**Database Concepts**

8th Edition

### David M. Kroenke • David J. Auer • Scott L. Vandenberg • Robert C. Yoder

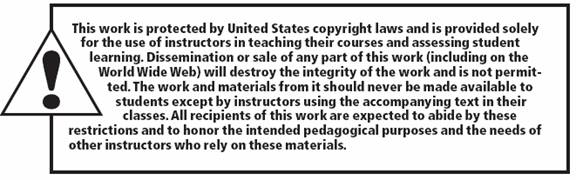
**Instructor’s Manual**

### Prepared by Robert C. Yoder

**Chapter Two**

**The Relational Model**

**All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America.**



Instructor’s Manual to accompany:

***Database Concepts (8th Edition)***

### David M. Kroenke • David J. Auer • Scott L. Vandenberg • Robert C. Yoder

© 2017, 2015, 2013, 2011, 2010, 2008 Pearson Education, Inc. Publishing as Prentice Hall

# 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 e08 Errata document, which will be available at [http://www.pearsonhighered.com/kroenke](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 2016:
* “Art Course List” in DBC-e08-Lists-And-Data.xlsx
* DBC-e08-Art-Course-Database-CH01.accdb
* Microsoft SQL Server 2016 Developer Edition:
* DBC-e08-MSSQL-Art-Course-Database-Create-Tables.sql
* DBC-e08-MSSQL-Art-Course-Database-Insert-Data.sql
* NOTE: Create a database diagram for the database
* Oracle Database Express Edition (XE) 11*g* Release 2:
* DBC-e08-ODB-Art-Course-Database-Create-Tables.sql
* DBC-e08-ODB-Art-Course-Database-Insert-Data.sql
* DBC-e08-ODB-Art-Course-Database-SQL-Queries-CH01.sql
* MySQL 5.7:
* DBC-e08-MySQL-Art-Course-Database-Create-Tables.sql
* DBC-e08-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 the 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

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.

1. 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**

1. List the characteristics a table must have to be considered a relation. Define the term **domain**, and explain the significance of the **domain integrity constraint** to 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.

A domain is a set of data that is of the same data type, and further, has the domain integrity constraint in that they must come from a set of allowed values, like the set of player positions on a baseball team.

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

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

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. Define the terms **candidate key** and **primary key**. Explain the difference between a primary key and a candidate key. Explain the significance of the entity integrity constraint to a primary key.

All candidate keys are unique identifiers. Any of the candidate keys could be the primary key, but one of the candidate keys is chosen to be the primary key for the relation and for foreign keys based on the relation. The other candidate keys will remain in the relation as alternate keys. The *entity integrity constraint* simply means that each value in a primary key must be unique with respect to the other values of the primary key in the relation.

1. 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.

1. 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. Surrogate keys are convenient and may improve database performance particularly when the candidate keys are composite and have long fields in them.

1. How do surrogate keys obtain their values?

They are supplied automatically by the DBMS.

1. 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.

1. 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 the primary key **PublisherName** in PUBLISHER.

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

Primary keys are underlined and foreign keys are in italics.

1. Define the term **referential integrity constraint** and give an example of one. How does the referential integrity constraint contribute to database integrity?

The **referential integrity constraint** is a rule specifying that every value of a foreign key matches a value of the primary key. It contributes to database integrity by ensuring that every row in a “child” relation connects to a “parent” relation, preventing errors in data entry.

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

1. Explain three possible interpretations of a null value.

Three possible interpretations are:

* + No value is appropriate for that attribute
  + The value is known to be blank for that attribute
  + Some value or values are appropriate but we don’t know which one it is yet

1. 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.

1. 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 only title for that ISBN. The item on the left—the one whose value is known—is called the ***determinant***.

1. 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” because we need both.

1. 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**

Answer: the functional dependency:

**A 🡪 (B, C)**

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

We can say that **A 🡪 B** and **A 🡪 C**

1. 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, since we need both values.

We **cannot** say that

**D 🡪 F and E 🡪 F**

1. 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”.)

1. 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 in the relation.

1. 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.

1. 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.

1. 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. Each relation will only have one theme in it.

1. Examine the data in the Veterinary Office List—Version One in Figure 1-34 (see page 63), 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 valid possibilities not apparent from sample data. For example, two owners might have pets with the same name.

1. 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**

1. 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.

1. Examine the data in the Veterinary Office List—Version Two in Figure 1-35 (see page 63), 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.

1. 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)**

1. 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

1. Apply the normalization process to the Veterinary Office List—Version One relation shown in Figure 1-34 (see page 63) 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 RELATIONS:**

**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! However, it may be unreasonable to require that PetName be unique.**

**FINAL NORMALIZED REALTIONS:**

**OWNER (OwnerPhone, OwnerLastName, OwnerFirstName, OwnerEmail)**

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

1. Apply the normalization process to the Veterinary Office List—Version Two relation shown in Figure 1-35 (see page 63) 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 2.38.

**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)**

1. What is the **multivalue, multicolumn problem?** What is a **multivalued dependency** and how is it resolved in 4NF? To answer these questions, 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.

* 1. Define and discuss the **multivalue, multicolumn problem**. Define and discuss a **multivalued dependency**.

The multivalue, multicolumn problem is basically a one-to-many relationship embedded in a single relation. A multivalued dependency is when a determinant is associated with a set of values. For example, suppose we want to store the names of children for each employee. We might be tempted to add a child1, child2, and perhaps a child3 attribute in the EMPLOYEE relation, or to duplicate the employee row for each child. This causes several problems. Duplicating data in rows will cause update anomalies. In addition, many employees may have less than three children, so many child attributes will remain blank. Suppose we hire a new employee with twelve children! We would have to add nine new child attributes to the EMPLOYEE relation AND change all SQL queries that display the children for each employee. It is best to split the relation so that the EMPLOYEE-CHILD relation is in a separate relation.

* 1. 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** |

* 1. List the candidate keys in this relation.

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

This assumes that StudentName is not unique.

* 1. State the functional dependencies in this relation.

**StudentNumber 🡪 (StudentName, Major)**

**(StudentNumber, SiblingName) 🡪 (StudentName, Major)**

* 1. 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.

* 1. Define and discuss 4NF and how 4NF can be used to allow a set of well-formed relations.

Multi-valued dependencies are directly handled by 4NF. The multiple values for each row in the original relation are split into a separate relation with a composite key consisting of the multi-value dependency attributes. Since StudentNumber multidetermines SiblingName, those two attributes should be placed into a new relation with the composite key (StudentNumber, SiblingName), and also the attribute SiblingName should be removed from the original relation.

* 1. 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)**

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

**STUDENT (StudentNumber, StudentName, SiblingName, Major)**

* 1. Define and discuss the **multivalue, multicolumn problem**. Define and discuss a **multivalued dependency**.

See the answer to Part A of Question 2.40 above.

* 1. 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** |

* 1. 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** |

* 1. Based on your answer to part C, 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** |

* 1. Explain the differences in your answers to parts C and D. 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. We only had to add a single row for the student with only one sibling. You can see how the redundant data is rapidly growing.

* 1. Define and discuss 4NF and how 4NF can be used to allow a set of well-formed relations.

See the answer to Part F of Question 2.40 above.

* 1. 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-SIBLING relation in 2.40 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:**

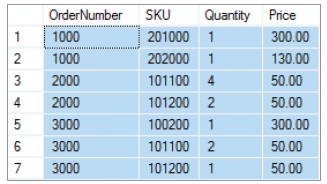
**STUDENT-2 (StudentNumber, StudentName)**

**STUDENT-MAJOR (*StudentNumber*, Major)**

**STUDENT-SIBLING (*StudentNumber*, SiblingName)**

1. 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.

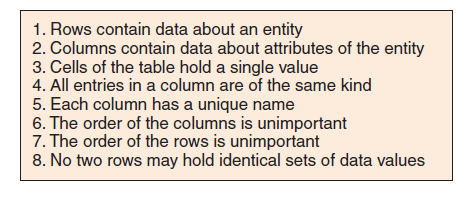
1. *Consider a table named ORDER\_ITEM, with data as shown in Figure 2-26. The schema for ORDER\_ITEM is:* 

**ORDER\_ITEM (OrderNumber, SKU, Quantity, Price)**

Where SKU is a “Stock 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 on pages 99-100 to answer the following questions.

* 1. 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. 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) is 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)

* 1. 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) is 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, and no nonkey attribute determines another nonkey attribute.

* 1. 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.

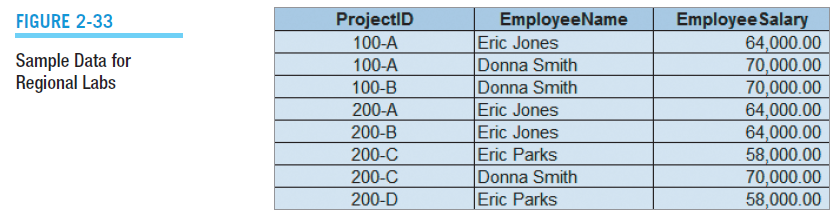
BCNF is any table that 1) is in Third Normal Form, and 2) and **all** determinants are candidate keys. Because the original question was not in Second Normal Form, it was NOT in BCNF. The solution in B fixes this problem, and then both ORDER\_ITEM and PRODUCT are in BCNF, and all determinants are candidate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

# 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)**



1. 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**

1. What is the primary key of PROJECT?

**(ProjectID, EmployeeName)**

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

**NO**, EmployeeSalary is dependent only on EmployeeName

1. In what normal form is PROJECT?

**1NF ONLY**

1. 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.

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

**NO**

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

**YES** **EmployeeName → EmployeeSalary**

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

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

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

**NO** Actually, for the data in Figure 2-33, it *is* a determinant. However, the dataset is ***too small*** to validate this determinant, and logically EmployeeSalary is *not* a determinant!

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

**NO**

1. Redesign the relation to eliminate modification anomalies.

The following seems workable:

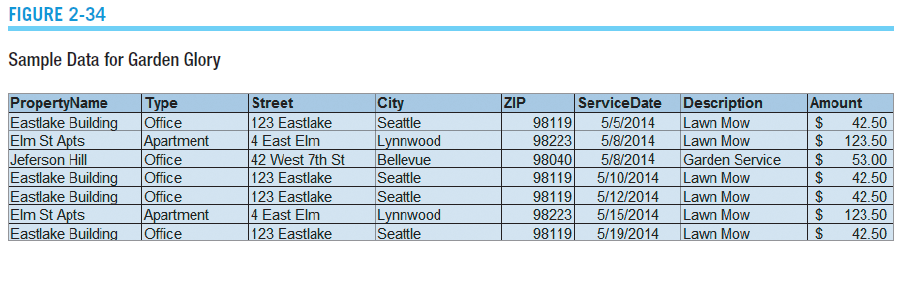
**ASSIGNMENT (ProjectID, *EmployeeName*)**

**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.



1. 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**

1. 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.

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

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

* 1. 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.

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

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

* 1. 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.

* 1. 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!

* 1. 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!

* 1. 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?

* 1. 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.

1. 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 2-34? 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)**

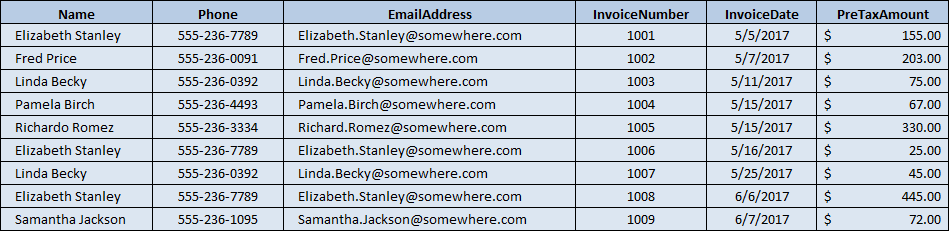
This means that a service can be applied to a property on different, but multiple, dates and that a property can have multiple, but different, services on the same date. This also means that a service can be applied to multiple, but different, properties on the same date. However, a property may not have the same service on the same date. All of this seems reasonable and will work with the data in Figure 2-34. 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](http://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 from 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). James River Jewelry 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.



1. 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, EmailAddress)**

**Phone 🡪 (Name, EmailAddress)**

**EmailAddress 🡪 (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, EmailAddress, InvoiceDate, PreTaxAmount)**

1. Given your assumptions in part A, comment on the appropriateness of the following designs:
   1. CUSTOMER (Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

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

* 1. CUSTOMER (Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

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

* 1. CUSTOMER (Name, Phone, EmailAddress, InvoiceNumber, InvoiceDate, PreTaxAmount)

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

* 1. CUSTOMER (CustomerID, Name, Phone, EmailAddress, 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.

* 1. CUSTOMER (Name, Phone, EmailAddress)

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.

* 1. CUSTOMER (Name, Phone, EmailAddress)

and

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

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

* 1. CUSTOMER (Name, EmailAddress)

and

PURCHASE (InvoiceNumber, Phone, InvoiceDate,

PreTaxAmount, *EmailAddress*)

**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:

**EmailAddress 🡪 Phone**

1. 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, EmailAddress, AwardPurchaseAmount)**

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

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

1. 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, *EmailAddress*)**

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

**CUSTOMER (Name, Phone, EmailAddress)**

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

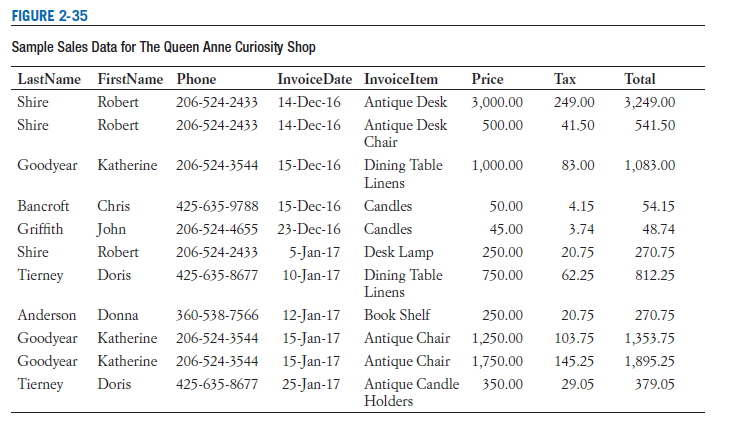
**AWARD (AwardID, AwardDate, AwardAmount, AwardPurchaseAmount, *EmailAddress*)**

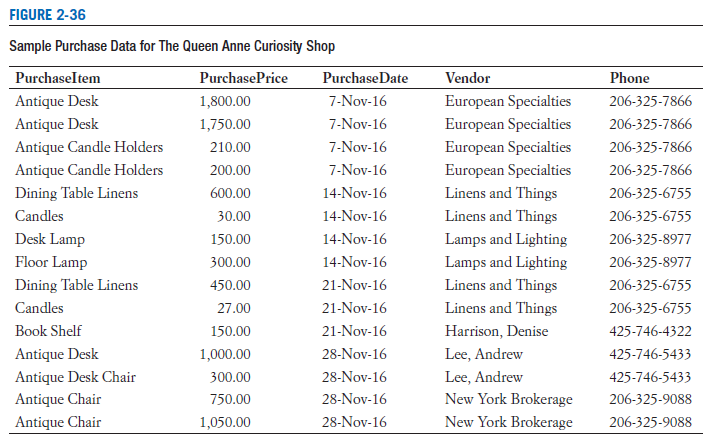
Placing AwardID into PURCHASE as a foreign key allows for each purchase to be allocated to a particular award. Business rules need to be in place to ensure that the count of the number of PURCHASEs having a positive PreTaxAmount minus the count of the number having a negative PreTaxAmount never exceeds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





1. 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:

**(PurchaseItem, PurchasePrice) 🡪 (PurchaseDate, Vendor, Phone)**

**(PurchaseItem, PurchasePrice, PurchaseDate) 🡪 (Vendor, Phone)**

**(PurchasePrice, PurchaseDate) 🡪 (PurchaseItem, Vendor, Phone)**

**Vendor 🡪 Phone**

**Phone 🡪 Vendor**

However, these are based on a very limited dataset and cannot be trusted. For example, PurchaseItem 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.

1. Given your assumptions in part A, comment on the appropriateness of the following designs:
2. CUSTOMER (LastName, FirstName, Phone, InvoiceDate, InvoiceItem, Price, Tax, Total)

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

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

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

1. CUSTOMER (LastName, FirstName, Phone, 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.

1. CUSTOMER (LastName, FirstName, Phone, 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.

1. CUSTOMER (LastName, FirstName, Phone, 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!

1. CUSTOMER (LastName, FirstName, Phone)

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!

1. CUSTOMER (LastName, FirstName, Phone, *InvoiceDate*)

and:

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

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

1. CUSTOMER (LastName, FirstName, Phone)

and:

SALE (InvoiceDate, InvoiceItem, Price, Tax, Total, *LastName, FirstName*)

**GOOD**. The customer can have lots of purchases, but not two of the same thing on the same day. Also, LastName and FirstName may not be the best choice for a primary key in CUSTOMER.

1. 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)**

**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!

1. 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)**

**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!

1. Given your assumptions, comment on the appropriateness of the following designs:
2. PURCHASE (PurchaseItem, PurchasePrice, PurchaseDate, Vendor, Phone)

**NOT GOOD.** There may be many purchases of the same item (“Candles”).

1. 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”).

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

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

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

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

1. 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.

1. 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.

1. 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.

1. 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.

1. 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)**

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

**SALE\_ITEM (*SaleID*, SaleItemID, *PurchaseID*, SalePrice)**

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

**VENDOR (VendorID, Vendor, Phone)**