Database Concepts

Seventh Edition

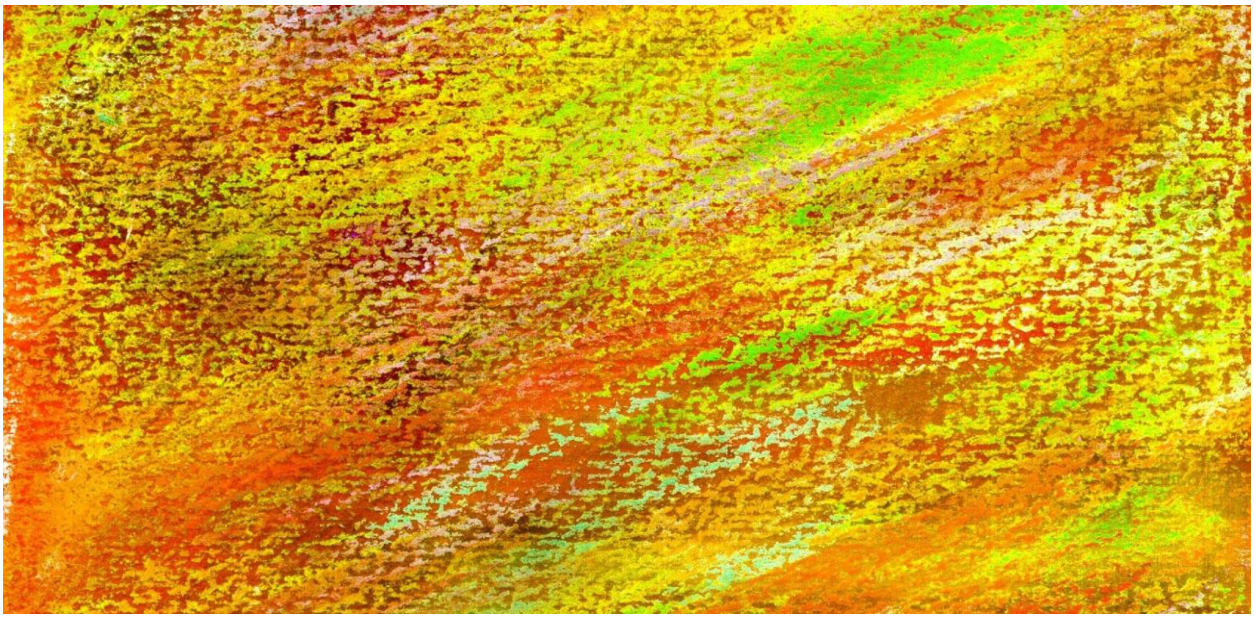
David M. Kroenke • David J. Auer

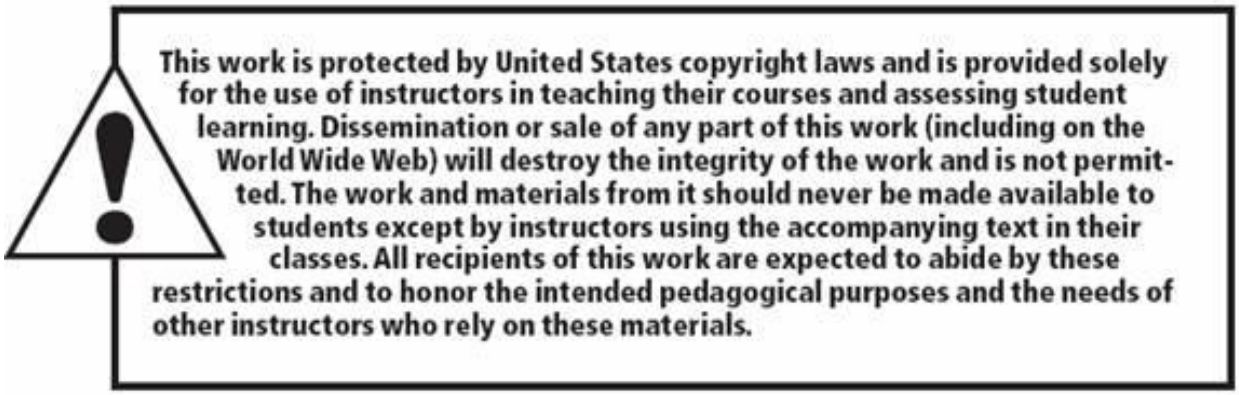
Instructor's Manual

Prepared by David J. Auer

CHAPTER TWO

THE RELATIONAL MODEL





Instructor's Manual to accompany:

Database Concepts (Seventh Edition)

David M. Kroenke and David J. Auer

CHAPTER OBJECTIVES

- Learn the conceptual foundation of the relational model
- Understand how relations differ from nonrelational tables
- Learn basic relational terminology
- Learn the meaning and importance of keys, foreign keys, and related terminology
- Understand how foreign keys represent relationships
- Learn the purpose and use of surrogate keys
- Learn the meaning of functional dependencies
- Learn to apply a process for normalizing relations

CHAPTER ERRATA

There are no known errors at this time. Any errors that are discovered in the future will be reported and corrected in the Online DBC e07 Errata document, which will be available at http://www.pearsonhighered.com/kroenke.

THE ACCESS WORKBENCH

Solutions to the *Access Workbench* exercises may be found in *Solutions to all Sections: The Access Workbench*, which is a separate document within the Instructor's Manual.

TEACHING SUGGESTIONS

- The Art Course database discussed in Chapter 1 is a good database to use for an in-class demo of the concepts in this chapter. The DBMS screenshots in Chapter 2 use that database as the example database. See the list, data and database files supplied, and use the following:
  - Microsoft Access 2013:
    - "Art Course List" in DBC-e07-Lists-And-Data.xlsx
    - DBC-e07-Art-Course-Database-CH01.accdb
  - Microsoft SQL Server 2014 Express Edition:
    - DBC-e07-MSSQL-Art-Course-Database-Create-Tables.sql
    - DBC-e07-MSSQL-Art-Course-Database-Insert-Data.sql
    - NOTE: Create a database diagram for the database

  - Oracle Database Express Edition 11*g* Release 2:
    - DBC-e07-ODB-Art-Course-Database-Create-Tables.sql
    - DBC-e07-ODB-Art-Course-Database-Insert-Data.sql
    - DBC-e07-ODB-Art-Course-Database-SQL-Queries-CH01.sql
  - MySQL 5.6:
    - DBC-e07-MySQL-Art-Course-Database-Create-Tables.sql
    - DBC-e07-MySQL-Art-Course-Database-Insert-Data.sql
- The goal of this chapter is to present an overview of the major elements of the relational model. This includes the definition of a relation, important terminology, the use of surrogate keys, and basic design principles.
- Students often misconstrue the statement that only a single element is allowed in a cell to mean that the cells must be fixed in length. One can have a variable length memo in a cell but that is considered, semantically, to be one thing. By the way, there are a number of reasons for this restriction. Perhaps the easiest to explain is that SQL has no means for addressing sub-elements in a cell.
- When students execute SQL SELECTs, they may generate relations with duplicate rows. Such results do not fit the definition of relations, but they are considered relations nonetheless. This is a good example of "theory versus practice".
- You may want to emphasize that foreign keys and the primary key that they reference need not have the same name. They must, however, have the same underlying set of values (domain). This means that the values not just look the same; it means that the values mean the same thing. A foreign key of CatName and a foreign key of ValentineNickName might look the same, but they do not mean the same thing. Using ValentineNickName as a foreign key to Name in the relation CAT would result in some weird results.
- Referential integrity constraints are important. You might ask the students to think of an example when a foreign key does not have a referential integrity constraint (answer: whenever a parent row is optional, say, STUDENTs need not have an ADVISER).
- We favor the use of surrogate keys. Unless there is a natural, numeric ID (like PartNumber), we almost always add a surrogate key to our database designs. Sometimes a surrogate key will be added even if there is a natural, numeric ID for consistency. Surrogate keys can cause problems (primarily patching up foreign keys) if the database imports data from other databases that either do not employ a surrogate key or use a different one. In some cases, institutions have developed policies for ensuring that surrogate keys are unique globally. It's probably best for the students to get into the habit of using them and consider not using them as an exception. Professional opinions vary on this, however.

- If you're using Oracle Database, then you'll need to teach the use of sequences to implement surrogate keys. Sequences are an awkward solution to this problem, however, and may be why surrogate keys are less used in the Oracle-world. Maybe there will be a better solution to them from Oracle in the future.
- The discussion of functional dependencies is critical—maybe the most important in the book. If students can understand that all tables do is record "data points" of functional dependencies, then normalization will be easier and seem more natural.
- In physics, because there are formulae like ***F = ma***, we need not store tables and tables of data recording data points for force, mass, and acceleration. The formula suffices for all data points. However, there is no formula for computing how much a customer of, say, American Airlines, owes for his or her ticket from New York to Houston. If we could say the cost of an airline ticket was $.05 per mile, then we could compute the cost of a ticket, and tables of airline flight prices would be unnecessary. But, we cannot; it all depends on … So, we store the data points for functional dependencies in tables.
- If we use domain/key normal form as the ultimate, then, insofar as functional dependencies are concerned, the domain/key definition that "every constraint is a logical consequence of domains and keys," comes down to Boyce-Codd Normal Form. Therefore, we proceed on good theoretical ground with the discussion as presented in this chapter.
- Students should understand three ambiguities in a null value. This understanding will help them comprehend the issues addressed by INNER and OUTER joins in the next chapter.
- Exercises 2.40 and 2.41 deal with multivalued dependencies and fourth normal form (4NF). These are instructive as they show students how to deal with situations where the value of one column in a table is associated with several values of another attribute in (at least initially) the same table. This is an important concept, and after BCNF it is the next important concept students need to understand about normalization.

ANSWERS TO REVIEW QUESTIONS

2.1 Why is the relational model important?

It is the single most important standard in database processing and is used for the design and implementation of almost every commercial database worldwide.

2.2 Define the term **entity** *and give an example of an entity (other than the one from this chapter).*

Entity is the formal name for a "thing" that is being tracked in a database, and is defined as something of importance to the user that needs to be represented in the database.

Example: TEXTBOOK

2.3 List the characteristics a table must have to be considered a relation.

- Rows contain data about an entity.
- Columns contain data about attributes of the entity
- Cells of the table hold a single value.
- All entries in a column are of the same kind.
- Each column has a unique name.
- The order of the columns is unimportant.
- The order of the rows is unimportant.

2.4 Give an example of a relation (other than one from this chapter).

Example: TEXTBOOK (ISBN, Title, Publisher, Copyright)

2.5 Give an example of a table that is not a relation (other than one from this chapter).

Example: TEXTBOOK (ISBN, Title, Publisher, Copyright, Authors)

A table is not a relation when there are multiple author names in the Authors column.

2.6 Under what circumstances can an attribute of a relation be of variable length?

It can be of a variable length, if that attribute is considered to be a single thing like a memo or other variable length data item.

2.7 Explain the use of the terms **file**, **record**, *and* **field**.

These terms are synonyms for table, row, and column. These terms, however, generally refer to pre-relational bases.

2.8 *Explain the use of the terms* **relation**, **tuple**, *and* **attribute**.

These terms are synonyms for table, row, and column. These terms, however, are the ones used in relational database theory.

2.9 *Under what circumstances can a relation have duplicate rows?*

When manipulating a relation with a DBMS we may end up with duplicate rows. Although in theory we should eliminate the duplicates, in practice this is often not done.

2.10 *Define the term* **unique key** *and give an example.*

A unique key is a column whose values identify one and only one row.

Example: TEXTBOOK (*ISBN*, Title, Publisher, Copyright)

where **ISBN** is a unique identifier.

2.11 *Define the term* **nonunique key** *and give an example.*

A nonunique key not only identifies a row, but it potentially identifies more than one row.

EXAMPLE: TEXTBOOK (ISBN, Title, Publisher, Copyright)

Publisher is a nonunique identifier.

2.12 *Give an example of a relation with a unique composite key.*

EXAMPLE: APARTMENT (BuildingNumber, ApartmentNumber, NumberOfBedrooms, Rent)

where **(BuildingNumber, ApartmentNumber)** is a unique composite key.

2.13 *Explain the difference between a primary key and a candidate key.*

Both are unique identifiers. One is chosen to be the identifier for the relation and for foreign keys based on the relation. The other could be chosen as well, but since it is not, it is called a candidate.

2.14 *Describe four uses of a primary key.*

A primary key can be used

- to identify a row.
- to represent the row in foreign keys.
- to organize storage for the relation.
- as a basis for indexes and other structures to facilitate searching in storage.

2.15 *What is a* **surrogate key**, *and under what circumstances would you use one?*

A surrogate key is a unique, numeric identifier that is appended to a relation to serve as the primary key.

2.16 *How do surrogate keys obtain their values?*

They are supplied automatically by the DBMS.

2.17 *Why are the values of surrogate keys normally hidden from users on forms, queries, and reports?*

Surrogate keys are normally hidden because they usually have no meaning to the users.

2.18 *Explain the term* **foreign key** *and give an example.*

A foreign key creates the relationship between the tables; its key value corresponds to a primary key in a relation other than the one where the key is a primary key.

EXAMPLE: **TEXTBOOK (ISBN, Title, *Publisher*, Copyright)**

PUBLISHER (PublisherName, Street, City, State, Zip)

Publisher in TEXTBOOK is a foreign key that references **PublisherName** in PUBLISHER.

2.19 *Explain how primary keys and foreign keys are denoted in this book.*

Primary keys are underlined and foreign keys are in italics.

2.20 *Define the term* **referential integrity constraint** *and give an example of one.*

Referential integrity constraint is a rule specifying that every value of a foreign key matches a value of the primary key.

Example: **Publisher in TEXTBOOK must exist in PublisherName in PUBLISHER.**

2.21 *Explain three possible interpretations of a null value.*

Three possible interpretations are:

- Value not appropriate
- Value known to be blank
- Value appropriate and unknown

2.22 *Give an example of a null value (other than one from this chapter), and explain each of the three possible interpretations for that value.*

An example of null value would be: Null value for the attribute DeceasedDate in the table SUBSCRIBER.

- The subscriber may be a corporation and a value is inappropriate.
- The subscriber may be alive, and the value is known to be blank.
- The subscriber may be dead, but the date of death is unknown, and the value is appropriate, but not none.

2.23 *Define the terms* **functional dependency** *and* **determinant,** *using an example not from this book.*

A functional dependency is a logical relationship in which the value of one item in the relationship can be determined by knowing the value of the other item.

EXAMPLE: ISBN → Title

This means that if the ISBN (of a textbook) is known, then we will also know (can determine) the title. The item on the left—the one whose value is known—is called the ***determinant***.

2.24 *In the following equation, name the functional dependency and identify the determinant(s):*

Area = Length × Width

The functional dependency is:

(Length, Width) → Area

(Length, Width) is the determinant.

Note this is different than saying "Length and Width are the determinants".

2.25 *Explain the meaning of the following expression:*

A → (B, C)

Given this expression, tell if it is also true that:

A → B

and

A → C

The functional dependency:

A → (B, C)

means that a value of A determines the value of both B and C.

Yes, it is true that

A → B and **A → C**

2.26 Explain the meaning of the following expression:

(D, E) → F

Given this expression, tell if it is also true that:

D → F

and

E → F

The functional dependency:

(D, E) → F

means that values of the pair (D, E) determine the value of F.

No, it is *not* true that

D → F and E → F

2.27 Explain the differences in your answers to questions 2.25 and 2.26.

A → (B, C) is just shorthand for **A → B** and **A → C**

However, **(D, E) → F** means that the composite, as a whole, identifies **F**.

For example:

EmployeeNumber → (FirstName, LastName)

This means that

EmployeeNumber → FirstName

and that

EmployeeNumber → LastName.

But:

(FirstName, LastName) → HireDate

does *not* mean that **FirstName → HireDate** (There could be lots of employees named "Bob".)

2.28 *Define the term* **primary key** *in terms of functional dependencies.*

A primary key is one or more attributes that functionally determines all of the other attributes.

2.29 *If you assume that a relation has no duplicate data, how do you know there is always at least one primary key?*

Because the collection of all the attributes in the relation can identify a unique row.

2.30 *How does your answer to question 2.29 change if you allow a relation to have duplicate data?*

It doesn't work—such tables do not have a primary key.

2.31 *In your own words, describe the nature and purpose of the normalization process.*

The purpose of the normalization process is to prevent update problems in the tables (relations) in the database. The nature of the normalization process is that we break up relations as necessary to ensure that every determinant is a candidate key.

2.32 *Examine the data in the Veterinary Office List—Version One in Figure 1-30 (see page 52), and state assumptions about functional dependencies in that table. What is the danger of making such conclusions on the basis of sample data?*

PetName → (PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

The danger is that there may be possibilities not apparent from sample data. For example, two owners might have pets with the same name.

2.33 *Using the assumptions you stated in your answer to question 2.32, what are the determinants of this relation? What attribute(s) can be the primary key of this relation?*

Attributes that can be the primary key are called candidate keys.

Determinants: **PetName, OwnerEmail, OwnerPhone**

Candidate keys: **PetName**

2.34 *Describe a modification problem that occurs when changing data in the relation in question 2.32 and a second modification problem that occurs when deleting data in this relation.*

Changes to owner data may need to be made in several rows.

Deleting data for the last pet of an owner deletes owner data as well.

2.35 *Examine the data in the Veterinary Office List—Version Two in Figure 1-31 (see page 55), and state assumptions about functional dependencies in that table.*

PetName → (PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

(PetName, Date) → (Service, Charge)

The last functional dependency assumes a pet is seen at most on one day and that there is no standard charge for a service.

2.36 *Using the assumptions you stated in your answer to question 2.35, what are the determinants of this relation? What attribute(s) can be the primary key of this relation?*

Determinants: **PetName, OwnerEmail, OwnerPhone, (PetName, Date)**

Candidate keys: **(PetName, Date)**

2.37 *Explain a modification problem that occurs when changing data in the relation in question 2.35 and a second modification problem that occurs when deleting data in this relation.*

Same as 2.34:

Changes to owner data may need to be made in several rows.

Deleting data for the last pet of an owner deletes owner data as well.

ANSWERS TO EXERCISES

2.38 *Apply the normalization process to the Veterinary Office List—Version One relation shown in Figure 1-30 (see page 55) to develop a set of normalized relations. Show the results of each of the steps in the normalization process.*

STEP ONE:

PET-AND-OWNER (PetName, PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

PET-AND-OWNER Candidate Keys: **PetName**

Is every determinant a candidate key?

NO—OwnerEmail and OwnerPhone are NOT candidate keys.

STEP TWO:

Break into two relations: OWNER and PET

OWNER (OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, {Foreign Key})

FOR OWNER:

Functional Dependencies:

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

OWNER Candidate Keys: **OwnerPhone, OwnerEmail**

Is every determinant a candidate key?

YES—OwnerEmail and OwnerPhone are candidate keys—Normalization complete!

We can choose either candidate key as primary key.

(A) IF WE USE OwnerPhone as primary key, THEN:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerPhone*)

Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerPhone)

PET Candidate Keys: **PetName**

Is every determinant a candidate key?

YES—PetName is a candidate key—Normalization complete!

FINAL NORMALIZED REALTIONS:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerPhone*)

(B) IF WE USE OwnerEmail as primary key, THEN:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerEmail*)

Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerEmail)

PET Candidate Keys: **PetName**

Is every determinant a candidate key?

YES—PetName is a candidate key—Normalization complete!

FINAL NORMALIZED REALTIONS:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerEmail*)

2.39 *Apply the normalization process to the Veterinary Office List—Version Two relation shown in Figure 1-31 (see page 56) to develop a set of normalized relations. Show the results of each of the steps in the normalization process.*

STEP ONE:

PET-AND-OWNER (PetName, PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail, Service, Date, Charge)

Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

(PetName, Date) → (Service, Charge)

The last functional dependency assumes a pet is seen at most on one day and that there is no standard charge for a service.

PET-AND-OWNER Candidate Keys: **(PetName, Date)**

Is every determinant a candidate key?

NO—PetName, OwnerEmail and OwnerPhone are NOT candidate keys.

STEP TWO:

Break into two relations: OWNER and PET-SERVICE

OWNER (OwnerLastName, OwnerFirstName, OwnerPhone, OwnerEmail)

PET-SERVICE (PetName, PetType, PetBreed, PetDOB, {Foreign Key}, Service, Date, Charge)

FOR OWNER:

Functional Dependencies:

OwnerEmail → (OwnerLastName, OwnerFirstName, OwnerPhone)

OwnerPhone → (OwnerLastName, OwnerFirstName, OwnerEmail)

OWNER Candidate Keys: **OwnerPhone, OwnerEmail**

Is every determinant a candidate key?

YES—OwnerEmail and OwnerPhone are candidate keys—Normalization complete!

We can choose either candidate key as primary key. We will use OwnerPhone.

If a student chooses OwnerEmail, the steps will be similar as shown in Exercise 2.37.

IF WE USE OwnerPhone as primary key, THEN:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET-SERVICE (PetName, PetType, PetBreed, PetDOB, *OwnerPhone*, Service, Date, Charge)

FOR PET-SERVICE:

Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerPhone)

(PetName, Date) → (Service, Charge)

The last functional dependency assumes a pet is seen at most on one day and that there is no standard charge for a service.

PET-AND-SERVICE Candidate Keys: **(PetName, Date)**

Is every determinant a candidate key?

NO—PetName is NOT a candidate key.

STEP THREE:

Break PET-SERVICE into two relations: PET and SERVICE

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerPhone*)

SERVICE (*PetName*, Date, Service, Charge)

PET Functional Dependencies:

PetName → (PetType, PetBreed, PetDOB, OwnerPhone)

PET Candidate Keys: **PetName**

Is every determinant a candidate key?

YES—PetName is a candidate key—Normalization complete!

SERVICE Functional Dependencies:

(PetName, Date) → (Service, Charge)

The functional dependency assumes a pet is seen at most on one day and that there is no standard charge for a service.

SERVICE Candidate Keys: **(PetName, Date)**

Is every determinant a candidate key?

YES—(PetName, Date) is a candidate key—Normalization complete!

FINAL NORMALIZED REALTIONS:

OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)

PET (PetName, PetType, PetBreed, PetDOB, *OwnerPhone*)

SERVICE (*PetName*, Date, Service, Charge)

2.40 Consider the following relation:

STUDENT (StudentNumber, StudentName, SiblingName, Major)

Assume that the values of SiblingName are the names of all of a given student's brothers and sisters; also assume that students have at most one major.

A. Show an example of this relation for two students, one of whom has three siblings and the other of whom has only two siblings.

StudentNumber	StudentName	SiblingName	Major
100	Mary Jones	Victoria	Accounting
100	Mary Jones	Slim	Accounting
100	Mary Jones	Reginald	Accounting
200	Fred Willows	Rex	Finance
200	Fred Willows	Billy	Finance

B. List the candidate keys in this relation.

STUDENT Candidate Keys: **(StudentNumber, SiblingName)**

This assumes that StudentName is not unique.

C. State the functional dependencies in this relation.

StudentNumber → (StudentName, Major)

(StudentNumber, SiblingName) → (StudentName, Major)

D. Explain why this relation does not meet the relational design criteria set out in this chapter (i.e., why this is not a well-formed relation).

Some attributes are functionally dependent on a part of the composite primary key.

E. *Divide this relation into a set of relations that meet the relational design criteria (that is, that are well formed).*

Break into two relations: **STUDENT and STUDENT-SIBLING**

STUDENT (StudentNumber, StudentName, Major)

STUDENT-SIBLING (*StudentNumber*, SiblingName)

FOR STUDENT-SIBLING:

Functional Dependencies:

(StudentNumber, SiblingName) → (StudentNumber)

(StudentNumber, SiblingName) → (SiblingName)

STUDENT-SIBLING Candidate Keys: **(StudentNumber, SiblingName)**

Is every determinant a candidate key?

YES—(StudentNum, SiblingName) is a candidate key—Normalization complete!

FOR STUDENT:

STUDENT (StudentNumber, StudentName, Major)

Functional Dependencies:

StudentNumber → (StudentName, Major)

STUDENT Candidate Keys: **StudentNumber**

Is every determinant a candidate key?

YES—StudentNumber is a candidate key—Normalization complete!

FINAL NORMALIZED REALTIONs:

STUDENT (StudentNumber, StudentName, Major)

STUDENT-SIBLING (*StudentNumber*, SiblingName)

2.41 *Alter question 2.40 to allow students to have multiple majors. In this case, the relational structure is:*

STUDENT (StudentNumber, StudentName, SiblingName, Major)

A. *Show an example of this relation for two students, one of whom has three siblings and the other of whom has one sibling. Assume that each student has a single major.*

StudentNumber	StudentName	SiblingName	Major
100	Mary Jones	Victoria	Accounting
100	Mary Jones	Slim	Accounting
100	Mary Jones	Reginald	Accounting
200	Fred Willows	Rex	Finance

B. *Show the data changes necessary to add a second major for only the first student.*

StudentNumber	StudentName	SiblingName	Major
100	Mary Jones	Victoria	Accounting
100	Mary Jones	Slim	Accounting
100	Mary Jones	Reginald	Accounting
200	Fred Willows	Rex	Finance
100	Mary Jones	Victoria	InfoSystems
100	Mary Jones	Slim	InfoSystems
100	Mary Jones	Reginald	InfoSystems

C. *Based on your answer to part B, show the data changes necessary to add a second major for the second student.*

StudentNumber	StudentName	SiblingName	Major
100	Mary Jones	Victoria	Accounting
100	Mary Jones	Slim	Accounting
100	Mary Jones	Reginald	Accounting
200	Fred Willows	Rex	Finance
100	Mary Jones	Victoria	InfoSystems
100	Mary Jones	Slim	InfoSystems
100	Mary Jones	Reginald	InfoSystems
200	Fred Willows	Rex	Accounting

D. *Explain the differences in your answers to parts B and C. Comment on the desirability of this situation.*

We had to add three rows in the first case—one major for each of the siblings of the student. If we didn't do that, it would appear the student has a sibling with one major, but doesn't have the sibling as a second major. This is nuts!

E. *Divide this relation into a set of well-formed relations.*

If we split STUDENT into two relations, STUDENT and STUDENT-SIBLING, then we get:

STUDENT (StudentNumber, StudentName, Major)

STUDENT-SIBLING (*StudentNumber*, SiblingName)

The relation is identical to the STUDENT-SIBS relation in 2.38 above, and is properly normalized. Now we need to check STUDENT-MAJOR.

FOR STUDENT:

STUDENT (StudentNumber, StudentName, Major)

Functional Dependencies:

StudentNumber → (StudentName)

(StudentNumber, Major) → StudentName

STUDENT Candidate Keys: **(StudentNumber, Major)**

Is every determinant a candidate key?

NO—StudentNumber is NOT a candidate key.

Break into two relations: **STUDENT-2 and STUDENT-MAJOR**

STUDENT-2 (StudentNumber, StudentName)

STUDENT-MAJOR (*StudentNumber*, Major)

FOR STUDENT-2:

Functional Dependencies:

StudentNumber → StudentName

STUDENT_2 Candidate Keys: **StudentNumber**

Is every determinant a candidate key?

YES—StudentNumber is a candidate key—Normalization complete!

FOR STUDENT-MAJOR:

Functional Dependencies:

(StudentNumber, Major) → StudentNumber

(StudentNumber, Major) → Major

STUDENT_2 Candidate Keys: **(StudentNumber, Major)**

Is every determinant a candidate key?

YES—(StudentNumber, Major) is a candidate key—Normalization complete!

FINAL NORMALIZED REALTIONs:

STUDENT-2 (StudentNumber, StudentName)

STUDENT-MAJOR (*StudentNumber*, Major)

STUDENT-SIBLING (*StudentNumber*, SiblingName)

2.42 *The text states that you can argue that "the only reason for having relations is to store instances of functional dependencies." Explain, in your own words, what this means.*

In a properly normalized relation, each row of the relation consists of a primary key value (which is a determinant) and attribute values (which are all functionally dependent on the primary key). Thus, properly normalized relations store instances of functional dependencies, and only instances of functional dependencies. So we can say that the purpose of relations is to store instances of functional dependencies.

2.43 Consider a table named ORDER_ITEM, with data as shown in Figure 2-26. The schema for ORDER_ITEM is:

	OrderNumber	SKU	Quantity	Price
1	1000	201000	1	300.00
2	1000	202000	1	130.00
3	2000	101100	4	50.00
4	2000	101200	2	50.00
5	3000	100200	1	300.00
6	3000	101100	2	50.00
7	3000	101200	1	50.00

ORDER_ITEM (OrderNumber, SKU, Quantity, Price)

Where SKU is a "Stocking Keeping Unit" number, which is similar to a part number. Here it indicates which product was sold on each line of the table. Note that one OrderNumber must have at least one SKU associated with it, and may have several. Use this table and the detailed discussion of normal forms of pages 88-89 to answer the following questions.

A. Define 1NF. Is ORDER_ITEM in 1NF? If not, why not, and what would have to be done to put it into 1NF? Make any changes necessary to put ORDER_ITEM into 1NF. If this step requires you to create an additional table, make sure that the new table is also in 1NF.

First Normal Form is any table that meets the definition of a relation (Figure 2.1 below). ORDER_ITEM is in 1NF.

1. Rows contain data about an entity
2. Columns contain data about attributes of the entity
3. Cells of the table hold a single value
4. All entries in a column are of the same kind
5. Each column has a unique name
6. The order of the columns is unimportant
7. The order of the rows is unimportant
8. No two rows may hold identical sets of data values

B. Define 2NF. Now that ORDER_ITEM is in 1NF, is it also in 2NF? If not, why not, and what would have to be done to put it into 2NF? Make any changes necessary to put ORDER_ITEM into 2NF. If this step requires you to create an additional table, make sure that the new table is also in 2NF.

Second Normal Form is any table that 1) in First Normal Form, and 2) all nonkey attributes are determined by the entire primary key. In this case, the Primary Key is OrderNumber, SKU. Because SKU alone determines Price (SKU → Price), ORDER_ITEM is NOT in 2NF.

This is solved by creating another table (PRODUCT), where SKU is both the Primary Key and Foreign Key in PRODUCT, and Price is moved out of ORDER_ITEM and into the PRODUCT Table.

ORDER_ITEM (OrderNumber, SKU, Quantity)

PRODUCT (SKU, Price)

C. Define 3NF. Now that ORDER_ITEM is in 2NF, is it also in 3NF? If not, why not, and what would have to be done to put it into 3NF? Make any changes necessary to put ORDER_ITEM into 3NF. If this step requires you to create an additional table, make sure that the new table and any other tables created in previous steps are also in 3NF.

Third Normal Form is any table that 1) in Second Normal Form, and 2) and no nonkey attributes are determined by any other nonkey attributes. Because the original question was not in Second Normal Form, it was NOT in Third Normal Form. The solution in B fixes this problem, and then both ORDER_ITEM and PRODUCT are in Third Normal Form.

D. Define BCNF. Now that ORDER_ITEM is in 3NF, is it also in BCNF? If not, why not, and what would have to be done to put it into BCNF? Make any changes necessary to put ORDER_ITEM into BCNF. If this step requires you to create an

additional table, make sure that the new table and any other tables created in previous steps are also in BCNF.

BCNF is any table that 1) in Third Normal Form, and 2) and all determinates are candidate keys. Because the original question was not in Second Normal Form, it was NOT in BCNF. The solution in B fixes this problem, and then both ORDER_ITEM and PRODUCT are in BCNF.

ANSWERS TO REGIONAL LABS CASE QUESTIONS

Regional Labs is a company that conducts research and development work on a contract basis for other companies and organizations. Figure 2-33 shows data that Regional Labs collects about projects and the employees assigned to them.

This data is stored in a relation (table) named PROJECT:

PROJECT (ProjectID, EmployeeName, EmployeeSalary)

FIGURE 2-33
Sample Data for Regional Labs

ProjectID	EmployeeName	EmployeeSalary
100-A	Eric Jones	64,000.00
100-A	Donna Smith	70,000.00
100-B	Donna Smith	70,000.00
200-A	Eric Jones	64,000.00
200-B	Eric Jones	64,000.00
200-C	Eric Parks	58,000.00
200-C	Donna Smith	70,000.00
200-D	Eric Parks	58,000.00

A. *Assuming that all functional dependencies are apparent in this data, which of the following are true?*

1. **ProjectID → EmployeeName** **FALSE**
2. **ProjectID → EmployeeSalary** **FALSE**
3. **(ProjectID, EmployeeName) → EmployeeSalary**

   TRUE, but only if **EmployeeName → EmployeeSalary**
4. **EmployeeName → EmployeeSalary** **TRUE**
5. **EmployeeSalary → ProjectID** **FALSE**
6. **EmployeeSalary → (ProjectID, EmployeeName)** **FALSE**

B. *What is the primary key of PROJECT?*

(ProjectID, EmployeeName)

C. *Are all the nonkey attributes (if any) dependent on the primary key?*

NO, EmployeeSalary is dependent only on EmployeeName

D. *In what normal form is PROJECT?*

1NF ONLY

E. *Describe two modification anomalies that affect PROJECT.*

The two modification anomalies that affect PROJECT are:

INSERTION: To give an employee a salary, we must first assign the employee to a project.

MODIFICATION: If we change a Salary, we have to change it in multiple places and may create inconsistent data.

F. *Is ProjectID a determinant? If so, based on which functional dependencies in part A?*

NO

G. *Is EmployeeName a determinant? If so, based on which functional dependencies in part A?*

YES **EmployeeName → EmployeeSalary**

H. *Is (ProjectID, EmployeeName) a determinant? If so, based on which functional dependencies in part A?*

YES **(ProjectID, EmployeeName) → EmployeeSalary**

I. *Is EmployeeSalary a determinant? If so, based on which functional dependencies in part A?*

NO Actually, for the data in Figure 2-23, it *is* a determinant. However, the dataset is ***too small*** to validate this determinant, and logically EmployeeSalary is *not* a determinant!

J. *Does this relation contain a transitive dependency? If so, what is it?*

NO

K. *Redesign the relation to eliminate modification anomalies.*

The following seems workable:

ASSIGNMENT (ProjectID, EmployeeName)

SALARY (EmployeeName, EmployeeSalary)

ANSWERS TO GARDEN GLORY PROJECT QUESTIONS

Garden Glory is a partnership that provides gardening and yard maintenance services to individuals and organizations. Garden Glory is owned by two partners. They employ two office administrators and a number of full- and part-time gardeners. Garden Glory will provide one-time garden services, but it specializes in ongoing service and maintenance. Many of its customers have multiple buildings, apartments, and rental houses that require gardening and lawn maintenance services.

Figure 2-34 shows data that Garden Glory collects about properties and services.

FIGURE 2-34

Sample Data for Garden Glory

PropertyName	Type	Street	City	ZIP	ServiceDate	Description	Amount
Eastlake Building	Office	123 Eastlake	Seattle	98119	5/5/2014	Lawn Mow	$ 42.50
Elm St Apts	Apartment	4 East Elm	Lynnwood	98223	5/8/2014	Lawn Mow	$ 123.50
Jeferson Hill	Office	42 West 7th St	Bellevue	98040	5/8/2014	Garden Service	$ 53.00
Eastlake Building	Office	123 Eastlake	Seattle	98119	5/10/2014	Lawn Mow	$ 42.50
Eastlake Building	Office	123 Eastlake	Seattle	98119	5/12/2014	Lawn Mow	$ 42.50
Elm St Apts	Apartment	4 East Elm	Lynnwood	98223	5/15/2014	Lawn Mow	$ 123.50
Eastlake Building	Office	123 Eastlake	Seattle	98119	5/19/2014	Lawn Mow	$ 42.50

A. *Using these data, state assumptions about functional dependencies among the columns of data. Justify your assumptions on the basis of these sample data and also on the basis of what you know about service businesses.*

From the data it appears that there are many functional dependencies that could be defined. Some examples are:

PropertyName → PropertyType

(PropertyName, Street) → (PropertyType, City, Zip)

(PropertyName, City) → (PropertyType, Street, Zip)

(PropertyName, Zip) → (PropertyType, Street, City)

(PropertyName, Description, ServiceDate) → Amount

None of these seem to be more than just coincidence, however. It would seem, for example, that an "Elm St Apts" could exist in more than one city—there are certainly enough cities with a street named Elm Street! There is simply not enough data to reply on it. Logically, it seems that we need one ID column—a surrogate key will be required here.

With regard to services, it would seem likely that a given service could be given to the same property, but on different dates. So, if we had a good determinant for property, then the last functional dependency would be true. So, the following seems workable:

PropertyID → (PropertyName, PropertyType, Street, City, Zip)

(PropertyID, Description, ServiceDate) → Amount

B. Given your assumptions in part A, comment on the appropriateness of the following designs:

1. PROPERTY (PropertyName, PropertyType, Street, City, Zip, ServiceDate, Description, Amount)

   NOT GOOD: For example, PropertyName does not determine ServiceDate.

2. PROPERTY (PropertyName, PropertyType, Street, City, Zip, ServiceDate, Description, Amount)

   NOT GOOD: There may be more than one service on a given date.

3. PROPERTY (PropertyName, PropertyType, Street, City, Zip, ServiceDate, Description, Amount)

   NOT GOOD: For example, (PropertyName, ServiceDate) does not determine Description since there may be more than one service at a property on a given date.

4. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip, ServiceDate, Description, Amount)

   NOT GOOD: For example, PropertyID does not determine ServiceDate.

5. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip, ServiceDate, Description, Amount)

   NOT GOOD: For example, (PropertyID, ServiceDate) does not determine Description since there may be more than one service at a property on a given.

6. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip, *ServiceDate*)

   and

   SERVICE (ServiceDate, Description, Amount)

   BETTER: ServiceDate is properly set up as a foreign key in PROPERTY. HOWEVER, this will limit the system to only one service per property—the foreign key is in the wrong table!

7. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip, ServiceDate)

   and:

   SERVICE (ServiceID, *ServiceDate*, Description, Amount)

   NOT GOOD: ServiceDate is supposedly a foreign key in PROPERTY, but isn't even a primary key in SERVICE. This simply doesn't work!

8. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip, *ServiceID*)

   and:

   SERVICE (ServiceID, ServiceDate, Description, Amount, *PropertyID*)

   NOT GOOD: ServiceID is properly used as a foreign key in PROPERTY, but we also have PropertyID as a foreign key in SERVICE. This simply doesn't work; there cannot be two foreign keys like this! The question then becomes: Which one should we keep?

9. PROPERTY (PropertyID, PropertyName, PropertyType, Street, City, Zip)

   and:

   SERVICE (ServiceID, ServiceDate, Description, Amount, PropertyID)

   GOOD! Finally the relationship is set up correctly. Now we can have many services (even on the same date) for one property.

C. Suppose Garden Glory decides to add the following table:

SERVICE-FEE (PropertyID, ServiceID, Description, Amount)

Add this table to what you consider to be the best design in your answer to part B. Modify the tables from part B as necessary to minimize the amount of data duplication. Will this design work for the data in Figure 2-31? If not, modify the design so that this data will work. State the assumptions implied by this design.

Here's the best design from part B:

PROPERTY(PropertyID, PropertyName, PropertyType, Street, City, Zip)

SERVICE(ServiceID, ServiceDate, Description, Amount, *PropertyID*)

Adding

SERVICE-FEE (PropertyID, ServiceID, ServiceDate, Amount)

means that we need to take PropertyID, ServiceDate, and Amount out of SERVICE.

Note that PropertyID of SERVICE-FEE is a foreign key in PROPERTY. Now, for this design to make sense, we need to allow for multiple services on a property by making (ServiceID, PropertyID, ServiceDate) the key in SERVICE-FEE.

PROPERTY(PropertyID, PropertyName, PropertyType, Street, City, Zip)

SERVICE (ServiceID, Description)

SERVICE-FEE (*PropertyID*, *ServiceID*, ServiceDate, Amount)

This means that a service can be applied to a property on different, but multiple, dates and that a property can have multiple, but different, services on the same date. This also means that a service can be applied to multiple, but different, properties on the same date. However, a property may not have the same service on the same date. All of this seems reasonable and will work with the data in Figure 2-29. Now we need to check with the users.

ANSWERS TO JAMES RIVER JEWELRY PROJECT QUESTIONS

[NOTE: The James River Jewelry Project Questions are available online for Appendix D, which can be downloaded from the textbook's Web site: www.pearsonhighered.com/kroenke. The solutions for these questions will be included in the Instructor's Manual for each chapter]

James River Jewelry is a small jewelry shop. While James River Jewelry does sell typical jewelry purchased form jewelry vendors, including such items as rings, necklaces, earrings, and watches, it specializes in hard-to-find Asian jewelry. Although some Asian jewelry is manufactured jewelry purchased from vendors in the same manner as the standard jewelry is obtained, many of the Asian jewelry pieces are often unique single items purchased directly from the artisan who created the piece (the term "manufactured" would be an inappropriate description of these pieces). It has a small but loyal clientele, and it wants to further increase customer loyalty by creating a frequent buyer program. In this program, after every 10 purchases, a customer will receive a credit equal to 50 percent of the sum of his or her 10 most recent purchases. This credit must be applied to the next (or "11th") purchase.

Figure D-1 shows data that James River Jewelry collects for its frequent buyer program.

Name	Phone	Email	InvoiceNumber	Date	PreTaxAmount
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1000	5/5/2014	$ 155.00
Fred Price	555-236-0091	Fred.Price@somewhere.com	1010	5/7/2014	$ 203.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1020	5/11/2014	$ 75.00
Pamela Birch	555-236-4493	Pamela.Birch@somewhere.com	1030	5/15/2014	$ 67.00
Richardo Romez	555-236-3334	Richard.Romez@somewhere.com	1040	5/15/2014	$ 330.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1050	5/16/2014	$ 25.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1060	5/16/2014	$ 45.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1070	5/18/2014	$ 445.00
Samantha Jackson	555-236-1095	Samantha.Jackson@somewhere.com	1080	5/19/2014	$ 72.00

Figure D-1: Sample Data for James River Jewelry

A. *Using these data, state assumptions about functional dependencies among the columns of data. Justify your assumptions on the basis of these sample data and also on the basis of what you know about retail sales.*

From the data it would appear:

Name → (Phone, Email)

Phone → (Name, Email)

Email → (Name, Phone)

However, these are based on a very limited dataset and cannot be trusted. For example, name is not a good determinant in a retail application; there may be many customers with the same name. It's also possible that some customers could have the same phone, even though they do not in this example.

Another functional dependency is:

InvoiceNumber → (Name, Phone, Email, InvoiceDate, PreTaxAmount)

B. *Given your assumptions in part A, comment on the appropriateness of the following designs:*

1. CUSTOMER (Name, Phone, Email, InvoiceNumber, InvoiceDate, PreTaxAmount)

   NOT GOOD: For example, Name does not determine InvoiceNumber because one customer may have made more than one purchase.

2. CUSTOMER (Name, Phone, Email, InvoiceNumber, InvoiceDate, PreTaxAmount)

   TRUE, but not normalized. For example, Email → Phone. And why is InvoiceNumber the key for data about a customer?

3. CUSTOMER (Name, Phone, Email, InvoiceNumber, InvoiceDate, PreTaxAmount)

   GOOD FOR CUSTOMERS, BUT NOT INVOICES: Given a unique Email address, Email works as a key for customer data. Unfortunately, Email does *not* determine InvoiceNumber and therefore is not a sufficient key.

4. CUSTOMER (CustomerID, Name, Phone, Email, InvoiceNumber, InvoiceDate, PreTaxAmount)

   GOOD FOR CUSTOMERS, BUT NOT INVOICES: A unique ID column is a good idea, and works as a key for customer data. Unfortunately, CustomerID does *not* determine InvoiceNumber and therefore is not a sufficient key.

5. CUSTOMER (Name, Phone, Email)

   and

   PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount)

   GETTING BETTER, BUT INCOMPLETE. We cannot be sure Name is unique, and the relationship between CUSTOMER and PURCHASE is not defined.

6. CUSTOMER (Name, Phone, Email)

   and

   PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, *Email*)

   GOOD. The design breaks up the themes and has a proper foreign key. However, the use of Email as a primary key may be a problem if two customers share an Email address.

7. CUSTOMER (Name, Email)

   and

   PURCHASE (InvoiceNumber, Phone, InvoiceDate, PreTaxAmount, *Email*)

   A GOOD DESIGN WAS JUST MADE BAD AGAIN. The design breaks up the themes and has a proper foreign key. However, why was Phone moved? It should have been left in CUSTOMER where it will only be entered once and where:

   Email → Phone

C. *Modify what you consider to be the best design in part B to include a column called AwardPurchaseAmount. The purpose of this column is to keep a balance of the customers' purchases for award purposes. Assume that returns will be recorded with invoices having a negative PreTaxAmount.*

   The best design in part B was number 6, so we'll put AwardPurchaseAmount in CUSTOMER. The result is:

   CUSTOMER (Name, Phone, Email, AwardPurchaseAmount)

   PURCHASE (InvoiceNumber, InvoiceDate, PreTaxAmount, *Email*)

   However, the problem with this design is that there's no history of prior AwardPurchaseAmounts.

D. *Add a new AWARD table to your answer to part C. Assume that the new table will hold data concerning the date and amount of an award that is given after a customer has purchased 10 items. Ensure that your new table has appropriate primary and foreign keys.*

The new table is:

AWARD (AwardID, AwardDate, AwardAmount, AwardPurchaseAmount, *Email*)

The other tables need to be adjusted, and the final design will be:

CUSTOMER (Name, Phone, Email)

PURCHASE (InvoiceNumber, InvoiceDate, PreTax Amount, Email, *AwardID*)

AWARD (AwardID, AwardDate, AwardAmount, AwardPurchaseAmount, *Email*)

Placing AwardID into PURCHASE as a foreign key allows for each purchase to be allocated to a particular award. Business rules need to be in place to ensure that the count of the number of PURCHASEs having a positive PreTaxAmount minus the count of the number having a negative PreTaxAmount never exceeds 10 for any given AWARD row.

ANSWERS TO THE QUEEN ANNE CURIOSITY SHOP PROJECT QUESTIONS

The Queen Anne Curiosity Shop sells both antiques and current-production household items that complement or are useful with the antiques. For example, the store sells antique dining room tables and new tablecloths. The antiques are purchased from both individuals and wholesalers, and the new items are purchased from distributors. The store's customers include individuals, owners of bed-and-breakfast operations, and local interior designers who work with both individuals and small businesses. The antiques are unique, although some multiple items, such as dining room chairs, may be available as a set (sets are never broken). The new items are not unique, and an item may be reordered if it is out of stock. New items are also available in various sizes and colors (for example, a particular style of tablecloth may be available in several sizes and in a variety of colors).

Figure 2-35 shows typical sales data for the Queen Anne Curiosity Shop, and Figure 2-36 shows typical purchase data.

FIGURE 2-35

Sample Sales Data for The Queen Anne Curiosity Shop

LastName	FirstName	Phone	InvoiceDate	InvoiceItem	Price	Tax	Total
Shire	Robert	206-524-2433	14-Dec-14	Antique Desk	3,000.00	249.00	3,249.00
Shire	Robert	206-524-2433	14-Dec-14	Antique Desk Chair	500.00	41.50	541.50
Goodyear	Katherine	206-524-3544	15-Dec-14	Dining Table Linens	1,000.00	83.00	1,083.00
Bancroft	Chris	425-635-9788	15-Dec-14	Candles	50.00	4.15	54.15
Griffith	John	206-524-4655	23-Dec-14	Candles	45.00	3.74	48.74
Shire	Robert	206-524-2433	5-Jan-15	Desk Lamp	250.00	20.75	270.75
Tierney	Doris	425-635-8677	10-Jan-15	Dining Table Linens	750.00	62.25	812.25
Anderson	Donna	360-538-7566	12-Jan-15	Book Shelf	250.00	20.75	270.75
Goodyear	Katherine	206-524-3544	15-Jan-15	Antique Chair	1,250.00	103.75	1,353.75
Goodyear	Katherine	206-524-3544	15-Jan-15	Antique Chair	1,750.00	145.25	1,895.25
Tierney	Doris	425-635-8677	25-Jan-15	Antique Candle Holders	350.00	29.05	379.05

FIGURE 2-36

Sample Purchase Data for The Queen Anne Curiosity Shop

PurchaseItem	PurchasePrice	PurchaseDate	Vendor	Phone
Antique Desk	1,800.00	7-Nov-14	European Specialties	206-325-7866
Antique Desk	1,750.00	7-Nov-14	European Specialties	206-325-7866
Antique Candle Holders	210.00	7-Nov-14	European Specialties	206-325-7866
Antique Candle Holders	200.00	7-Nov-14	European Specialties	206-325-7866
Dining Table Linens	600.00	14-Nov-14	Linens and Things	206-325-6755
Candles	30.00	14-Nov-14	Linens and Things	206-325-6755
Desk Lamp	150.00	14-Nov-14	Lamps and Lighting	206-325-8977
Floor Lamp	300.00	14-Nov-14	Lamps and Lighting	206-325-8977
Dining Table Linens	450.00	21-Nov-14	Linens and Things	206-325-6755
Candles	27.00	21-Nov-14	Linens and Things	206-325-6755
Book Shelf	150.00	21-Nov-14	Harrison, Denise	425-746-4322
Antique Desk	1,000.00	28-Nov-14	Lee, Andrew	425-746-5433
Antique Desk Chair	300.00	28-Nov-14	Lee, Andrew	425-746-5433
Antique Chair	750.00	28-Nov-14	New York Brokerage	206-325-9088
Antique Chair	1,050.00	28-Nov-14	New York Brokerage	206-325-9088

A. *Using these data, state assumptions about functional dependencies among the columns of data. Justify your assumptions on the basis of these sample data and also on the basis of what you know about retail sales.*

From the sample sales data it would appear:

LastName → (FirstName, Phone)

FirstName → (LastName, Phone)

Phone → (LastName, FirstName)

Price → (Tax, Total)

(LastName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

(FirstName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

(PhoneName, InvoiceDate, InvoiceItem) → (Price, Tax, Total)

However, these are based on a very limited dataset and cannot be trusted. For example, name is not a good determinant in a retail application; there may be many customers with the same name. It's also possible that some customers could have the same phone, even though they do not in this example. The one trustable functional dependency here is:

Price → (Tax, Total)

From the sample purchase data it would appear:

(InvoiceItem, PurchasePrice) → (PurchaseDate, Vendor, Phone)

(InvoiceItem, PurchasePrice, PurchaseDate) → (Vendor, Phone)

(PurchasePrice, PurchaseDate) → Invoice (InvoiceItem, Vendor, Phone)

Vendor → Phone

Phone → Vendor

However, these are based on a very limited dataset and cannot be trusted. For example, Item is not a good determinant in a retail application; there may be many Items with the same designator. It's also possible that multiple purchases on the same purchase date will be for the purchase price. The most trustworthy functional dependencies here are:

Vendor → Phone

Phone → Vendor

But, the first of these may fail if the vendor has multiple phone numbers.

B. *Given your assumptions in part A, comment on the appropriateness of the following designs:*

1. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

NOT GOOD. There may be many customers with the same last name.

2. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. There may be many customers with the same last name and first name.

3. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. Phone will be fairly unique, and combined with FirstName may overcome the problem of two people with the same phone number being in the customer list (but not necessarily—what if three students are sharing an apartment, and two of them are Bill Smith and Bill Jones?). However, this will not determine the purchase information of purchase date, etc.

4. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. There may be many customers with the same last name and first name, and some of them may make a purchase on the same date. The only good thing about this is that it is starting to address the purchase information. If (LastName, FirstName) was unique, then customers would be limited to one purchase per day.

5. CUSTOMER (LastName, FirstName, Phone, Email, InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. We're still trying to make the unworkable actually work. Same objections as above in 4, except that now customers would be limited to one of a particular item per day. For example, a customer could not purchase two antique chairs on the same day!

6. CUSTOMER (LastName, FirstName, Phone, Email)

   and:

   SALE (InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. Finally the customer and purchase data is effectively broken up, but there is no foreign key to link the two tables. In CUSTOMER, using (LastName, FirstName) is still a problem. In SALE, with InvoiceDate as the primary key there can only be one purchase per day!

7. CUSTOMER (LastName, FirstName, Phone, Email, *InvoiceDate*)

   and:

   SALE (InvoiceDate, InvoiceItem, Price, Tax, Total)

   NOT GOOD. We've got a foreign key of PurchaseDate in SALE. But everything else that was wrong in design 6 above is still a problem. Moreover, by using PurchaseDate as the foreign key, we limit the customer to only one purchase!

8. CUSTOMER (LastName, FirstName, Phone, Email, *InvoiceDate*, *InvoiceItem*)

   and:

   SALE (InvoiceDate, Item, Price, Tax, Total)

   STILL NOT GOOD. The customer is still limited to only one purchase! However, this is the best design of the bunch if we rework the foreign keys.

C. Modify what you consider to be the best design in part B to include surrogate ID columns called CustomerID and SaleID. How does this improve the design?

The best design in part B was number 8, so we'll put in the ID columns. These columns will become the new primary keys, and we'll need to adjust the foreign key so that it is in SALE. The result is:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, InvoiceItem, Price, Tax, Total)

We now have a clean design, and CUSTOMER is in BCNF. SALE is not in BCNF because

Price → (Tax, Total)

We could further normalize SALE, but we will intentionally leave it this way. This is called **denormalization**, and is discussed in Chapter 5. The point is that creating the extra table (PRICE_TAX_TOTAL) is more trouble than it is worth. (Can you imagine what the data for that table would look like?)

The primary key problems with both tables are resolved, and now a customer can purchase as many of an item on the same date as he or she wants to!

D. Modify the design in part C by breaking SALE into two relations named SALE and SALE_ITEM. Modify columns and add additional columns as you think necessary. How does this improve the design?

The main problem with the design in part C is that only one item can be included in each sale. Moving items into a SALE_ITEM table linked SALE will allow multiple items to be purchased as part of one sale. We'll need to include SaleID as part of a composite primary key so that the sale items are grouped according to their corresponding SALE. SaleID will also be the foreign key linking to SALE. Item and Price now belong in SALE_ITEM, and we'll need to add a PreTaxTotal to SALE—tax will now only be calculated on the pretax total value of the sale. The result is:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, PreTaxTotal, Tax, Total)

SALE_ITEM (*SaleID*, SaleItemID, InvoiceItem, Price)

We now have an improved clean design, and the CUSTOMER and SALE_ITEM tables are in BCNF. SALES is still denormalized as discussed in part C.

We have good primary and foreign keys, and now a customer can purchase as many of an item on the same date as he or she wants to and all items can be part of just one sale!

E. Given your assumptions, comment on the appropriateness of the following designs:

1. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. There may be many purchases of the same item ("Candles").

2. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. There may be many purchases of the same item that cost the same amount of money ("Candles, $30.00").

3. PURCHASE (*PurchaseItem*, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. This limits purchases of a particular item to one per day.

4. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

   NOT GOOD. This limits purchases of a particular item to one per vendor.

5. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate)

   and:

   VENDOR (Vendor, Phone)

   NOT GOOD. It does, however separate the two themes of PURCHASE and VENDOR. However, there is no foreign key to link the tables. Moreover, this design still limits purchases of a particular item to one per day.

6. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor)

   and:

   VENDOR (Vendor, Phone)

   BETTER, BUT STILL NOT GOOD. It separates the two themes of PURCHASE and VENDOR, but they are ***not*** properly linked by a foreign key.

7. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, *Vendor*)

   and:

   VENDOR (Vendor, Phone)

   GOOD, but could be BETTER. It separates the two themes of PURCHASE and VENDOR, which are now properly linked by a foreign key. However, this design still limits purchases of a particular item to one per day.

   That said, this is the best design of the bunch.

F. Modify what you consider to be the best design in part E to include surrogate ID columns called PurchaseID and VendorID. How does this improve the design?

The best design in part E was number 7, so we'll put in the ID columns. These columns will become the new primary keys, and we'll need to adjust the foreign key in PURCHASE. The result is:

PURCHASE (PurchaseID, Item, PurchasePrice, PurchaseDate, *VendorID*)

VENDOR (VendorID, Vendor, Phone)

We now have a clean design, and PURCHASE is in BCNF. VENDOR is not in BCNF because

Vendor → (VendorID, Phone)

Phone → (VendorID, Vendor)

We could further normalize VENDOR, but we will intentionally leave it this way. As discussed in part D, this is called denormalization and is discussed in Chapter 5. The point is that creating the extra table (VENDOR_PHONE) is more trouble than it is worth.

The primary key problems with both tables are resolved, and now the Queen Anne Curiosity Shop can purchase as many of an item on the same date as needed.

G. *The relations in your design from part D and part F are not connected. Modify the database design so that sales data and purchase data are related.*

The connection between the two parts of the database design is the item being first purchased and then sold. Thus, we can create an integrated design by replacing InvoiceItem in SALE_ITEM with PurchaseID as a foreign key. We will rename Price in SALE_ITEM as SalePrice. Our final design will be:

CUSTOMER (CustomerID, LastName, FirstName, Phone, Email)

SALE (SaleID, *CustomerID*, InvoiceDate, PreTaxTotal, Tax, Total)

SALE_ITEM (*SaleID*, SaleItemID, *PurchaseID*, SalePrice)

PURCHASE (PurchaseID, PurchaseItem, PurchasePrice, PurchaseDate, *VendorID*)

VENDOR (VendorID, Vendor, Phone)