seems to be no slowing down of the popularity of the language. Learning SQL is probably one of the best long-term investments you can make for a number of reasons:

- SQL is used by most commercial database applications.
- Although the language has evolved over the years with a large array of syntax enhancements and additions, most of the basic functionality has remained essentially unchanged.
- SQL knowledge will continue to be a fundamental skill as there is currently no mature and viable alternative language that accomplishes the same functionality.
- Learning Oracle's specific SQL implementation provides you with great insight into the feature-rich functionality of one of the largest and most successful database vendors.

Understanding relational database concepts provides you with the foundation for understanding the SQL language. Those unfamiliar with relational concepts or interested in a refresher will receive an overview of basic relational theories in the next two labs. If you are already familiar with relational theory, you can skip the first two labs and jump directly to Lab 1.3, "The STUDENT Schema Diagram." The STUDENT database manages student enrollment data at a fictional university. Lab 1.3 teaches you about the organization and relationships of the STUDENT database, which is used throughout the exercises in this book.

LAB 1.1

DATA, DATABASES, AND THE DEFINITION OF SQL

LAB OBJECTIVES

After this lab, you will be able to:

- ✔ Identify and Group Data
- ✔ Define SQL
- ✔ Define the Structures of a RDBMS: Tables, Columns, Rows, and Keys

Data is all around you—you make use of it every day. Your hair may be brown, your flight leaves from gate K10, you try to get up in the morning at 6:30 A.M. Storing data in related groups and making the connections among them are what databases are all about.

You interact with a database when you withdraw cash from an ATM machine, order a book from a Web site, or check stock quotes on the Internet. The switch from the information processing society to the knowledge management society will be facilitated by databases. Databases provide a major asset to any organization by helping it run its business and databases represent the backbones of the many technological advances we enjoy today.

Before the availability of relational databases, data was stored in individual files that could not be accessed unless you knew a programming language. Data could not be combined easily and modifications to the underlying database structures were extremely difficult. The Relational Model conceived by E. F. Codd provided the framework to solve a myriad of these and many other database problems. Relational databases offer *data independence,* meaning a user does not need to know on which hard drive and file a particular piece of information is stored. The RDBMS provides users with *data consistency* and *data integrity*. For example, if an employee works in the Finance department and we know that he can only work

for one department, there should not be duplicate department records or contradicting data in the database. As you work through this lab, you will discover many of these useful and essential features. Let's start with a discussion of the terminology used in relational databases.

TABLES

A relational database stores data in tables. Tables typically contain data about a single subject. Each table has a unique name that signifies the contents of the data. For example, you can store data about books you read in a table called BOOK.

COLUMNS

Columns in a table organize the data further and a table consists of at least one column. Each column represents a single, low-level detail about a particular set of data. The name of the column is unique within a table and identifies the data you find in the column. For example, the BOOK table may have a column for the title, publisher, date the book was published, and so on. The order of the columns is unimportant because SQL allows you to display data in any order you choose.

ROWS

Each row usually represents one unique set of data within this table. For example, the row in Figure 1.1 with the title "The Invisible Force" is unique within the BOOK table. All the columns of the row represent respective data for the row. Each intersection of a column and row in a table represents a value and some do not, as you see in the PUBLISH_DATE column. The value is said to be *NULL*. Null is an unknown value, so it's not even blank spaces. Nulls cannot be evaluated or compared because they are unknown.

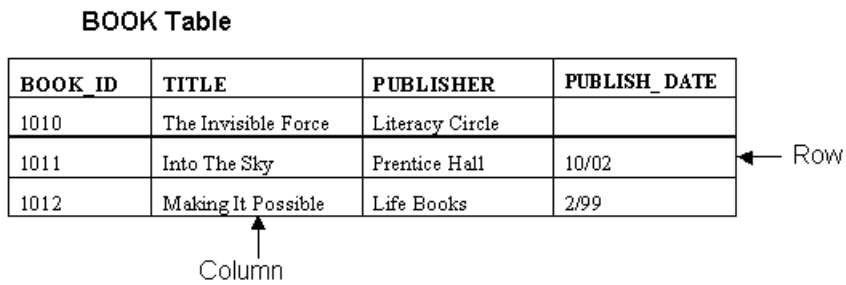


Figure 1.1 ■ Example of the BOOK table.

PRIMARY KEY

When working with tables, you must understand how to uniquely identify data within a table. This is the purpose of the *primary key*; it uniquely identifies a row within a table, which means that you find one, and only one row in the table by

CUSTOMER_ID	CUSTOMER_NAME	ADDRESS	PHONE	ZIP
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095
2011	Acme Mfg. Ltd.	555 Broadway	212-566-1212	10004
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010

PRIMARY KEY

Figure 1.2 ■ Primary key example.

looking for the primary key value. Figure 1.2 shows an example of the CUSTOMER table with the CUSTOMER_ID as the primary key of the table.

At first glance you may think that the CUSTOMER_NAME column can serve as the primary key of the CUSTOMER table because it is unique. However, it is entirely possible to have customers with the same name. Therefore, the CUSTOMER_NAME column is not a good choice for the primary key. Sometimes the unique key is a system-generated sequence number; this type of key is called a *synthetic* or *surrogate key*. The advantage of such a surrogate key is that it is unique and does not have any inherent meaning or purpose; therefore, it is not subject to changes. In this example, the CUSTOMER_ID column is such a surrogate key.

It is best to avoid any primary keys that are subject to updates as they cause unnecessary complexity. For example, the phone number of a customer is a poor example of a primary key column choice. Though it may possibly be unique within a table, phone numbers can change and then cause a number of problems with updates of other columns that reference this column.

A table may have only one primary key, which consists of one or more columns. If the primary key contains multiple columns it is referred to as a *composite primary key* or *concatenated primary key*. (Choosing appropriate keys is discussed more in Chapter 11, "Create, Alter, and Drop Tables.") Oracle does not require that every table have a primary key and there may be cases where it is not appropriate to have one. However, it is strongly recommended that most tables have a primary key.

FOREIGN KEYS

If you store the customer and the customer's order information in one table, the customer's name and address is repeated for each order. Figure 1.3 depicts such a table. Any change to the address requires the update of all the rows in the table for that individual customer.

If, however, the data is split into two tables (CUSTOMER and ORDER as shown in Figure 1.4) and the customer's address needs to be updated, only one row in the CUSTOMER table needs to be updated. Furthermore, splitting data this way avoids data inconsistency whereby the data differs between the different rows.

ID	CUSTOMER_ NAME	ADDRESS	PHONE	ZIP	ORDER ID	ORDER_ DATE	TOTAL_ ORDER
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095	100	12/23/01	$500
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095	102	7/20/02	$100
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095	103	8/25/02	$400
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095	104	9/20/02	$200
2011	Acme Mfg. Ltd.	555 Broadway	212-566-1212	10004	105	8/20/02	$900
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010	101	01/05/02	$600

Figure 1.3 ■ Example of CUSTOMER data mixed with ORDER data.

Eliminating redundancy is one of the key concepts in relational databases and this process, referred to as *normalization*, is discussed shortly.

Figure 1.4 illustrates how the data is split into two tables to provide data consistency. In this example, the CUSTOMER_ID becomes a *foreign key* column in the ORDER table. The foreign key is the column that links the CUSTOMER and ORDER table together. In this example, you can find all orders for a particular customer by looking for the particular CUSTOMER_ID in the ORDER table. The CUSTOMER_ID would correspond to a single row in the CUSTOMER table that provides the customer-specific information. The foreign key column CUSTOMER_ID happens to have the same column name in the ORDER table. This makes it easier to recognize the fact that the tables share common column values. Often the foreign key column and the primary key have identical column names, but it is not required. You will learn more about foreign key columns with the same and different names and how to create foreign key relationships in Chapter 11, "Create, Alter, and Drop

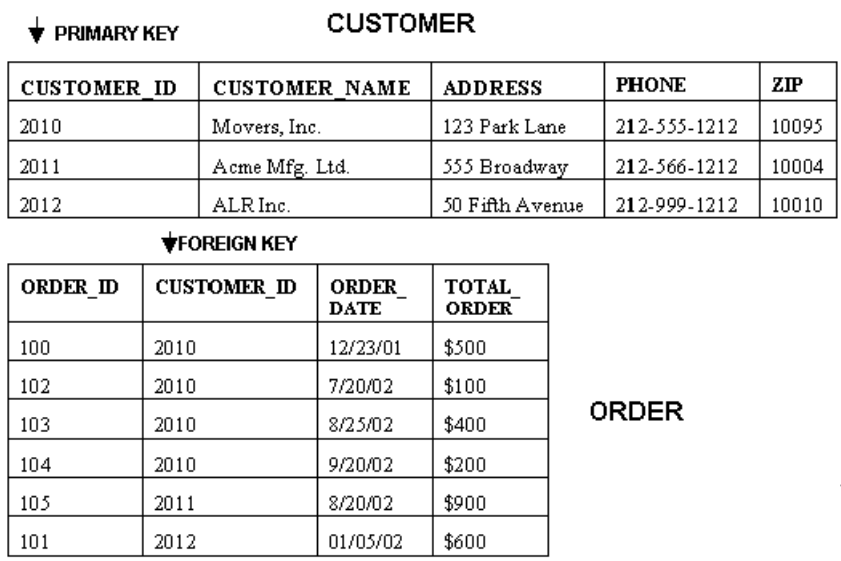


Figure 1.4 ■ Primary and foreign key relationship between CUSTOMER and ORDER tables.

Tables." Chapter 6, "Equijoins," teaches you how to combine results from the two tables using SQL.

You connect and combine data between tables in a relational database via data common columns.

SQL LANGUAGE COMMANDS

You work with the tables, rows, and columns using the SQL language. SQL allows you to query data, create new data, modify existing data, and delete data. Within the SQL language you can differentiate between individual sublanguages, which are a collection of individual commands.

For example, the *Data Manipulation Language* (DML) commands allow you to query, insert, update, or delete data. SQL allows you to create new database structures such as tables or modify existing ones; this subcategory of SQL language commands is called the *Data Definition Language* (DDL). Using the SQL language you can control access to the data using *Data Control Language* (DCL) commands. Table 1.1 shows you an overview of different language categories with their respective SQL commands.

Table 1.1 ■ Overview of SQL Language Commands

Description	SQL Commands
Data Manipulation	SELECT, INSERT, UPDATE, DELETE, MERGE
Data Definition	CREATE, ALTER, DROP, TRUNCATE, RENAME
Data Control	GRANT, REVOKE
Transaction Control	COMMIT, ROLLBACK, SAVEPOINT

One of the first statements you will execute is the SELECT command, which allows you to retrieve data. For example, to retrieve the TITLE and PUBLISHER columns from the BOOK table you may issue a SELECT statement such as the following:

```
SELECT title, publisher
  FROM book
```

The INSERT command lets you add new rows to a table. The next command shows you an example of an INSERT statement that adds a row to the BOOK table. The row contains the values Oracle SQL as a book title, a BOOK_ID of 1013, and a publish date of 12/02 with Prentice Hall as the publisher.

```
INSERT INTO book
(book_id, title, publisher, publish_date)
VALUES
(1013, 'Oracle SQL', 'Prentice Hall', '12/02')
```

To create new tables you use the CREATE TABLE command. The following statement illustrates how to create a simple table called AUTHOR with three columns. The first column, called AUTHOR_ID, holds numeric data; the FIRST_NAME and LAST_NAME columns contain alphanumeric character data.

```
CREATE TABLE author
(author_id   NUMBER,
 first_name  VARCHAR2(30),
 last_name   VARCHAR2(30))
```

You can manipulate the column definitions of a table with the ALTER TABLE command. This allows you to add or drop columns. You can also create primary and foreign key constraints on a table. Constraints allow you to enforce business rules within the database. For example, a primary key constraint can enforce the uniqueness of the AUTHOR_ID column in the AUTHOR table.

To grant SELECT and INSERT access to the AUTHOR table, you issue a GRANT command. It allows the user Scott to retrieve and insert data in the AUTHOR table.

```
GRANT SELECT, INSERT ON author TO scott
```

Starting with Chapter 2, "SQL: The Basics," you will learn how to execute the SELECT command against the Oracle database; Chapter 10, "Insert, Update, and Delete," will teach you the details of data manipulation; and Chapter 11, "Create, Alter, and Drop Tables," introduces you to the creation of tables and the definition of constraints to enforce the required business rules. Chapter 14, "Security," discusses how to control the access to data and the various Oracle database features.

LAB 1.1 EXERCISES

1.1.1 IDENTIFY AND GROUP DATA

a) Give three examples of types of data.

b) What groupings of data do you use in your daily life?

c) Give an example of a database system you use outside of the workplace and explain how it helps you.

1.1.2 DEFINE SQL

a) What is SQL and why is it useful?

b) Try to match each of the SQL commands on the left with a verb from the list on the right.

1. CREATE	a. manipulate
2. UPDATE	b. define
3. GRANT	c. control

c) Why do you think it is important to control access to data in a database?

1.1.3 DEFINE THE STRUCTURES OF A RDBMS: TABLES, COLUMNS, ROWS, AND KEYS

a) How is data organized in a relational database?

b) Do you think it's possible to have a table with no rows at all?

c) Figure 1.5 displays a listing of an EMPLOYEE and a DEPARTMENT table. Identify the columns you consider to be primary keys and foreign keys for the tables.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	DEPT_NO
230	Kyle	Hsu	80,000	40
231	Kirsten	Soehner	130,000	50
232	Madeline	Dimitri	70,000	40
234	Joshua	Hunter	90,000	20

DEPT_NO	DEPARTMENT_NAME
20	Finance
40	Human Resources
50	Sales
60	Information Systems

Figure 1.5 ■ EMPLOYEE and DEPARTMENT tables.

LAB 1.1 EXERCISE ANSWERS

1.1.1 ANSWERS

a) Give three examples of types of data.

Answer: The answer to this question will vary depending on your choices.

A circle, square, and triangle are all data about geometrical shapes. Your mother, father, and sister are data about your immediate family members. Fiction, comedy, cookbook, and computer are all data about types of books.

b) What groupings of data do you use in your daily life?

Answer: The answer to this question will vary depending on your situation.

I use my address book daily. It contains addresses and phone numbers of friends, relatives, and coworkers. I also keep a running to-do list of tasks at work, which groups together the tasks I have completed, as well as separately grouping those tasks I have yet to do.

When grouping data, each piece of data should be related to the others. A person's physical appearance is typically described by more than just brown hair; they may also have green eyes, be six feet tall, and be of the female sex. In my address book, I group together a person's name, address, and telephone number. I may keep a separate address book for my business contacts that would group together the person's name, company name, work telephone number, fax number, and email address.

c) Give an example of a database system you use outside of the workplace and explain how it helps you.

Answer: Again, the answer to this question will vary depending on your situation.

When I'm in a record store, I often use a computerized information kiosk to search for information about an album, such as where it is located in the store. Another example is an ATM machine, where I can inquire about my account balance.

1.1.2 ANSWERS

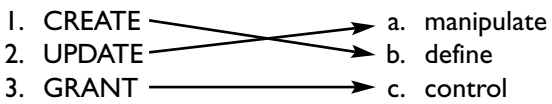
a) What is SQL and why is it useful?

Answer: SQL, the Structured Query Language, is a standardized relational database access language. It is useful because it allows a user to query, manipulate, define, and control data in a RDBMS.

The SQL language is sanctioned by ANSI, which determines standards on all aspects of the SQL language, including datatypes. However, most relational database products, including Oracle, have their own extensions to the ANSI standard, providing additional functionality within their respective products by further extending the use of SQL.

b) Try to match each of the SQL commands on the left with a verb from the list on the right:

Answer: The following shows how these commands match with the appropriate verb.



DML is used to *manipulate* data, with the SELECT, INSERT, UPDATE, and DELETE commands. (Note that in some of Oracle's own documentation, the SELECT command is not part of the DML language, but is considered Data Retrieval Language.) DDL is used to *define* objects such as tables with the CREATE, ALTER, and DROP commands. DCL is used to *control* access privileges in a RDBMS, such as with the GRANT and REVOKE commands to give or remove privileges. These SQL commands are written and executed against the database using a software program. In this workbook, Oracle's SQL*Plus program or *i*SQL*Plus with your Web browser is used to communicate these commands to the RDBMS. The use of SQL*Plus and SQL commands will be covered in Chapter 2, "SQL: The Basics."

c) Why do you think it is important to control access to data in a database?

Answer: Data can contain sensitive information to which some users should have limited access privileges. Some users may be allowed to query certain data but not change it, while others are allowed to add data to a database, but not delete it. By controlling access to data, the security of the data is assured for all users. You learn about safeguarding your data in Chapter 14, "Security."

1.1.3 ANSWERS

a) How is data organized in a relational database?

Answer: Data is organized by placing like pieces of information together in a table that consists of columns and rows.

For example, the data found in a library is typically organized in several ways to facilitate finding a book. Figure 1.6 shows information specific to books. The data is organized into columns and rows; the columns represent a type of data (title vs. genre), and the rows contain data. A table in a database is organized in the same way. You might call this table BOOK as it contains information related to books only. Each intersection of a column and row in a table represents a value.

TITLE	AUTHOR	ISBN#	GENRE	LOCATION_ID
Magic Gum	Harry Smith	0-11-124456-2	Computer	D11
Desk Work	Robert Jones	0-11-223754-3	Fiction	H24
Beach Life	Mark Porter	0-11-922256-8	Juvenile	J3
From Here to There	Gary Mills	0-11-423356-5	Fiction	H24

Figure 1.6 ■ BOOK table.

Searching for a book by location might yield this excerpt of data shown in Figure 1.7. This set of columns and rows represents another database table called LOCATION, with information specific to locations in a library.

LOCATION_ID	FLOOR	SECTION	SHELF
D11	1	3	1
H24	2	2	3
J3	3	1	1

Figure 1.7 ■ LOCATION table.

The advantage to storing information about books and their locations separately is that information is not repeated unnecessarily, and maintenance of the data is much easier.

For instance, two books in the BOOK table have the same LOCATION_ID, H24. If the floor, section, and shelf information were also stored in the BOOK table, this information would be repeated for each of the two book rows. In that situation, if the floor of LOCATION_ID H24 changed, both of the rows in the BOOK table would have to change. Instead, by storing the location information separately, the floor information only has to change once in the LOCATION table.

The two tables (BOOK and LOCATION) have a common column between them, namely LOCATION_ID. In a relational database, SQL can be used to query information from more than one table at a time, making use of the common column they contain by performing a *join*. The join allows you to query both the BOOK and LOCATION tables to return a list of book titles together with floor, section, and shelve information to help you locate the books easily.

b) Do you think it's possible to have a table with no rows at all?

Answer: Yes, it is possible, though clearly it is not very useful to have a table with no data.

c) Figure 1.5 displays a listing of an EMPLOYEE and its respective DEPARTMENT table. Identify the columns you consider to be primary keys and foreign keys for the tables.

Answer: The primary key of the EMPLOYEE table is the EMPLOYEE_ID. The primary key of the DEPARTMENT table is DEPT_NO. The DEPT_NO is also the foreign key column of EMPLOYEE table and is common between the two tables.

In the DEPT_NO column of the EMPLOYEE table you can ONLY enter values that exist in the DEPARTMENT table. The DEPARTMENT table is the parent table from which the child table, the EMPLOYEE table, gets its DEPT_NO values. Establishing a foreign key relationship highlights the benefit of *referential integrity*. Only valid primary key values from the parent table are allowed in the child's foreign key column, therefore avoiding *orphan rows* (child rows without parent rows). For example, you cannot enter a DEPT_NO of 10 in the EMPLOYEE table if such a value does not exist in the DEPARTMENT table.

Note that the DEPARTMENT table contains one row with the department number of 60, which does not have any corresponding employees. The referential integrity rule allows a parent without child(ren), but does not allow a child without a parent because this would be considered an orphan row. You will learn how to establish primary key and foreign key relationships in Chapter 11, "Create, Alter, and Drop Tables."

LAB 1.1 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) A university's listing of students and the classes they are enrolled in is an example of a database system.

a) _____ True
b) _____ False

2) A table must always contain both columns and rows.

a) _____ True
b) _____ False

3) SQL is software that interacts with a relational database.

a) _____ True
b) _____ False

4) More than one user can be connected to a database at the same time.

a) _____ True
b) _____ False

5) Referential integrity ensures that each value in a foreign key column of the child table links back to a matching primary key value in the parent table.

a) _____ True
b) _____ False

Answers appear in Appendix A, Section 1.1.

LAB 1.2

TABLE RELATIONSHIPS

LAB OBJECTIVES

After this lab, you will be able to:

- ✔ Read a Schema Diagram
- ✔ Identify Data Normalization Rules and Table Relationships
- ✔ Understand the Database Development Context

Although this is a book about SQL, you must understand the basic concepts, terminology, and issues involving database design to be able to understand why tables are organized in specific ways. This lab will introduce you to the practical aspects of designing tables and determining their respective relationships to each other.

DATA NORMALIZATION

The objective of *normalization* is the elimination of redundancy in tables, therefore avoiding any future data manipulation problems. There are a number of different rules for minimizing duplication of data, which are formulated into the various *normal forms*.

The rules verify that the columns you placed in the tables do in fact belong there. You design your tables, the appropriate columns, and the matching primary and foreign keys to comply with these rules. This process is called normalization. The normalization rules will be quite intuitive after you have read through the examples in this lab. Although there are many normalization rules, the *five normal forms* and the *Boyce–Codd normal form* (BCNF) are the most widely accepted. This lab will discuss the first three normal forms as programmers and analysts typically don't bother normalizing beyond third normal form; with the exception of experienced database designers.

FIRST NORMAL FORM

For a table to be in *first normal form*, all repeating groups must be removed and placed in a new table. The example in Figure 1.8 illustrates the repeating groups

BOOK Table

BOOK_ ID	TITLE	RETAIL _PRICE	LOCATION_1	LOCATION_2	LOCATION_3
1010	The Invisible Force	29.95	New York	San Francisco	
1011	Into The Sky	39.95	Chicago		
1012	Making It Possible	59.95	Miami	Austin	New York

Figure 1.8 ■ Repeating group.

in the BOOK table. The table has the location information of various warehouses across the country where the title is stocked. The location is listed in three columns as LOCATION_1 LOCATION_2, and LOCATION_3.

Imagine the scenario when you have more than three locations for a book. To avoid this and other problems, the database designer will move the location information to a separate table named BOOK_LOCATION, as illustrated in Figure 1.9. This design is more flexible and allows the storing of books at an unlimited number of locations.

BOOK Table

BOOK_ID	TITLE	RETAIL_ PRICE
1010	The Invisible Force	29.95
1011	Into The Sky	39.95
1012	Making It Possible	59.95

BOOK_LOCATION Table

BOOK_ID	LOCATION
1010	New York
1010	San Francisco
1011	Chicago
1012	Miami
1012	Austin
1012	New York

Figure 1.9 ■ Tables in first normal form.

SECOND NORMAL FORM

Second normal form states that all nonkey columns must depend on the entire primary key, not just part of it. This form only applies to tables that have composite primary keys. Figure 1.10 shows the BOOK_AUTHOR table with both the BOOK_ID and AUTHOR_ID as the composite primary key. In this example, authors with the ID 900 and 901 coauthored the book with the ID of 10002. If you add the author's phone number to the table, the second normal form is violated because the phone

BOOK_ID	AUTHOR_ID	ROYALTY_SHARE	AUTHOR_PHONE_NO
10001	900	100	212-555-1212
10002	901	75	901-555-1212
10002	900	25	212-555-1212
10003	902	100	899-555-1212

Figure 1.10 ■ Violation of second normal form in the BOOK_AUTHOR table.

number is dependent only on the AUTHOR_ID, not on the BOOK_ID. Note ROYALTY_SHARE is dependent completely on the combination of both columns because the percentage of the royalty varies from book to book and is split among authors.

THIRD NORMAL FORM

The *third normal form* goes a step further than the second normal form: It states that every nonkey column must be a fact about the primary key column. The third normal form is quite intuitive. Figure 1.11 shows a table that violates third normal form. The PUBLISHER_PHONE_NO column is not dependent on the primary key column BOOK_ID but on the PUBLISHER_NAME column. Therefore, it should not be part of the BOOK table.

BOOK Table

BOOK_ID	TITLE	PUBLISHER _NAME	PUBLISH_ DATE	PUBLISHER _PHONE_NO
1010	The Invisible Force	Literacy Circle	12/01	801-111-1111
1011	Into The Sky	Prentice Hall	10/02	999-888-1212
1012	Making It Possible	Life Books	2/99	777-555-1212
1013	Wonders of the World	Literacy Circle	5/99	801-111-1111

Figure 1.11 ■ Violation of third normal form.

Instead, the publisher's phone number should be stored in a separate table called PUBLISHER. This has the advantage that when a publisher's phone number is updated, it only needs to be updated in one place, rather than all occurrences of this publisher in the BOOK table. Removing the PUBLISHER_PHONE_NO column eliminates redundancy and avoids any possibilities of data inconsistencies (see Figure 1.12).

Also, the BOOK table can benefit by introducing a surrogate key, such as a PUBLISHER_ID. Such a key is not subject to changes and is easily referenced in any additional tables that may need to refer to data about the publisher.

BOOK Table

BOOK_ID	TITLE	PUBLISHER_ID	PUBLISH_DATE
1010	The Invisible Force	1	12/01
1011	Into The Sky	2	10/02
1012	Making It Possible	3	2/99
1013	Wonders of the World	1	5/99

PUBLISHER Table

PUBLISHER_ID	PUBLISHER_NAME	PUBLISHER_PHONE_NO
1	Literacy Circle	801-111-1111
2	Pen Books	999-888-1212
3	Life Books	777-555-1212

Figure 1.12 ■ Tables in third normal form.

BOYCE–CODD NORMAL FORM, FOURTH NORMAL FORM, AND FIFTH NORMAL FORM

The Boyce-Codd normal form is an even more elaborate version of the third normal form and deals with deletion anomalies. The *fourth normal form* tackles potential problems when three or more columns are part of the unique identifier and their dependencies to each other. The *fifth normal form* splits the tables even further apart to eliminate all redundancy. These different normal forms are beyond the scope of this book; for more details, please consult one of the many excellent books on database design.

TABLE RELATIONSHIPS

When two tables have a common column or columns, the tables are said to have a *relationship* between them. The *cardinality* of a relationship is the actual number of occurrences for each entity. We will explore one-to-one, one-to-many, and many-to-many relationships.

ONE-TO-MANY RELATIONSHIP (1:M)

Figure 1.13 shows the CUSTOMER table and the ORDER table. The common column is CUSTOMER_ID. The link between the two tables is a *one-to-many* relationship, the most common type of relationship. This means that "one" individual customer can have "many" order rows in the ORDER table. This relationship represents the business rule that "One customer can place one or many orders (or no orders)." Reading the relationship in the other direction, an order is associated with only one customer row (or no customer rows). In other words, "each order may be placed by one and only one customer."

CUSTOMER Table

PRIMARY KEY

CUSTOMER_ID	CUSTOMER_NAME	ADDRESS	PHONE	ZIP
2010	Movers, Inc.	123 Park Lane	212-555-1212	10095
2011	Acme Mfg Ltd.	555 Broadway	212-566-1212	10004
2012	ALR Inc.	50 Fifth Avenue	212-999-1212	10010

ORDER Table

PRIMARY KEY FOREIGN KEY

ORDER_ID	CUSTOMER_ID	ORDER_DATE	TOTAL_ORDER_AMOUNT
100	2010	12/23/01	$500
101	2012	01/05/02	$600
102	2010	07/20/02	$100
103	2010	08/25/02	$400
104	2010	09/20/02	$200

Figure 1.13 ■ One-to-many relationship example between CUSTOMER and ORDER table.

ONE-TO-ONE RELATIONSHIP (1:1)

One-to-one relationships exist in the database world, but they are not typical because most often data from both tables are combined into one table for simplicity. Figure 1.14 shows an example of a *one-to-one* relationship between the PRODUCT table and the PRODUCT_PRICE table. For every row in the PRODUCT table you may find only "one" matching row in the PRODUCT_PRICE table. And for every row in the PRODUCT_PRICE table there is "one" matching row in the

PRODUCT

PRIMARY KEY

PRODUCT_ID	PRODUCT_NAME	MANUFACTURER
10001	Bolt	ACME, Inc.
10002	Screw	KR Mfg.
10003	Nail	ABC, Ltd.

PRODUCT_PRICE

FOREIGN KEY & PRIMARY KEY

PRODUCT_ID	RETAIL_PRICE	IN_STOCK_QTY
10001	$0.45	10,000
10002	$0.02	20,000
10003	$0.10	50,000

Figure 1.14 ■ One-to-one relationship example.

PRODUCT table. If the two tables are combined, the RETAIL_PRICE and IN_STOCK_QTY columns can be included in the PRODUCT table.

MANY-TO-MANY RELATIONSHIP (M:M)

The examination of Figure 1.15 reveals a *many-to-many* relationship between the BOOK and AUTHOR tables. One book can have one or more authors and one author can write one or more books. The relational database model requires the resolution of many-to-many relationships into one-to-many relationship tables. This is done by creating an *associative table* (also called an *intersection table*). The solution in this example is achieved via the BOOK_AUTHOR table. Figure 1.16 shows the columns of this table.

The BOOK_AUTHOR table lists the individual author(s) for each book and shows, for a particular author, the book(s) he or she wrote. The primary key of the BOOK_AUTHOR table is the combination of both columns: the BOOK_ID column and the AUTHOR_ID column. These two columns represent the concatenated primary key that uniquely identifies a row in the table. As you may recall from the previous lab, multicolumn primary keys are referred to as a composite or concatenated primary key. Additionally, the BOOK_AUTHOR table has the AUTHOR_ID and the BOOK_ID as two individual foreign keys linking back to the AUTHOR and the BOOK table, respectively.

The BOOK_AUTHOR table contains an additional column, the ROYALTY_SHARE column. It identifies the royalty percentage for each author for an individual book. When there are multiple authors, the percentage of the royalty is split; in the case of a sole author the share is 100 percent. This column is appropriately located in the BOOK_AUTHOR table as the values are relevant for the combination of the BOOK_ID and AUTHOR_ID. This combination of columns uniquely identifies both a book and an author and the respective percentage share of the royalty.

BOOK

Primary Key → BOOK_ID	TITLE	RETAIL_PRICE
10001	Call in the Dark	39.95
10002	The Spy	29.95
10003	Perspectives	59.95

AUTHOR

Primary Key → AUTHOR_ID	FIRST_NAME	LAST_NAME
900	King	John
901	Oats	Heather
902	Turrow	Stephen

Figure 1.15 ■ Many-to-many relationship example.

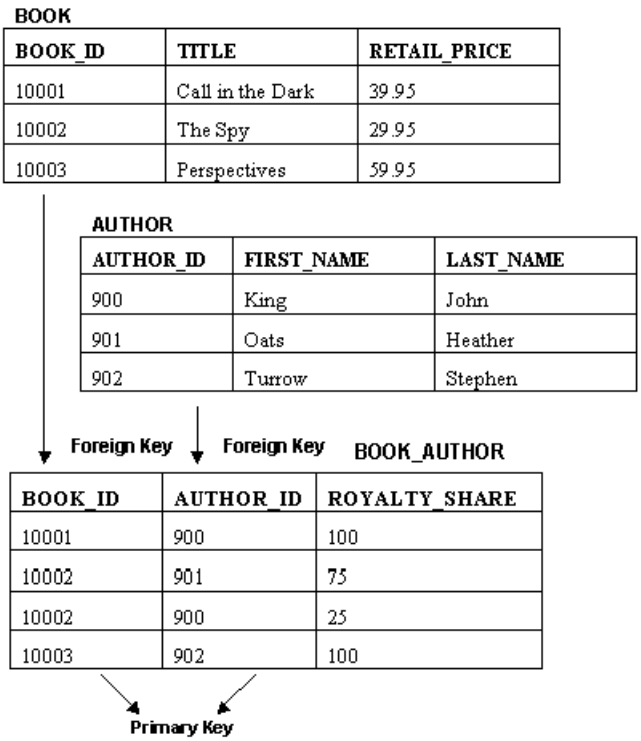


Figure 1.16 ■ Associative BOOK_AUTHOR table that resolves the many-to-many relationship.

DRAWING RELATIONSHIPS

For clarity of meaning and conceptual consistency, it is useful to show table relationships using drawings (called *schema diagrams*) and there are a number of standard notations for this type of diagram. For example, Figure 1.17 illustrates one of the ways to graphically depict the relationship between tables. The convention used in this book for a one-to-many relationship is a line with a "crow's foot" (fork) on one end indicating the "many" side of the relationship; at the other end, a "single line" depicts the "one" side of the relationship. You will see

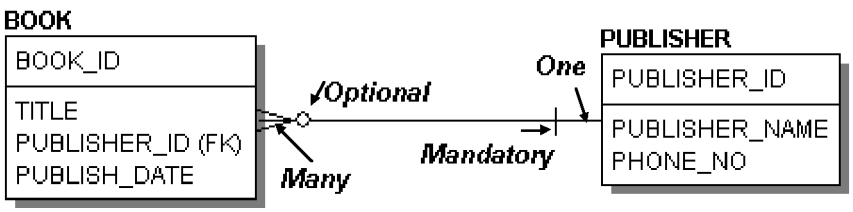


Figure 1.17 ■ Crow's foot notation.

the use of the *crow's-foot notation* throughout this book. Software diagramming programs that support the graphical display of relational database models often allow you to choose your notation preference.

CARDINALITY AND OPTIONALITY

The cardinality expresses the ratio of a parent and child table from the perspective of the parent table. It describes how many rows you may find between the two tables for a given primary key value. For example, in Figure 1.13 you saw a one-to-many relationship between the CUSTOMER and ORDER tables and the relationship ratio is expressed in the form of a 1:M ratio.

Graphical relationship lines indicate the *optionality* of a relationship, whether a row is required or not (mandatory or optional). Specifically, optionality shows if one row in a table can exist without a row in the related table.

Figure 1.17 shows a one-to-many relationship between the PUBLISHER (parent) and the BOOK (child). Examining the relationship line on the "many" end, you notice a "circle" identifying the *optional relationship* and a crow's foot indicating "many." The symbols indicate that a publisher *may* publish zero, one, or many books. You use the word "may" to indicate that the relationship is *optional* and allows a publisher to exist without a corresponding value in the BOOK table.

The relationship line also reads the other way. The solid line on the PUBLISHER end of the line indicates the "one" side, a "vertical bar" intersects it and this bar identifies a *mandatory relationship*. You read this direction of the relationship as "One book *must* be published by one and only one publisher." This means a row in the BOOK table must always have the PUBLISHER_ID value filled in. It cannot be null because that means unknown and indicates there is no associated PUBLISHER row.

The "(FK)" symbol next to the PUBLISHER_ID column indicates that this is the foreign key column. In this diagram, the primary key is separated from the other columns with a line; you observe the BOOK_ID and the PUBLISHER_ID as the primary keys or unique identifiers.

Figure 1.18 shows an optional relationship on both sides; a book may be published by zero or one publisher. Effectively, this means the value in the

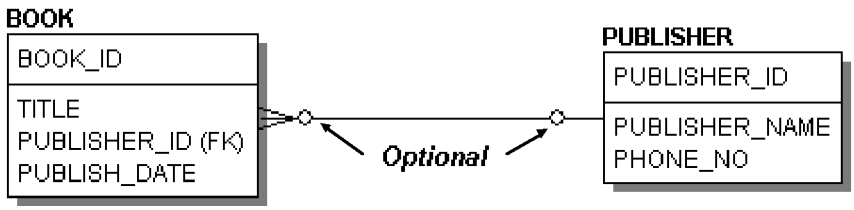


Figure 1.18 ■ Optional relationship on both sides.

PUBLISHER_ID column in BOOK is optional. Reading the relationship from the PUBLISHER, you can say that "one publisher may publish zero, one, or many books" (which is identical to Figure 1.17).

REAL WORLD BUSINESS PRACTICE

You will typically see only these two types of relationships: First, mandatory on the "one" side and optional on the "many" end as in Figure 1.17; and second, optional on both ends as in Figure 1.18. Only rarely will you find other types of relationships. For example, mandatory relationships on both sides are infrequently implemented; it means that rows must be inserted in both tables simultaneously. Occasionally you will find one-to-one relationships but most often the columns from both tables are combined into one table. Many-to-many relationships are not allowed in the relational database; they must be resolved via an associative or intersection table into one-to-many relationships.

LABELING RELATIONSHIPS

To clarify and explain the nature of the relationship on the diagram, it's useful to add a label or name with a verb on the relationship line. Figure 1.19 shows an example of a labeled relationship. For the utmost in clarity, a labeled relationship should be labeled on both sides. You then read it as: "One PUBLISHER may publish zero, one, or many BOOKs; and one BOOK must be published by one and only one PUBLISHER." This kind of labeling makes the relationship perfectly clear and states the relationship in terms that a business user can understand.

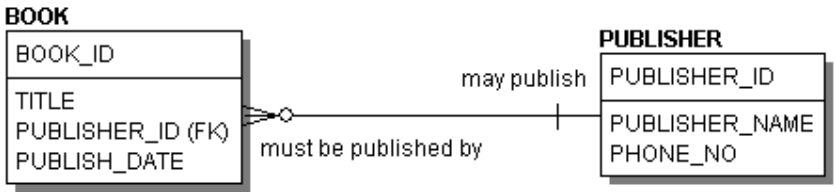


Figure 1.19 ■ Labeled relationship between BOOK and PUBLISHER.

IDENTIFYING AND NONIDENTIFYING RELATIONSHIPS

In an *identifying relationship,* the foreign key is propagated to the child entity as the primary key. This is in contrast to a *nonidentifying relationship,* in which the foreign key becomes one of the nonkey columns. Nonidentifying relationships may accept null value in the foreign key column.

Figure 1.20 depicts some of the tables used in the lab; the many-to-many relationship between the BOOK and AUTHOR tables is now resolved to the associative table called BOOK_AUTHOR. If a graphical representation of a table's box has *rounded edges* it means that the relationship is *identifying*. Effectively, one of the foreign keys became the primary key or part of the primary key. In the case of

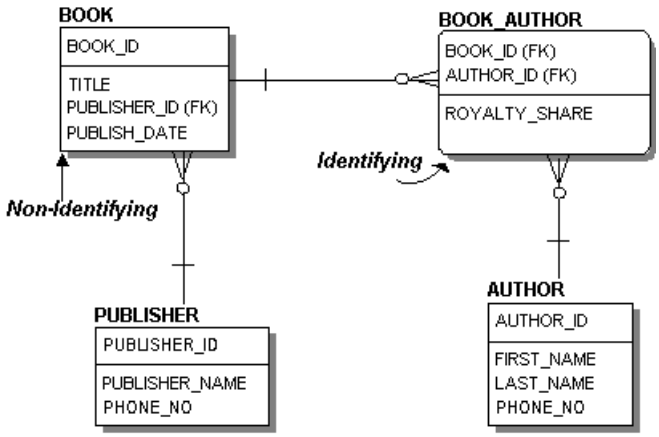


Figure 1.20 ■ Identifying and nonidentifying relationships.

the BOOK_AUTHOR table, both foreign key columns constitute the primary key and both columns may not be null because a primary key is never null.

The relationship between the PUBLISHER and BOOK tables is nonidentifying, as indicated by the sharp edges. The foreign key column PUBLISHER_ID is not part of the primary key. The foreign key columns of a nonidentifying relationship may be either NULL or NOT NULL. In this instance you can determine if a null is allowed by checking if the relationship is optional or mandatory. Although the foreign key column allows null values in non-identifying relationships, here the relationship depicts a single bar on the relationship line. Effectively, for every row in the BOOK table there must be a corresponding row in the PUBLISHER table and the PUBLISHER_ID column of the BOOK table cannot be null.

DATABASE DEVELOPMENT CONTEXT

Now that you are familiar with the some of the relational database terminology and its core concepts, you are ready to learn about how all this information fits into the context of database development. From the initial idea of an application until the final system implementation, the data model is continuously refined. Figure 1.21 indicates the essential phases of the development project with respect to the database.

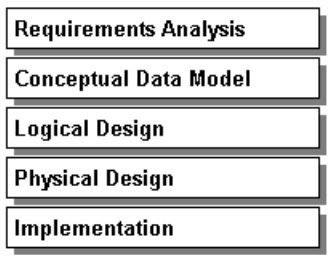


Figure 1.21 ■ Database development and design phases.

REQUIREMENTS ANALYSIS

Initially, the process starts off with gathering data requirements that identify the needs and wants of the users. One of the outputs of this phase is a list of individual data elements that need to be stored in the database.

CONCEPTUAL DATA MODEL

The *conceptual data model* logically groups the major data elements from the requirements analysis into individual *entities*. An entity is just something of significance for which you need to store data. For example, all data related to books such as the title, publish date, and retail price are placed in the book entity. Data elements such as the author's name and address are part of the author entity. The individual data elements are referred to as *attributes*.

You designate a *unique identifier* or *candidate key* that uniquely distinguishes a row in the entity. Notice that in this conceptual data model we use the terms entity, attribute, and candidate key or unique identifier instead of table, column, and primary key, respectively.

Noncritical attributes are not included in the model to emphasize the business meaning of those entities, attributes, and relationships. Many-to-many relationships are acceptable and not resolved. The diagram of the conceptual model is useful to communicate the initial understanding of the requirements to business users. The conceptual model gives no consideration to the implementation platform or database software. Many projects skip the conceptual model and go directly to the logical model.

LOGICAL DATA MODEL

The purpose of the *logical data model* is to show that all of the entities, their respective attributes, and the relationship between entities represent the business requirements without considering technical issues. The focus is entirely on business problems and considers a design that accommodates growth and change. The entities and attributes require descriptive names and documentation of their meaning. Labeling and documenting the relationships between entities clarify the business rules between them.

The diagram may show the datatype of an attribute in general terms such as text, number, and date. In many logical design models you will find foreign key columns identified, in others they are implied. (For example, Oracle's Designer software product doesn't show the foreign keys in the logical model diagram because they are an implementation detail that is implied by the relationships.)

The complete model is called the *logical data model* or *Entity Relationship Diagram* (ERD). At the end of the analysis phase the entities are fully normalized, the unique identifier for each entity is determined, and any many-to-many relationships are resolved into associative entities.

PHYSICAL DATA MODEL

The *physical data model*, also referred to as the *schema diagram,* is a graphical model of the physical design implementation of the database. This physical schema diagram is what the programmers and you will use to learn about the database and the relationship between the tables. In Lab 1.3 you will be introduced to the STUDENT schema diagram used throughout this workbook.

This physical data model is derived from the fully normalized logical model. Before the actual implementation (installation) of the physical data model in the database, multiple physical data models may exist. They represent a variety of alternative physical database designs that consider the performance implications and application constraints. One of the physical design models will be implemented in the database. The schema diagram graphically represents the chosen implemented physical data model; it is specific to a particular RDBMS product such as Oracle.

Figure 1.22 depicts the schema diagram of the book publishing database discussed in this chapter. It shows the structure of the tables with their respective columns, and it illustrates the relationships between the tables.

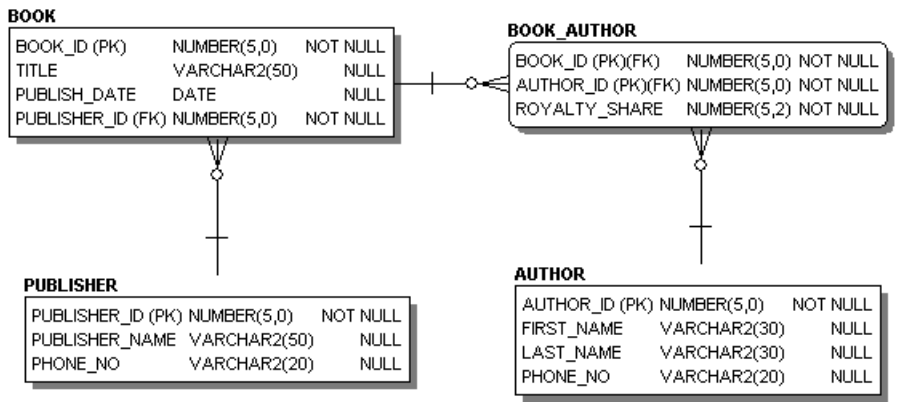


Figure 1.22 ■ Book publishing database diagram.

The physical data model has a different terminology than the conceptual or logical data model. The physical data model refers to tables instead of entities; the individual pieces of data are columns instead of attributes in the logical model.

TRANSFER FROM LOGICAL TO PHYSICAL MODEL

The transfer from the logical to the physical models, which ultimately means the actual implementation in a database as tables, columns, primary keys, foreign keys, indexes, and so on, requires a number of steps and considerations. The entities identified in the logical data model are resolved to physical tables; the entity

name is often identical to the table name. Some designers use singular names for entities and plural names for tables; others abbreviate the entity names when implementing the physical model to follow certain business naming standards. Frequently, the physical data model includes additional tables for specific technical implementation requirements and programming purposes such as a report queue table or an error log table.

As mentioned, attributes become columns with names being either identical or following business naming conventions and abbreviations. The columns are associated with the database software vendor's specific datatypes, which considers valid column lengths and restrictions. Individual data entry formats are determined (e.g., phone numbers must be in numeric format with dashes between). Rules for maintaining data integrity and consistency are created and physical storage parameters for individual tables are determined. You will learn about these and many other aspects of creating these restrictions in Chapter 11, "Create, Alter, and Drop Tables." Sometimes additional columns are added that were never in the logical design with the purpose of storing precalculated values; this is referred to as *denormalization,* which we will discuss shortly.

Another activity that occurs in the physical data design phase is the design of indexes. *Indexes* are database objects that facilitate speedy access to data to a specific column or columns of a table. Placing indexes on tables is necessary to optimize efficient query performance, but indexes have the negative impact of requiring additional time for insert, update, or delete operations. Balancing the trade-offs with the advantages requires careful consideration of these factors, including knowledge in optimizing SQL statements and an understanding of the features of a particular database version. You will learn more about different types of indexes and the success factors of a well-placed index strategy in Chapter 12, "Views, Indexes, and Sequences."

Poor physical database design is very costly and difficult to correct.

Database designers must be knowledgeable and experienced in many aspects of programming, design, and database administration to fully understand how design decisions impact cost, system interfaces, programming effort, and future maintenance.

You may wonder how the graphical models you see in this book are produced. Specific software packages allow you to visually design the various models and they allow you to display different aspects of it such as showing only table names or showing table names, columns, and their respective datatypes. Many of these tools even allow you to generate the DDL SQL statements to create the tables. For a list of software tools that allow you to visually produce the diagrams, see the book's Web site at http://www.phptr.com/rischert and Appendix H,"Resources."

DENORMALIZATION

Denormalization is the act of adding redundancy to the physical database design. Typically, logical models are fully normalized or at least in third normal form. When designing the physical model, database designers must weigh the benefit of eliminating all redundancy with data split into many tables against potentially poor performance when these many tables are joined.

Therefore database designers, also called database architects, sometimes purposely add redundancy to their physical design. Only experienced database designers should do denormalization. Increasing redundancy may greatly increase the overall programming effort because now many copies of the same data must be kept in sync; however, the time it takes to query data may be less.

In some applications, particularly data warehousing applications where massive amounts of detailed data are stored and summarized, denormalization is required. *Data warehouse applications* are database applications that benefit users that need to analyze large data sets from various angles and use this data for reporting and decision-making purposes. Typically, the source of the data warehouse is historical transaction data but can also include data from various other sources for the purpose of consolidating data. For example, the purchasing department of a supermarket chain could determine how many turkeys to order for a specific store on the week before Thanksgiving or use the data to determine what promotional offers have the largest sales impact on stores with certain customer demographics.

The primary purpose of a data warehouses is to query, report, and analyze data. Therefore redundancy is encouraged and necessary for queries to perform efficiently.

LAB 1.2 EXERCISES

1.2.1 READ A SCHEMA DIAGRAM

a) Describe the nature of the relationship between the ORDER_HEADER table and the ORDER_DETAIL table (Figure 1.23).

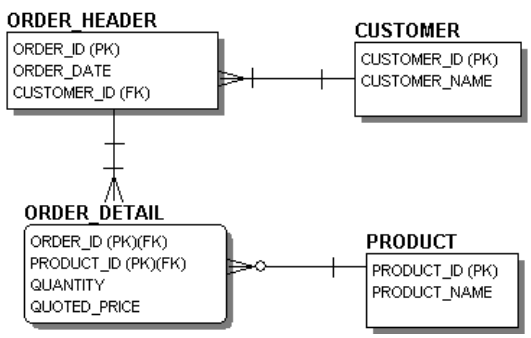


Figure 1.23 ■ Order tables.

1.2.2 Identify Data Normalization Rules and Table Relationships

a) One of the tables in Figure 1.24 is not fully normalized. Which normal form is violated? Draw a new diagram.

b) How would you change Figure 1.24 to add information about the sales representative that took the order?

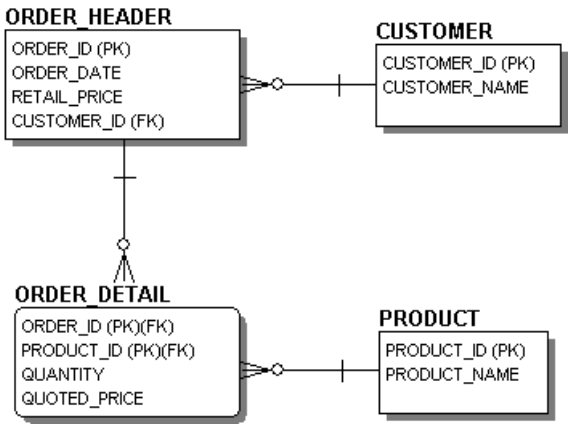


Figure 1.24 ■ Not fully normalized table.

c) How would you change Figure 1.25 if an employee does not need to belong to a department?

d) Based on Figure 1.25, why do you think the social security number (SSN) column should not be the primary key of the EMPLOYEE table?

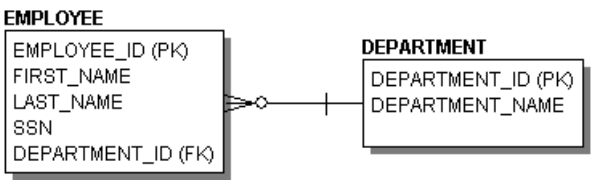


Figure 1.25 ■ EMPLOYEE to DEPARTMENT relationship.

1.2.3 Understand the Database Development context

a) Figures 1.26 and 1.27 depict the logical and physical model of a fictional movie rental database. What differences do you notice between the following entity relationship diagram and the physical schema diagram?

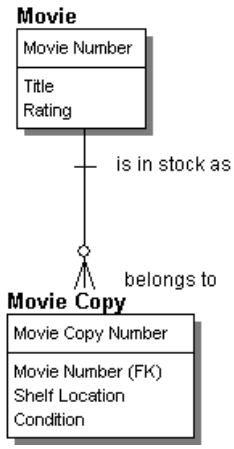


Figure 1.26 ■ Logical Data Model.

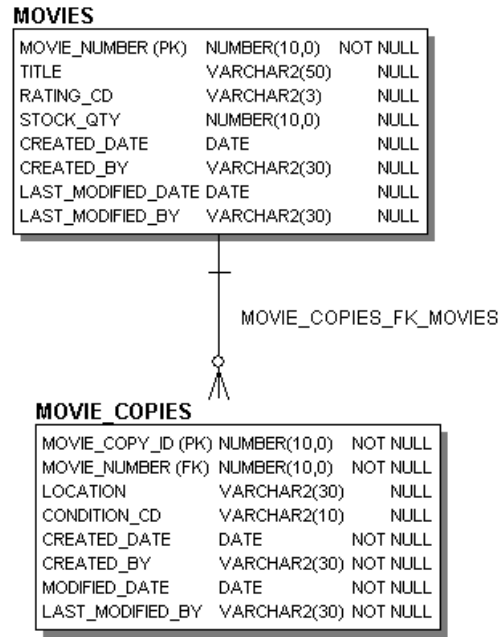


Figure 1.27 ■ Physical Data Model.

LAB 1.2 EXERCISE ANSWERS

1.2.1 Answers

a) Describe the nature of the relationship between the ORDER_HEADER table and the ORDER_DETAIL table (Figure 1.23).

Answer: The relationship depicts a mandatory one-to-many relationship between the ORDER_HEADER and the ORDER_DETAIL tables. The ORDER_HEADER table contains data found only once for each order, such as the ORDER_ID, the CUSTOMER_ID, and the ORDER_DATE. The ORDER_DETAIL table holds information about the individual order lines of an order. One row in the ORDER_HEADER table must have one or many order details. One ORDER_DETAIL row must have one and only one corresponding row in the ORDER_HEADER table.

MANDATORY RELATIONSHIP ON BOTH ENDS

The mandatory relationship indicates from the ORDER_HEADER to ORDER_DETAIL that a row in the ORDER_HEADER table cannot exist unless a row in ORDER_DETAIL is created simultaneously. This is a "chicken and egg" problem whereby a row in the ORDER_HEADER table cannot be created without an ORDER_DETAIL row and vice versa. In fact, it really doesn't matter as long as you create the rows within one transaction. Furthermore, you must make sure that every row in the ORDER_HEADER table has at least one row in the ORDER_DETAIL table and rows in the ORDER_DETAIL table have exactly one corresponding row in the ORDER_HEADER table. There are various ways to physically implement this relationship.

Another example of a mandatory relationship on the figure is the relationship between ORDER_HEADER and CUSTOMER. You can see the bar on the many side of the relationship as an indication for the mandatory row. That means a customer must have placed an order before a row in the CUSTOMER table is saved, and an order can only be placed by a customer.

However, for most practical purposes a mandatory relationship on both ends is rarely implemented unless there is a very specific and important requirement.

NO DUPLICATES ALLOWED

On the previous diagrams, such as Figure 1.23, you noticed that some foreign keys are part of the primary key. This is frequently the case in associative entities; in this particular example it requires the combination of ORDER_ID and PRODUCT_ID to be unique. Ultimately the effect is that a single order containing the same product twice is not allowed. Figure 1.28 lists sample data in the ORDER_DETAIL table for ORDER_ID 345 and PRODUCT_ID P90, which violates the primary key and is therefore not allowed. Instead, you must create one order with a quantity of 10 or create a second order with a different ORDER_ID so the primary key is not violated. You will learn about how Oracle responds with error messages when you attempt to violate the primary key constraint and other types of constraints in Chapter 10, "Insert, Update, and Delete."

ORDER_DETAIL Table

ORDER_ID	PRODUCT_ID	QUANTITY	QUOTED_PRICE
123	P90	5	$50
234	S999	9	$12
345	P90	7	$50
345	X85	3	$10
345	P90	3	$50

Figure 1.28 ■ Sample data of the ORDER_DETAIL table.

1.2.2 Answers

a) One of the tables in Figure 1.24 is not fully normalized. Which normal form is violated? Draw a new diagram.

Answer: The third normal form is violated on the ORDER_HEADER table. The RETAIL_PRICE column belongs to the PRODUCT table instead (Figure 1.29).

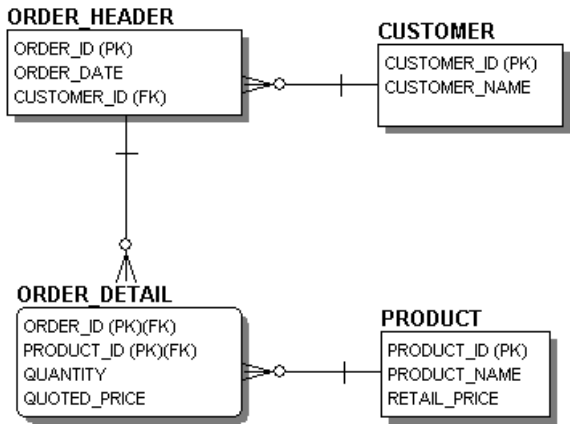


Figure 1.29 ■ Fully normalized tables.

Third normal form states that every nonkey column must be a fact about the primary key column, which is the ORDER_ID column in the ORDER_HEADER table. This is clearly not the case in the ORDER_HEADER table, as the RETAIL_PRICE column is not a fact about the ORDER_HEADER and does not depend upon the ORDER_ID; it is a fact about the PRODUCT. The QUOTED_PRICE column is included in the ORDER_DETAIL table because the price may vary over time, from order to order, and from customer to customer. (If you want to track any changes in the retail price, you may want to create a separate table called PRODUCT_PRICE_HISTORY that keeps track of the retail price per product and the effective date of each price change.) Table 1.2 provides a review of the normal forms.

Table 1.2 ■ The Three Normal Forms

Description	Rule
First Normal Form (1NF)	No repeating groups are permitted
Second Normal Form (2NF)	No partial key dependencies are permitted
Third Normal Form (3NF)	No nonkey dependencies are permitted.

b) How would you change Figure 1.24 to add information about the sales representative that took the order?

Answer: As you see in Figure 1.30, you need to add another table that contains the sales representative's name, SALES_REP_ID, and any other important information. The SALESREP_ID then becomes a foreign key in the ORDER_HEADER table.

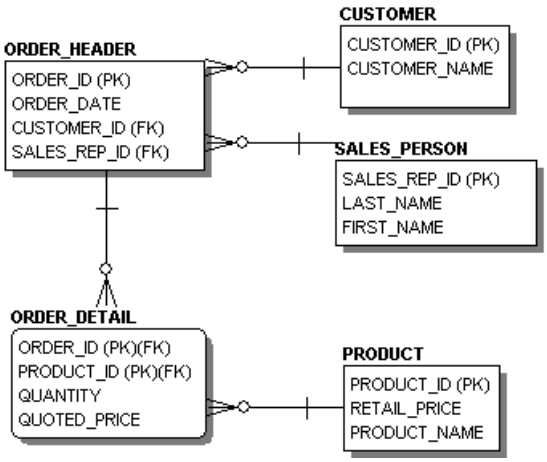


Figure 1.30 ■ ORDER_HEADER with SALES_REP_ID column.

c) How would you change Figure 1.25 if an employee does not need to belong to a department?

Answer: You change the relationship line on the DEPARTMENT table end to make it optional. This has the effect that the DEPARTMENT_ID column on the EMPLOYEE table can be null; that is, a value is not required (Figure 1.31).

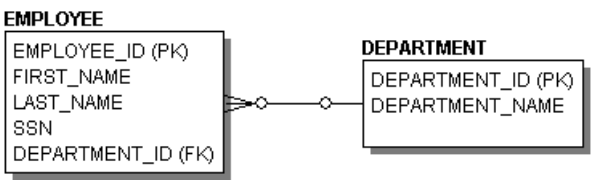


Figure 1.31 ■ EMPLOYEE to DEPARTMENT with optional relationship line.

d) Based on Figure 1.25, why do you think the social security number (SSN) column should not be the primary key of the EMPLOYEE table?

Answer: The requirement for a primary key is that it is unique, not subject to updates, and not null.

Although the SSN is unique, there have been incidents (though rare) of individuals with the same SSN or individuals who had to change their SSN. It is conceiv-

able to have an employee without a SSN assigned yet (e.g., a legal alien with a work permit), hence the column is null. There is a myriad of reasons for not using a SSN, therefore it's best to create a surrogate or artificial key.

1.2.3 Answers

a) Figures 1.26 and 1.27 depict the logical and physical model of a fictional movie rental database. What differences do you notice between the following entity relationship diagram and the physical schema diagram?

Answer: You can spot a number of differences between the logical model (entity relational diagram) and the physical model (schema diagram). While some logical and physical models are identical, these figures exhibit distinguishing differences you may find in the real world.

The entity name of the logical model is singular versus plural for the table name on the physical model. Some table names have special prefixes that denote the type of application the table belongs to. For example, if a table belongs to the purchase order system, it may be prefixed with PO_; if it belongs to the accounts payable system, the prefix is AP_; and so on. In the logical model, the spaces are allowed for table and column names. Typically, in Oracle implementations, table names are defined in uppercase and use the underscore (_) character to separate words.

Although the logical model may include the datatypes, here the datatype (such as DATE, VARCHAR2, NUMBER) shows on the physical model only. The physical model also indicates if a column allows NULL values.

The attribute and column names differ between the two models. For example, the RATING attribute changed to RATING_CD, which indicates the values are encoded such as for example "PG" rather than a descriptive "Parental Guidance" value. Designers create or follow established naming conventions and abbreviations for consistency. Naming conventions can help describe the type of values stored in the column.

The STOCK_QTY is another example of using the abbreviation QTY to express that the column holds a quantity of copies. Notice this column is absent from the logical model; it is a *derived column*. The quantity of movies for an individual movie title could be determined from the MOVIE_COPIES table. The database designer deliberately denormalized the table by adding this column. This simplifies any queries that determine how many copies of this particular title exist. Rather than issuing another query that counts the number of rows in the MOVIE_COPIES for the specific title, this column can be queried. Adding a derived column to a table requires that the value stay in sync with the data in the related table (MOVIE_COPIES in this case). The synchronization can be accomplished by writing a program that is executed from the end-user's screen. Alternatively, the developer could write a PL/SQL trigger on the table that automatically updates the STOCK_QTY value whenever a new row is added or deleted on the MOVIE_

COPIES table for each individual title. (For an example of a table trigger, refer to Chapter 12, "Create, Alter, and Drop Tables.")

The schema diagram prominently exhibits columns that did not exist in the logical data model, namely CREATED_DATE, MODIFIED_DATE, CREATED_BY, and MODIFIED_BY. Collectively these columns are sometimes referred to as "audit columns." They keep information about when a row was created and last changed together with the respective user that executed this action.

On the logical data model the relationship is labeled in both directions. On the physical model, the name of the foreign key constraint between the tables is listed instead. You may find that some physical models depict no label at all. There are no set standards for how a physical or logical model must graphically look and therefore the diagrams produced by various software vendors that offer diagramming tools not only look different, they also allow a number of different display options.

LAB 1.2 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) An entity relationship diagram depicts entities, attributes, and tables.
- **a)** _____ True
- **b)** _____ False

2) The crow's foot depicts the M of a 1:M relationship.
- **a)** _____ True
- **b)** _____ False

3) Repeating groups are a violation of the first normal form.
- **a)** _____ True
- **b)** _____ False

4) The logical model is derived from the schema diagram.
- **a)** _____ True
- **b)** _____ False

5) The concept of denormalization deals with eliminating redundancy.
- **a)** _____ True
- **b)** _____ False

6) When you issue a SQL statement, you are concerned with the logical design of the database.
- **a)** _____ True
- **b)** _____ False

7) In a mandatory relationship, null values are not allowed in the foreign key column.
- **a)** _____ True
- **b)** _____ False

8) A nonidentifying relationship means that the foreign key is propagated as a nonkey attribute in the child entity or child table.
- **a)** _____ True
- **b)** _____ False

Answers appear in Appendix A, Section 1.2.

LAB 1.3

THE STUDENT SCHEMA DIAGRAM

LAB OBJECTIVES

After this lab, you will be able to:

- ✔ Understand the STUDENT Schema Diagram and Identify Table Relationships

Throughout this series of Oracle Interactive Workbooks, the database for a school's computer education program is used as a case study upon which all exercises are based. If you have worked through the previous two labs, you know that the schema diagram is a model of data that reflects the relationships among data in a database. The name of the case study schema diagram is STUDENT. Before you begin to write SQL statements against the database, it is important to familiarize yourself with the diagram. You can find this graphical representation in Appendix D, "Student Database Schema."

In this book you will be frequently referring to the STUDENT schema diagram shown in Appendix D, "STUDENT Database Schema." Rather than flipping back and forth, you may find it more convenient to print out the schema diagram from the companion Web site of this book located at http://www.phptr.com/rischert.

THE STUDENT TABLE

Examine the STUDENT schema diagram and locate the STUDENT table. This table contains data about individual students, such as their name, address, employer, and date they registered in the program.

DATATYPES

Next to each column name in the diagram you find the datatype of the column. Each column contains a different kind of data, which can be classified by a datatype. You will notice that the FIRST_NAME column is of datatype VARCHAR2(25). This means that a variable length of (with a maximum of 25) alphanumeric characters (letters or numbers) may be stored in this column. Another datatype, the CHAR datatype, also stores alphanumeric data, but is a fixed-length datatype and pads any unused space in the column with blanks until it reaches the defined column length. The STUDENT_ID column is of datatype NUMBER with a maximum number of eight integer digits and no decimal place digits; the column is the primary key as denoted with the "(PK)" symbol. Oracle also provides a DATE datatype (as seen on the CREATED_DATE and MODIFIED_DATE columns) that stores both the date and time. You will learn more about the various datatypes in the next chapter.

Next to each column, the schema diagram indicates if a column allows NULL values. A NULL value is an unknown value. A space or value of zero is not the same as NULL. When a column in a row is defined as allowing NULL values, it means that a column does not need to contain a value. When a column is defined as NOT NULL it must always contain a value.

You will observe that the STUDENT table does not show the city and state. This information can be looked up via the foreign key column ZIP as indicated with the "(FK)" symbol after the column name. The ZIP column is a NOT NULL column and requires that every student row have a corresponding zip code entered.

THE COURSE TABLE

The COURSE table lists all the available courses that a student may take. The primary key of the table is the COURSE_NO column. The DESCRIPTION column shows the course description and the COST column lists the dollar amount charged for the enrollment in the course. The PREREQUISITE column displays the course number, which must be taken as a prerequisite to this course. This column is a foreign key column and its values refer back the COURSE_NO column. Only valid COURSE_NO values may be listed in this column. The relationship line of the COURSE table to itself represents a *recursive* or *self-referencing relationship*.

RECURSIVE RELATIONSHIP

As the term recursive or self-referencing relationship implies, a column in the COURSE table refers back to another column in the same table. The PREREQUISITE column refers back to the COURSE_NO column, which provides the list of acceptable values (also referred to as a *domain*) for the PREREQUISITE column. Because the relationship is optional, the foreign key column PREREQUISITE column allows null. Recursive relationships are always optional relationships; otherwise, there is no starting point in the hierarchy.

COURSE_NO	DESCRIPTION	PREREQUISITE	...
10	DP Overview		...
20	Intro to Computers		...
100	Hands-On Windows	20	...
140	Structured Analysis	20	...
25	Intro to Programming	140	...
...	...	...	...

Figure 1.32 ■ Data from the COURSE table.

Figure 1.32 lists an excerpt of data from the COURSE table. Notice that the courses with the COURSE_NO column values of 10 and 20 do not have a value in the PREREQUISITE column, those are the courses which a student must take to be able to take any subsequent courses (unless equivalent experience can be substituted). Course number 20 is a prerequisite course for course number 100, Hands-On-Windows, and course number 140, Structured Analysis. You will explore more about the intricacies of recursive relationships in Chapter 15, "Advanced SQL Queries."

THE SECTION TABLE

The SECTION TABLE includes all the individual sections a course may have. An individual course may have zero, one, or many sections, each of which can be taught at different rooms, times, and by different instructors. The primary key of the table is the SECTION_ID. The foreign key that links back to the COURSE table is the COURSE_NO column. The SECTION_NO column identifies the individual section number. For example, for the first section of a course, it contains the number 1; the second section lists the number 2, and so on. The two columns, COURSE_NO and SECTION_NO, also uniquely identify a row, but SECTION_ID has been created instead. This SECTION_ID column is called a surrogate key because it does not have any meaning to the user.

The column START_DATE_TIME shows the date and time the section meets for the first time. The LOCATION column lists the classroom. The CAPACITY column shows the maximum number of students that may enroll in this section. The INSTRUCTOR_ID column is another foreign key column within the SECTION table; it links back to the INSTRUCTOR table. The relationship between the SECTION and the INSTRUCTOR table indicates that an instructor must always be assigned to a section. The INSTRUCTOR_ID column of the SECTION table may never be null and when you read the relationship from the opposite end, you can say that an individual instructor may teach zero, one, or multiple sections.

The relationship line leading from the COURSE table to the SECTION table means that a course may have zero, one, or multiple sections. Conversely, every individual section *must* have a corresponding row in the COURSE table.

Relationships between tables are based on *business rules*. In this case, the business rule is that a course can exist without a section, but a section cannot exist unless it is assigned to a course. As mentioned, this is indicated with the bar (|) on the other end of the relationship line. Most of the child relationships on the schema diagram are considered mandatory relationships (with two exceptions); this dictates that the foreign key columns in the child table must contain a value (must be NOT NULL) and that value must correspond to a row in the parent table via its primary key value.

THE INSTRUCTOR TABLE

The INSTRUCTOR table lists information related to an individual instructor, such as name, address, phone, and zip code. The ZIP column is the foreign key column to the ZIPCODE table. The relationship between the INSTRUCTOR and the ZIPCODE is an optional relationship so a null value in the ZIP column is allowed. For a given ZIP column value there is one and only one value in the ZIPCODE table. For a given ZIP value in the ZIPCODE table you may find zero, one, or many of the same value in the INSTRUCTOR table. Another foreign key relationship exists to the SECTION table: an instructor may teach zero, one, or multiple sections and an individual section can be taught by one and only one instructor.

THE ZIPCODE TABLE

The primary key of ZIPCODE is the ZIP column. For an individual zip code it allows you to look up the corresponding CITY and STATE column values. The datatype of this column is VARCHAR2 and not a NUMBER, as it allows you to enter leading zeros. Both the STUDENT and the INSTRUCTOR table reference the ZIPCODE table. The relationship between the ZIPCODE and STUDENT tables is mandatory: For every ZIP value in the STUDENT table there must be a corresponding value in the ZIPCODE table, and for one given zip code, there may be zero, one, or multiple students with that zip code. In contrast, the relationship between the INSTRUCTOR and ZIPCODE table is optional; the ZIP column of the INSTRUCTOR table may be null.

WHAT ABOUT DELETE OPERATIONS?

Referential integrity does not allow deletion in a parent table of a primary key value that exists in a child table as a foreign key value. This would create orphan rows in the child table. There are many ways to handle deletes and you will learn about this topic and the effects of the deletes on other tables in Chapter 10, "Insert, Update, and Delete."

THE ENROLLMENT TABLE

The ENROLLMENT table is an intersection table between the STUDENT and the SECTION table. It lists the students enrolled in the various sections. The primary key of the table is a composite primary key consisting of the STUDENT_ID and

SECTION_ID columns. This unique combination does not allow a student to register for the same section twice. The ENROLL_DATE column contains the date the student registered for the section and the FINAL_GRADE column lists the student's final grade. The final grade is to be computed from individual grades such as quizzes, homework assignments, and so on.

The relationship line between the ENROLLMENT and STUDENT tables indicates that one student may be enrolled in zero, one, or many sections. For one row of the ENROLLMENT table you can find one and only one corresponding row in the STUDENT table. The relationship between the ENROLLMENT and SECTION table shows that a section may have zero, one, or multiple enrollments. A single row in the ENROLLMENT table always links back to one and only one row in the SECTION table.

THE GRADE_TYPE TABLE

The GRADE_TYPE table is a lookup table for other tables as it relates to grade information. The table's primary key is the GRADE_TYPE_CODE column that lists the unique category of grade, such as MT, HW, PA, and so on. The DESCRIPTION column describes the abbreviated code. For example, for the GRADE_TYPE_CODE of MT you will find the description Midterm, for HW you see Homework.

THE GRADE TABLE

This table lists the grades a student received for an individual section. The primary key columns are STUDENT_ID, SECTION_ID, GRADE_TYPE_CODE, and GRADE_CODE_OCCURRENCE. For an individual student you will find the all the grades related to the section the student is enrolled in. For example, the listed grades in the table may include the midterm grade, individual quizzes, final examination grade, and so on. For some grades (e.g., quizzes, homework assign-

STUDENT_ID	SECTION_ID	GRADE_TYPE_CODE	GRADE_CODE_OCCURRENCE	NUMERIC_GRADE	...
221	104	FI	1	77	...
221	104	HM	1	76	...
221	104	HM	2	76	...
221	104	HM	3	86	...
221	104	HM	4	96	...
221	104	MT	1	90	...
221	104	PA	1	83	...
221	104	QZ	1	84	...
221	104	QZ	2	83	...
...	...	...	...	...	...

Figure 1.33 ■ Data from the GRADE table.

ments) there may be multiple grades and the sequence number is shown in the GRADE_CODE_OCCURRENCE column. Figure 1.33 displays an excerpt of data from the GRADE table. The NUMERIC_GRADE column lists the actual grade received. This grade may be converted to a letter grade with the help of the GRADE_CONVERSION table discussed later.

From the relationship between the ENROLLMENT and GRADE table, you can learn that rows only exist in the GRADE table if the student is actually enrolled in the section listed in the ENROLLMENT table. In other words, it is not possible for a student to have grades for a section in which he or she is not enrolled. The foreign key columns STUDENT_ID and SECTON_ID from the ENROLLMENT table enforce this relationship.

THE GRADE_TYPE_WEIGHT TABLE

The GRADE_TYPE_WEIGHT table aids in computation of the final grade a student receives for an individual section. This table lists how the final grade for an individual section is computed. For example, the midterm may constitute 50 percent of the final grade, all the quizzes 10 percent, and the final examination 40 percent. If there are multiple grades for a given GRADE_TYPE_CODE, the lowest grade may be dropped if the column DROP_LOWEST contains the value "Y". The final grade is determined by using the individual grades of the student and section in the GRADE table in conjunction with this table. This computed final grade value is stored in the FINAL_GRADE column of the ENROLLMENT table discussed previously. (The FINAL_GRADE column is a derived column. As mentioned, the values to compute this number are available in the GRADE and GRADE_TYPE_WEIGHT tables, but because the computation of this value is complex, it is stored to simplify queries.)

The primary key of this table consists of the SECTION_ID and GRADE_TYPE_CODE columns. A particular GRADE_TYPE_CD value may exist zero, one, or multiple times in the GRADE_TYPE_WEIGHT table. For every row of the GRADE_TYPE_WEIGHT table you will find one and only one corresponding GRADE_TYPE_CODE value in the GRADE_TYPE table.

The relationship between the GRADE_TYPE_WEIGHT table and the SECTION table indicates that a section may have zero, one, or multiple rows in the GRADE_TYPE_WEIGHT table for a given SECTION_ID value. For one SECTION_ID value in the GRADE_TYPE_WEIGHT table there must always be one and only one corresponding value in the SECTION table.

THE GRADE_CONVERSION TABLE

The purpose of the GRADE_CONVERSION table is to convert a number grade to a letter grade. The table does not have any relationship with any other tables. The column LETTER_GRADE contains the unique grades, such as A+, A, A–, B, and so forth. For each of these letter grades, there is an equivalent number range. For example, for the letter B, the range is 83 through 86 and is listed in the MIN_GRADE and MAX_GRADE columns.

You can find the individual table and column descriptions of all the tables listed in the STUDENT schema diagram in Appendix E, "Table and Column Descriptions."

LAB 1.3 EXERCISES

1.3.1 UNDERSTAND THE SCHEMA DIAGRAM AND IDENTIFY TABLE RELATIONSHIPS

a) What does the STUDENT schema diagram represent?

b) Does the STUDENT schema diagram tell you where a student lives? Explain.

c) What four columns are common to all tables in the STUDENT schema diagram?

d) What is the primary key of the COURSE table?

e) How many primary keys does the ENROLLMENT table have? Name the column(s).

f) How many foreign keys does the SECTION table have?

g) Will a foreign key column in a table accept any data value? Explain using the STUDENT and ZIPCODE tables.

h) If the relationship between the ZIPCODE and STUDENT tables were optional, what would have to change in the STUDENT table?

i) From what domain of values (what column in what table) do you think the PREREQUISITE column of the COURSE table gets its values?

j) Explain the relationship(s) the ENROLLMENT table has to other table(s).

LAB 1.3

LAB 1.3 EXERCISE ANSWERS

1.3.1 ANSWERS

a) What does the STUDENT schema diagram represent?

Answer: The STUDENT schema diagram is a graphical representation of tables in a relational database.

A schema diagram is a useful tool during the software development lifecycle. English-like words should be used to name tables and columns so that anyone, whether developer or end-user, can look at a schema diagram and grasp the meaning of data, and the relationships among them, represented there. Developers study it to understand the design of a database, long before they put hands to keyboard to develop a system, and end-users can use it to understand how their data is stored.

b) Does the STUDENT schema diagram tell you where a student lives? Explain.

Answer: No. The STUDENT schema diagram tells you how data is organized in a relational database: the names of tables, the columns in those tables, and the relationship among them. It cannot tell you what actual data looks like. You use the SQL language to interact with a relational database to view, manipulate, and store the data in the tables.

c) What four columns are common to all tables in the STUDENT schema diagram?

Answer: The four columns are CREATED_BY, CREATED_DATE, MODIFIED_BY, and MODIFIED_DATE.

Database tables are often created with columns similar to these four to create an audit trail. These columns are designed to identify who first created or last modified a row of a table and when the action occurred. You will typically find these columns only on the physical schema diagram, not on the logical model. Some of these values in the columns can be filled in automatically by writing triggers. You will see an example of a table trigger in Chapter 12, "Create, Alter, and Drop

Tables." (Triggers are described in further detail in the *Oracle PL/SQL Interactive Workbook* by Benjamin Rosenzweig and Elena Silvestrova; Prentice Hall, 2003.)

d) What is the primary key of the COURSE table?

Answer: The primary key of the COURSE table is the column COURSE_NO.

You can identify the primary key with the "PK" symbol listed next to the column. In general a primary key uniquely identifies a row in a table, and the column or columns of the primary key are defined as NOT NULL.

e) How many primary keys does the ENROLLMENT table have? Name the column(s).

Answer: A table can have only one primary key. The primary key of the ENROLLMENT table consists of the two columns STUDENT_ID and SECTION_ID.

As mentioned earlier, a primary key uniquely identifies a single row in a table. In the case of the ENROLLMENT table, two columns uniquely identify a row and create a composite primary key.

Looking at the schema diagram you also notice that these two columns are also foreign keys. The STUDENT_ID column is the foreign key to the STUDENT table and the SECTION_ID is the foreign key to the SECTION table. Both foreign key relationships are identifying relationships.

f) How many foreign keys does the SECTION table have?

Answer: Two. The foreign keys of the SECTION table are COURSE_NO and INSTRUCTOR_ID.

g) Will a foreign key column in a table accept any data value? Explain using the STUDENT and ZIPCODE tables.

Answer: No. A foreign key must use the values of the primary key it references as its domain of values.

The ZIP column is the primary key in the ZIPCODE table. The STUDENT table references this column with the foreign key ZIP column. Only values that exist in the ZIP column of the ZIPCODE table can be valid values for the ZIP column of the STUDENT table. If you attempt to create a row or change an existing row in the STUDENT table with a zip code not found in the ZIPCODE table, the foreign key constraint on the STUDENT table will reject it.

In general, a foreign key is defined as being a column, or columns, in the child table. This column refers back to the primary key of another table, referred to as the parent table.

The primary key values are the domain of values for the foreign key column. A *domain* is a set of values that shows the possible values a column can have. The primary key values of the parent table are the only acceptable values that may appear in the foreign key column in the other table. (Domains are not only used in context with primary key and foreign key relationships, but can also be used for a list of values that may not be stored in a table. For example, common domains include Yes/No, Gender: Male/Female/Unknown, Weekday: Sun/Mon/Tue/Wed/Thu/Fri/Sat.)

h) If the relationship between the ZIPCODE and STUDENT tables were optional, what would have to change in the STUDENT table?

Answer: The foreign key column ZIP in the STUDENT table would have to be defined as allowing NULL values. It is currently defined as NOT NULL. The relationship should be indicated as optional instead of mandatory as shown in Figure 1.34.

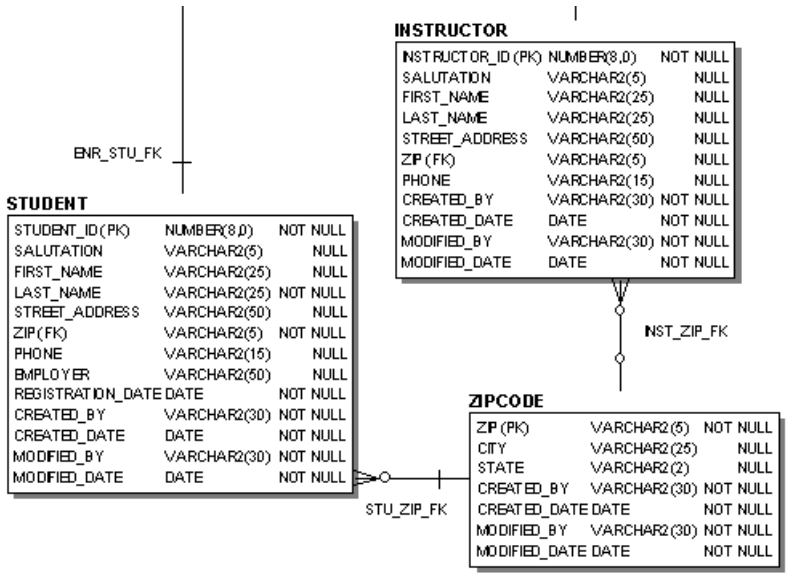


Figure 1.34 ■ The relationships of the ZIPCODE table.

There is such an optional relationship between the INSTRUCTOR and ZIPCODE tables. All the nonnull values of ZIP in the INSTRUCTOR table must be found in the ZIPCODE table.

i) From what domain of values (what column in what table) do you think the PREREQUISITE column of the COURSE table gets its values?

Answer: From the COURSE_NO column in the COURSE table.

In this case, the PREREQUISITE column refers back to the COURSE_NO column, which provides the domain of values for the PREREQUISITE column. A prerequisite is valid only if it is also a valid course number in the COURSE table. This relationship is shown in Figure 1.35.

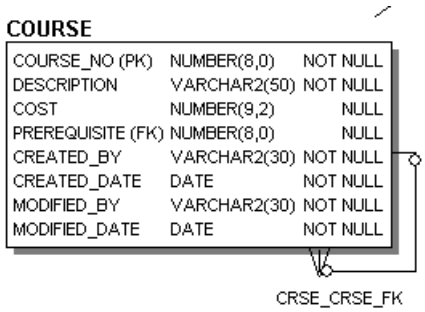


Figure 1.35 ■ The self-referencing relationship of the COURSE table.

j) Explain the relationship(s) the ENROLLMENT table has to other table(s).

Answer: The STUDENT table and the SECTION table are the parent tables of the ENROLLMENT table. The ENROLLMENT table is one of the parent tables of the GRADE table.

As shown in Figure 1.36, the relationship between the STUDENT and SECTION tables signifies a student may be enrolled in zero, one, or many sections. One individual student can be enrolled in one specific section only once, otherwise the unique combination of the two columns in the ENROLLMENT table would be violated. The combination of these two foreign key columns represents the primary key of the ENROLLMENT table.

The relationship of the ENROLLMENT table as the parent of the GRADE table shows that for an individual student and her or his enrolled section there may be zero, one, or many grades. The primary key columns of the ENROLLMENT table (STUDENT_ID and SECTION_ID) are foreign keys in the GRADE table that become part of the GRADE table's composite primary key. Therefore, only enrolled students may have rows in the GRADE as indicated with the optional line. If a row in GRADE exists, it must be for one specific enrollment in a section for one specific student.

Note: In some cases, the foreign keys become part of a table's primary key, as in the ENROLLMENT or the GRADE table. If a composite primary key contains many columns (perhaps more than four or five), a surrogate key may be considered for simplicity. The decision to use a surrogate key is based on the database designer's understanding of how data is typically accessed by the application programs.

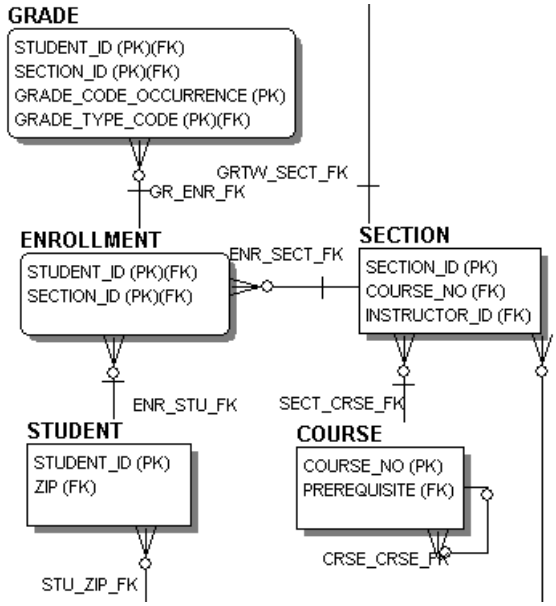


Figure 1.36 ■ The relationships of the ENROLLMENT table.

LAB 1.3 SELF-REVIEW QUESTIONS

In order to test your progress, you should be able to answer the following questions.

1) What role(s) does the STUDENT_ID column play in the GRADE table? Check all that apply.
- **a)** _____ Part of composite primary key
- **b)** _____ Primary key
- **c)** _____ Foreign key

2) The GRADE_TYPE table does not allow values to be NULL in any column.
- **a)** _____ True
- **b)** _____ False

3) The number of columns in a table matches the number of rows in that table.
- **a)** _____ True
- **b)** _____ False

4) The SECTION table has no foreign key columns.
- **a)** _____ True
- **b)** _____ False

5) A table can contain 10 million rows.
- **a)** _____ True
- **b)** _____ False

6) A primary key may contain NULL values.
- **a)** _____ True
- **b)** _____ False

7) A column name must be unique within a table.
- **a)** _____ True
- **b)** _____ False

8) If a table is a child table in three different one-to-many relationships, how many foreign key columns does it have?
- **a)** _____ One
- **b)** _____ Exactly three
- **c)** _____ Three or more

9) Referential integrity requires the relationship between foreign key and primary key to maintain values from the same domain.
- **a)** _____ True
- **b)** _____ False

10) A foreign key may be NULL.
- **a)** _____ True
- **b)** _____ False

11) Orphan rows are not allowed in the relational model.
- **a)** _____ True
- **b)** _____ False

Answers appear in Appendix A, Section 1.3.

CHAPTER 1

TEST YOUR THINKING

The projects in this section are meant to have you utilize all of the skills that you have acquired throughout this chapter. The answers to these projects can be found at the companion Web site to this book, located at *http://www.phptr.com/rischert*.

Visit the Web site periodically to share and discuss your answers.

In this chapter you learned about data, how data is organized in tables, and how the relationships among it is depicted in a schema diagram. Based on your newly acquired knowledge, design a schema diagram based on the fictional ACME Construction Company. Draw on your own work experience to design the following components.

1) Draw boxes for these three tables: EMPLOYEE, POSITION, and DEPARTMENT.
2) Create at least three columns for each of the tables and designate a primary key for each table.
3) Create relationships among the tables that make sense to you. At least one table should have a self-referencing relationship. Hint: Be sure to include the necessary foreign key columns.
4) Think about which columns should NOT allow NULL values.