Database Concepts 8th edition Kroenke Solutions Manual

SKU: 9780134601533-SOLUTIONS

Database Concepts

8th Edition

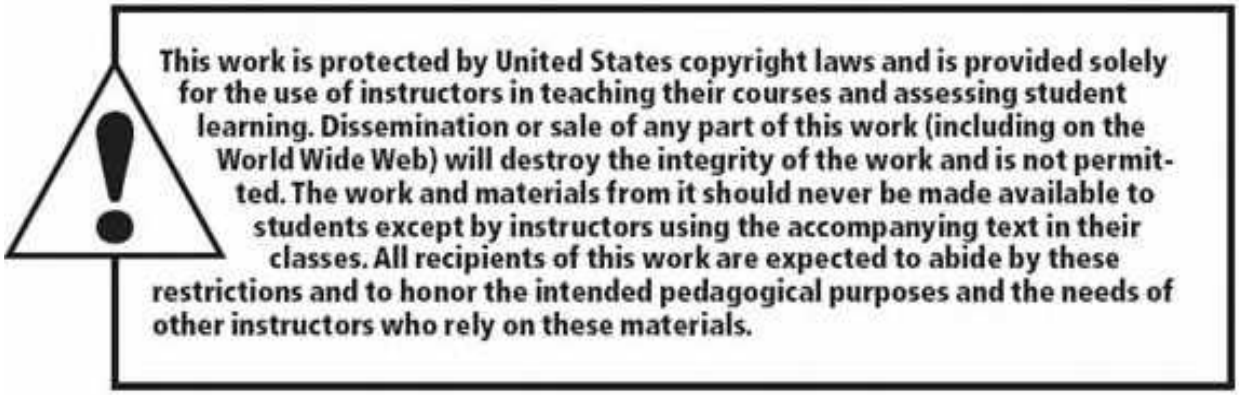
David M. Kroenke • David J. Auer • Scott L. Vandenberg • Robert C. Yoder

Instructor's Manual

Prepared by Robert C. Yoder

Chapter Two

The Relational Model



Instructor's Manual to accompany:

Database Concepts (8th Edition)

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

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.

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

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. 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.

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 *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.

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

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 the primary key **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. 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.**

2.21 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

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 only title for that ISBN. 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" because we need both.

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

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**

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, since we need both values.

We **cannot** say 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 in the relation.

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. Each relation will only have one theme in it.

2.32 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.

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 1-35 (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.

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 1-34 (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*)

2.39 *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)

*2.40 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.

*A. 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.

B. 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

C. List the candidate keys in this relation.

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

This assumes that StudentName is not unique.

D. State the functional dependencies in this relation.

StudentNumber → (StudentName, Major)

(StudentNumber, SiblingName) → (StudentName, Major)

E. 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.

F. 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.

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

See the answer to Part A of Question 2.40 above.

B. 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

C. 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

D. 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

E. 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.

F. 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.

G. 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)

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 "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.

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) 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)

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) 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.

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) 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)

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 2-33, 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 2-34? 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. 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.

Name	Phone	EmailAddress	InvoiceNumber	InvoiceDate	PreTaxAmount
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1001	5/5/2017	$ 155.00
Fred Price	555-236-0091	Fred.Price@somewhere.com	1002	5/7/2017	$ 203.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1003	5/11/2017	$ 75.00
Pamela Birch	555-236-4493	Pamela.Birch@somewhere.com	1004	5/15/2017	$ 67.00
Richardo Romez	555-236-3334	Richard.Romez@somewhere.com	1005	5/15/2017	$ 330.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1006	5/16/2017	$ 25.00
Linda Becky	555-236-0392	Linda.Becky@somewhere.com	1007	5/25/2017	$ 45.00
Elizabeth Stanley	555-236-7789	Elizabeth.Stanley@somewhere.com	1008	6/6/2017	$ 445.00
Samantha Jackson	555-236-1095	Samantha.Jackson@somewhere.com	1009	6/7/2017	$ 72.00

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, 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)

B. *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.

2. 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?

3. 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.

4. 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.

5. 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.

6. 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.

7. 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

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, EmailAddress, AwardPurchaseAmount)

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

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, *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.

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-16	Antique Desk	3,000.00	249.00	3,249.00
Shire	Robert	206-524-2433	14-Dec-16	Antique Desk Chair	500.00	41.50	541.50
Goodyear	Katherine	206-524-3544	15-Dec-16	Dining Table Linens	1,000.00	83.00	1,083.00
Bancroft	Chris	425-635-9788	15-Dec-16	Candles	50.00	4.15	54.15
Griffith	John	206-524-4655	23-Dec-16	Candles	45.00	3.74	48.74
Shire	Robert	206-524-2433	5-Jan-17	Desk Lamp	250.00	20.75	270.75
Tierney	Doris	425-635-8677	10-Jan-17	Dining Table Linens	750.00	62.25	812.25
Anderson	Donna	360-538-7566	12-Jan-17	Book Shelf	250.00	20.75	270.75
Goodyear	Katherine	206-524-3544	15-Jan-17	Antique Chair	1,250.00	103.75	1,353.75
Goodyear	Katherine	206-524-3544	15-Jan-17	Antique Chair	1,750.00	145.25	1,895.25
Tierney	Doris	425-635-8677	25-Jan-17	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-16	European Specialties	206-325-7866
Antique Desk	1,750.00	7-Nov-16	European Specialties	206-325-7866
Antique Candle Holders	210.00	7-Nov-16	European Specialties	206-325-7866
Antique Candle Holders	200.00	7-Nov-16	European Specialties	206-325-7866
Dining Table Linens	600.00	14-Nov-16	Linens and Things	206-325-6755
Candles	30.00	14-Nov-16	Linens and Things	206-325-6755
Desk Lamp	150.00	14-Nov-16	Lamps and Lighting	206-325-8977
Floor Lamp	300.00	14-Nov-16	Lamps and Lighting	206-325-8977
Dining Table Linens	450.00	21-Nov-16	Linens and Things	206-325-6755
Candles	27.00	21-Nov-16	Linens and Things	206-325-6755
Book Shelf	150.00	21-Nov-16	Harrison, Denise	425-746-4322
Antique Desk	1,000.00	28-Nov-16	Lee, Andrew	425-746-5433
Antique Desk Chair	300.00	28-Nov-16	Lee, Andrew	425-746-5433
Antique Chair	750.00	28-Nov-16	New York Brokerage	206-325-9088
Antique Chair	1,050.00	28-Nov-16	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:

(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.

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

1. CUSTOMER (<u>LastName</u>, FirstName, Phone, InvoiceDate, InvoiceItem, Price, Tax, Total)

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

2. CUSTOMER (LastName, FirstName, Phone, 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, 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, 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, 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)

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, *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!

8. 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.

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)

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)

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)

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

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

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

VENDOR (VendorID, Vendor, Phone)

Database Concepts

8th Edition

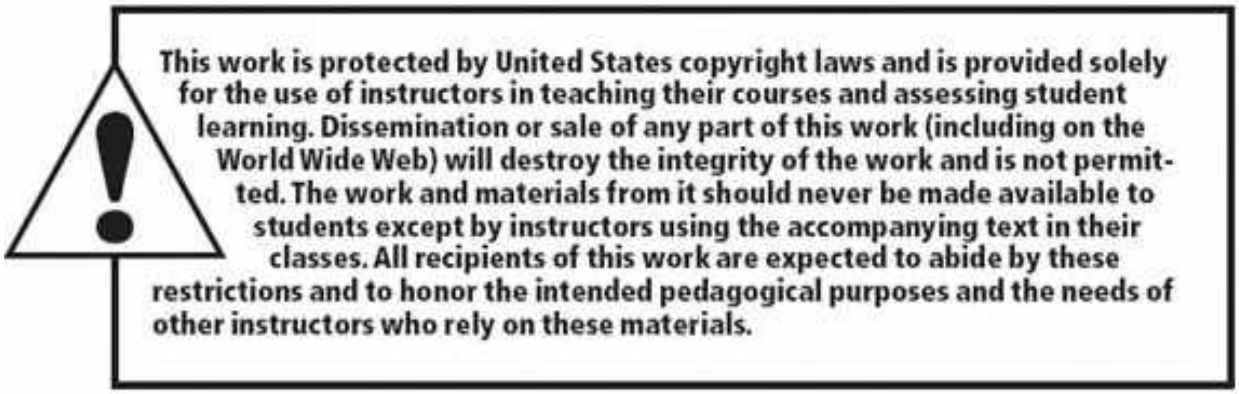
David M. Kroenke • David J. Auer • Scott L. Vandenberg • Robert C. Yoder

Instructor's Manual

Prepared by Scott L. Vandenberg

Appendix B

Getting Started with Oracle Database XE



Instructor's Manual to accompany:

Database Concepts (8th Edition)

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

APPENDIX OBJECTIVES

- Learn how to install Oracle Database XE
- Learn how to install Oracle SQL Developer
- Learn how to create a database in Oracle Database XE
- Learn how to submit SQL commands to create table structures
- Learn how to submit SQL commands to insert database data
- Learn how to submit SQL commands to query a database
- Learn how to install the Oracle ODBC Client software
- Learn how to use Microsoft Access as a front end to an Oracle Database XE database
- Learn how to import Microsoft Excel worksheet data into a database

APPENDIX 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.

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.

There is no section of *The Access Workbench* associated with this appendix.

NOTES ON MICROSOFT WINDOWS 10

This book uses the Microsoft Windows 10 operating system as the basis for screenshots and step-by-step instructions. However, with Windows 10, Microsoft has introduced a continuous update system that has already resulted in some fundamental differences in how different versions of Windows 10 look and operate.

For example, in the original version of Microsoft Windows 10, clicking the **Windows Start** button (or pressing the Windows key on the keyboard) displayed the menu shown in Figure 1. In this menu, we need to click the **All apps button** in order to see the **All apps menu** shown in Figure 2.

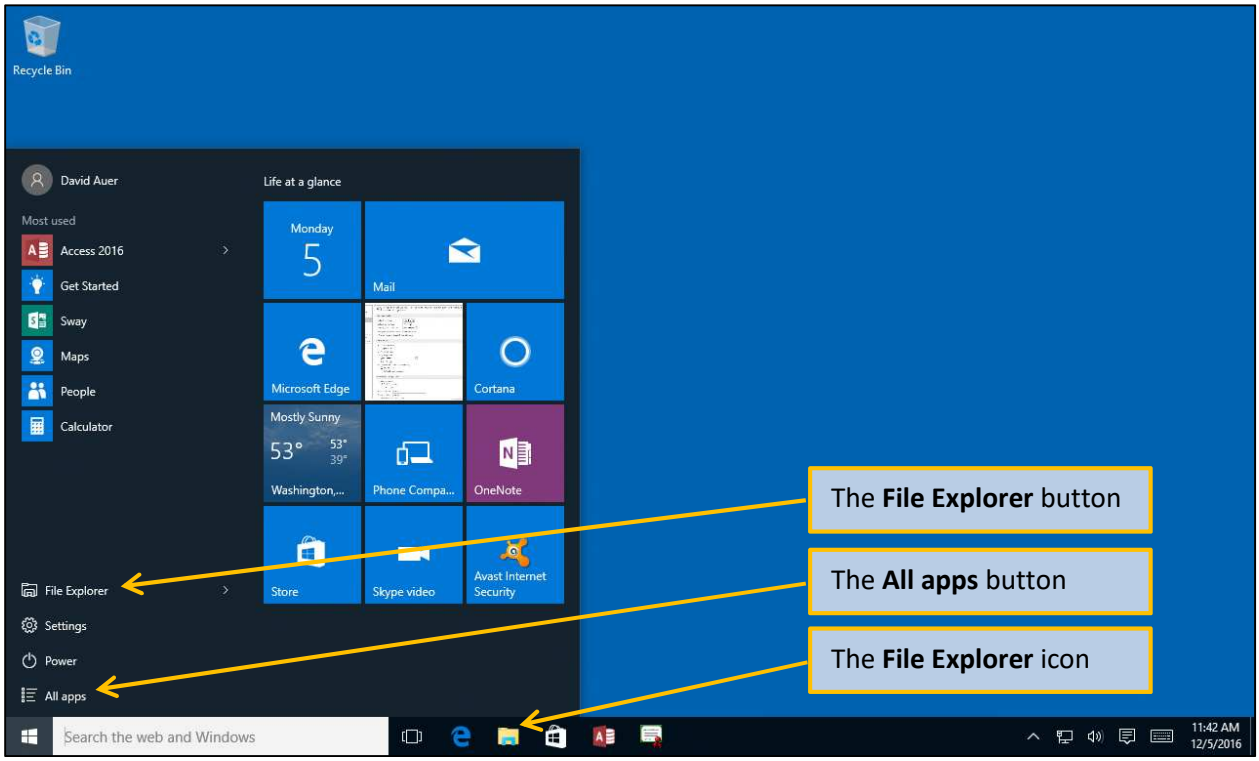


Figure 1 – Windows 10 Main Menu

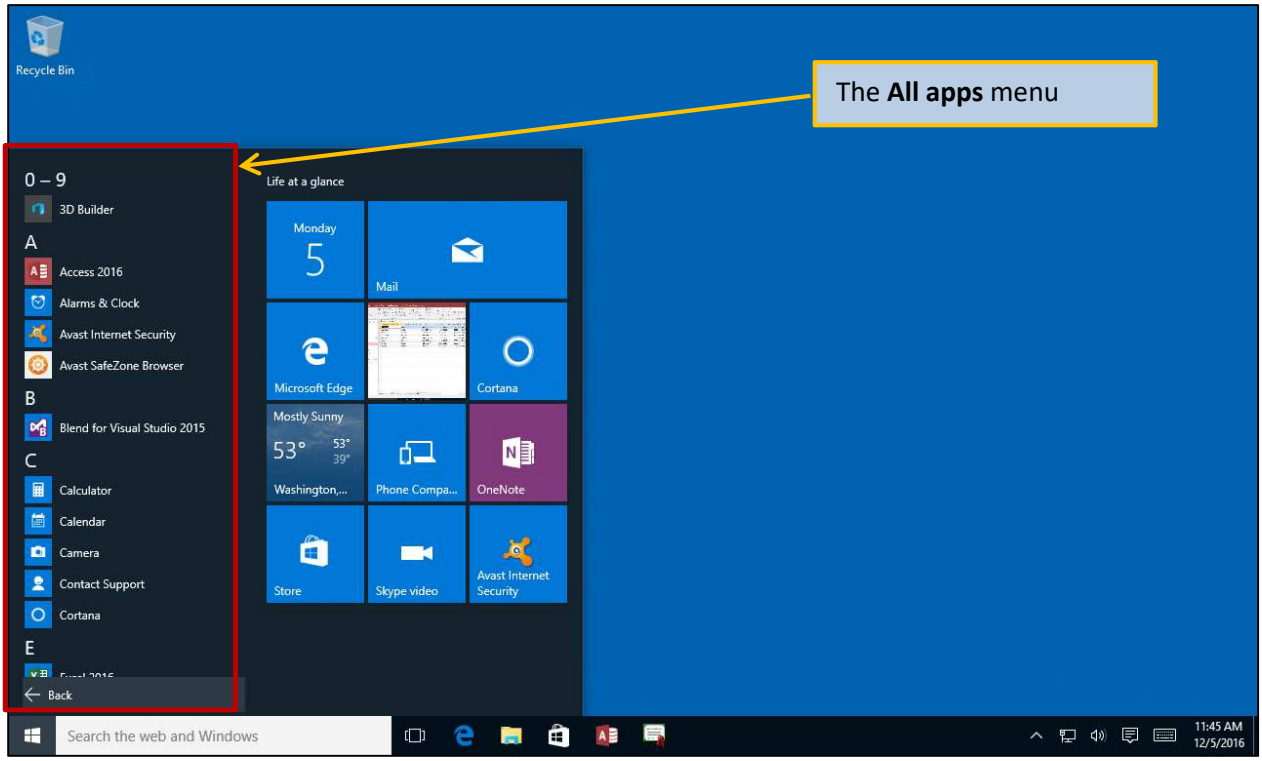


Figure 2 – Windows 10 All Apps Menu

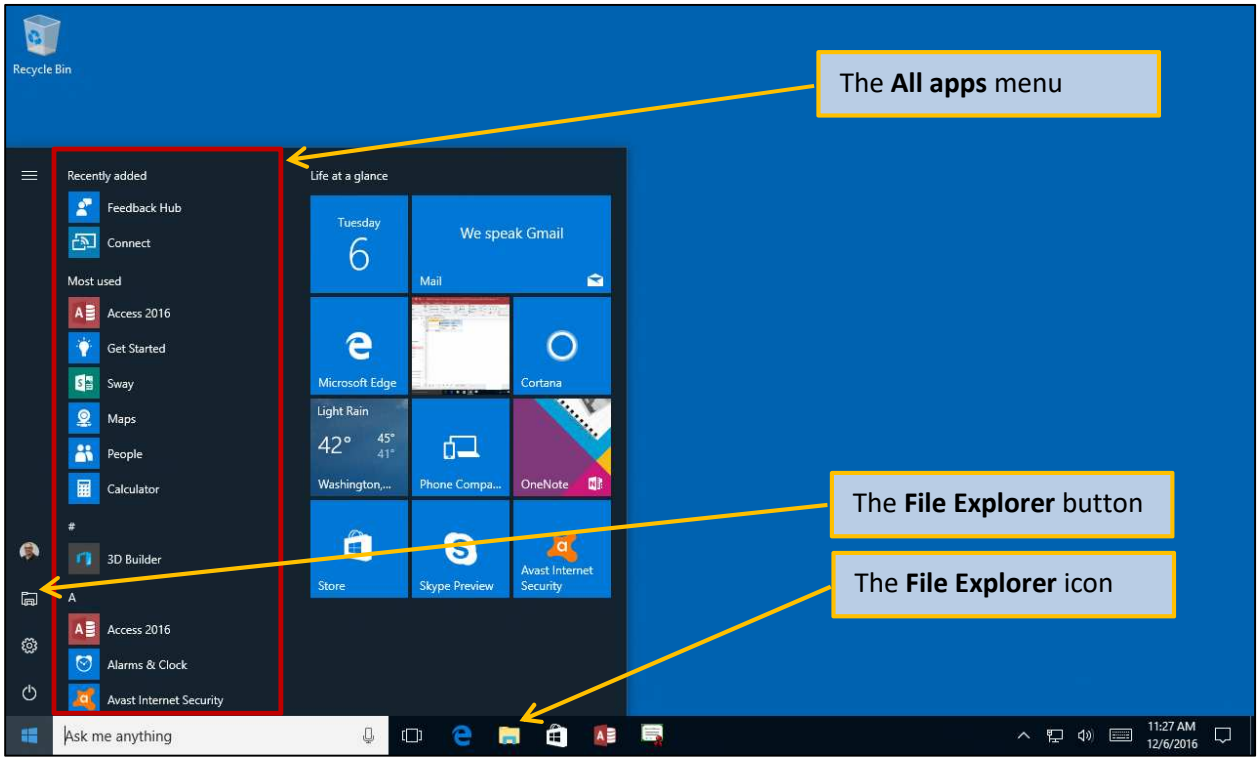


Figure 3 – Windows 10 Anniversary Update Main Menu with All Apps Menu Included

Microsoft then released the Windows 10 Anniversary Update (Feature update to Windows 10, version 1607) (see the blog discussion at https://blogs.windows.com/windowsexperience/2016/08/02/how-to-get-the-windows-10-anniversary-update/#K1CZuiw4auiuE9A5.97). One of the changes introduced in the Anniversary Update was a major change to the menu system. Now, as shown in Figure 3, the **All apps menu** is immediately available when the Start button is used (or the keyboard Windows key is pressed).

Therefore, note that the step by step instructions in this book may need to be altered for your use depending upon which version of Microsoft Windows 10 you or your students are using!

We recommend that you update Windows 10 to the Windows 10 Anniversary Update (Feature update to Windows 10, version 1607), and make sure it is patched with all updates to that version (at a minimum patched to Windows 10 Version 1607 update for August 23, 2016 (KB3176936), and the Windows 10 Version 1607 cumulative update for September 29, 2016 (KB3194496). We also recommend using the 32-bit version of Microsoft Office. This insures that all the examples discussed in this book will function properly.

TEACHING SUGGESTIONS

- For individual student personal use on their own computer, we recommend (and use in this book) Oracle Database Express Edition 11g Release 2. Oracle Database 12c can be used instead, but we recommend that only if the student does not have to install or configure the software.
- When you are using Oracle Database XE, the best text editor to use is the text editor built into Oracle SQL Developer. Take some time to show your students how to use it.
- It may be helpful to head off some confusion and frustration by listing some of the differences between SQL Server and Oracle early on, specifically:
  - The DATE format (default, TO_DATE, etc.)
  - Use of sequences to achieve autonumber fields
  - Lack of "ON UPDATE CASCADE" option
  - Disallowing of "AS" keyword in range variable declarations
  - Padding of CHAR fields with blanks and its effect on LIKE semantics (trailing blanks are not ignored)
- Make sure that your students work through Appendix B in conjunction with other material presented in the text by working through Appendix B in the following sequence:
  - Before starting Chapter 3 on SQL, install Oracle Database XE and Oracle SQL Developer by working through this appendix up to and including "Installing Oracle SQL Developer."
  - When studying the Chapter 3 sections on how to create and populate database tables, work up to and including "How Do I Use SQL Statements to Insert Database Data?" Both Chapter 3 and this appendix use the same WP database, and this work will show you how to create your own copy of the WP database.
  - When studying the Chapter 3 sections on how to use SQL Data Manipulation Language (DML) and SQL Data Definition Language (DDL), work through the section named "How Do I Work with SQL Queries in Oracle Database XE?" Both Chapter 3 and this appendix use the same WP database, and you can run the SQL Statements shown in Chapter 3 yourself, and see the results.
  - When studying the Appendix E material on SQL Views, you can run the SQL Statements shown in Appendix E yourself and see the results.
  - When studying the Appendix E material on SQL Persistent Stored Modules (SQL/PSM), you can run the Oracle versions of the SQL Statements, which can be found in the IRC Oracle files for Appendix E, yourself and see the results.
  - Work through the section "How Do I Import Microsoft Excel Data into an Oracle Database XE Table?" in this appendix to understand how to import Microsoft Excel data into a database table.

- Work through the section "How Do I Create an ODBC Connection from Microsoft Access 2016 to an Oracle Database XE Database?" in this appendix to understand how to use Microsoft Access as a development environment for an Oracle Database XE database.

ANSWERS TO REVIEW QUESTIONS

B.1 What is Oracle Database XE?

Oracle Database Express Edition 11g Release 2 (Oracle Database XE) is the latest version of the Oracle Database Express editions that Oracle has released. It is the least powerful of several versions of Oracle Database that Oracle has released. It is intended for general use, and can be downloaded for free from Oracle. It comes with a primitive but usable web-based interface for administering and interacting with databases. The latest full edition, generally available release of Oracle Database is Oracle Database 12c Release 1, which has more features and fewer limitations than Oracle Database XE. Note that there is no "version 12c" of Oracle Database XE.

B.2 What is the primary advantage of using Oracle Database XE instead of Microsoft Access?

Oracle Database XE handles SQL much better than Microsoft Access. It also provides more complete functionality for large databases and multiple users.

B.3 What are the two Oracle programs that are recommended as a necessary set of Oracle Database XE software products? In what order should you install these products?

1. Oracle Database XE
2. Oracle SQL Developer

Oracle SQL Developer requires that the Java JRE or JDK be installed. Install Oracle Database XE first, and then install the Oracle SQL Developer (after installing JDK if necessary).

B.4 How do you install Oracle Database XE?

Install Oracle Database XE as follows (this is an outline – we will not repeat the full set of instructions contained in the appendix):

1. Create an Oracle account if you don't already have one.
2. Download the Oracle Database XE software.
3. Extract the zipped contents to a folder.

4. Run the setup.exe program and follow the instructions in the installation wizard, as shown in the text of the appendix on pages B-8 through B-13.

B.5 What is the purpose of the Oracle Database XE Web utility?

The Oracle Database XE Web utility is the main tool of DBMS administration for Oracle Database XE. It is used to create user accounts and workspaces (databases).

B.6 How do you install Oracle SQL Developer?

Install Oracle SQL Developer as follows (this is an outline – we will not repeat the full set of instructions contained in the appendix on pages B-14 and B-15):

1. Download the proper version of the software from the Oracle website (taking care to note whether you need the 64-bit or 32-bit version and whether you need to install Java or already have Java installed or will download the version with Java included).

2. Extract the zipped contents to a folder.

3. If desired, create a desktop shortcut for sqldeveloper.exe.

B.7 What is the purpose of the Oracle SQL Developer utility?

Oracle SQL Developer is the graphical management utility for Oracle Database XE (and all other editions of Oracle), and using Oracle SQL Developer makes it much easier to work with Oracle Database XE. It allows connections to Oracle databases, creation and management of tables, running queries, etc.

B.8 How do you create a new database in Oracle Database XE?

To create the Oracle Database XE equivalent of what we call a database, create a new application workspace in the Oracle Database XE Web utility. This also creates a user account associated with that database.

B.9 How do you connect to a database in Oracle Database XE?

Use the connection tool in Oracle SQL Developer.

B.10 What is an SQL script? What types of SQL statements and commands can you run more efficiently as scripts?

An SQL script is a related group of SQL statements intended to be run at the same time. Scripts are efficient for processing groups of SQL statements such as:

1. A set of CREATE TABLE/ALTER TABLE/CREATE SEQUENCE commands to build a new database structure.

2. A set of INSERT commands when data needs to be added to a table.

B.11 What tool(s) can be used to create a script?

Scripts can be created in Oracle SQL Developer or in any other ASCII text editor, such as the Windows Notepad text editor. Oracle SQL Developer is recommended as it allows both editing and running of the scripts and individual commands within them, making testing easier.

B.12 What file extension should you use for SQL scripts?

The file extension *.sql* should be used so that such files are recognizable by Oracle SQL Developer and other relational database management systems.

B.13 How do you open and run a script in Oracle SQL Developer?

To open a script in Oracle SQL Developer, click the **Open File** button and follow its prompts to locate and open the script. The script will open up in a new SQL Worksheet tab. To run the script, first ensure the proper database connection is selected in the drop-down box in the upper-right corner (see Figure B-9(a)), and then click the **Run Script** button in the **SQL Worksheet toolbar**.

After the script is executed, a Message window appears below the script window indicating success (or displaying appropriate error messages).

B.14 How do you create database tables in Oracle SQL Developer?

To create database tables, create an SQL script containing the necessary SQL CREATE TABLE statements and any other necessary statements (ALTER TABLE, CREATE SEQUENCE, etc.). Save and name the script. Be sure the correct database has been selected in the drop-down database list, and run the script.

B.15 What is a sequence*? How are sequences used in Oracle Database XE?*

A sequence is an Oracle-supplied object that generates a sequential series of unique numbers. Sequences can start at any integer and be incremented by any non-zero integer. The following statement defines a sequence called seqAID that starts at 1 and is incremented by 1 each time it is used.

```
CREATE SEQUENCE seqAID INCREMENT BY 1 START WITH 1;
```

Sequences are used primarily to create surrogate keys in Oracle Database XE. Two sequence methods are important to us. The method NextVal provides the next value in a sequence and the method CurrVal provides the current value in a sequence. Thus, seqAID.NextVal provides the next value of the seqAID sequence. You can insert a row into an ARTIST table using a sequence, as follows:

```
INSERT INTO ARTIST (ArtistID, LastName, FirstName, Nationality)
VALUES (seqAID.NextVal, 'Miro', 'Joan', 'Spanish');
```


An ARTIST row will be created with the next value in the sequence as the value for ArtistID. Once this statement has been executed, you can retrieve the row just created with the CurrVal method, as follows:

```
SELECT *
FROM ARTIST
WHERE ArtistID = seqAID.CurrVal;
```

Here, seqAID.CurrVal returns the current value of the sequence, which is the value just used.

B.16 How do you populate database tables in Oracle SQL Developer?

To populate database tables, create an SQL script containing the necessary SQL INSERT statements. Save and name the script. Be sure the correct database has been selected in the drop-down database list, and run the script.

B.17 How do you create and run an SQL query in Oracle SQL Developer?

To run a query, first connect to the database you wish to query if it does not already have a connection in the Connections list (use the SQL Developer connections manager tool to do this). Then right-click on its connection name in the Connections list and choose the "Open SQL Worksheet" option. This opens up an **SQL Worksheet** within the SQL Developer window along with the SQL Worksheet toolbar. In the new worksheet tab, type the text of the SQL query you want to run, and then click the **Execute** button (right-pointing green triangle) in the SQL Worksheet toolbar.

The results appear in a tabbed **Query Result** window below the query window in a spreadsheet-style display. The size of the query window and the results window can be adjusted, and the column widths in the results display can be modified using standard Windows drag-and-drop techniques to help make more data visible. You can run multiple queries at the same time—clicking the SQL Worksheet button again will open another tabbed query window.

B.18 How do you import Microsoft Excel data into an Oracle Database XE database?

The first step is to normalize that data in the Microsoft Excel worksheet into one or more correctly formatted worksheets. The worksheets should not have titles, but should include the proper column headings.

We then use the **Oracle SQL Developer Data Import Wizard** as our tool for data import. This tool allows us to specify source files and types as well as column names and data types.

B.19 Why would you want to create an ODBC connection to link a Microsoft Access 2016 to an Oracle Database XE Database?

While Oracle Database XE is an excellent enterprise-class DBMS, the SQL Developer environment does not provide any application development tools. Oracle Forms and Reports and Oracle Application Express do have some of this functionality, but often it is useful to

quickly prototype and implement simple forms and reports using commonly-installed software. Microsoft Access 2016 fills this role and provides a set of application development tools such as forms, reports, stored queries, and menu systems (see Appendix H, "The Access Workbench—Section H—Microsoft Access 2016 Switchboards"). Thus, it would be useful to have a way to use Microsoft Access 2016 as the application development frontend for an Oracle Database XE database. We can connect Microsoft Access 2106 to an Oracle Database XE database via an **Open Database Connectivity (ODBC)** link between the two.

B.20 What is an ODBC DSN? Why is one needed?

An ODBC DSN is a data source, and DSN stands for data source name. The DSN stores the actual connection information, such as specific database to be used, user login name and password, etc., needed to establish an ODBC link to a database.

B.21 How do you create an ODBC connection to link a Microsoft Access 2016 database to an Oracle Database XE database?

1. Create an empty Access database and ensure that it is using **ANSI standard SQL (ANSI 92)**.
2. Use the **Get External Data-ODBC Database Wizard** in Access to create a new DSN via the **Select Data Source** dialog box. Be sure to choose the **Oracle in XE** driver and choose an appropriate name for the data source.
3. Establish the connection as shown by example in Figure B-19(f).
4. Complete the process by following the instructions as exemplified in Figures B-19(g, h) to select the DSN you just created and choose which table(s) from the Oracle Database XE database to link to.

ANSWERS TO EXERCISES

B.22 If you haven't already done so, download and install Oracle Database XE as described in the text. Use the default settings for the installation. Be sure that Oracle SQL Developer is also correctly installed.

Installation is easy and straightforward. The complete instructions on how to download and install Oracle Database XE and Oracle SQL Developer are in Appendix B, and will not be repeated here.

B.23 If you haven't already done so, work through the steps described in this appendix to create and populate the WP database.

For table creation, use the file: **DBC-e08-ODB-WP-Create-Tables.sql**

For data entry, use the file: **DBC-e08-ODB-WP-Insert-Data.sql**

B.24 Using Oracle Database XE and SQL Developer, run the following SQL queries from Chapter 3:

SQL-Query-CH03-01 through SQL-Query-CH03-32

SQL-Query-CH03-35 through SQL-Query-CH03-53 (remember that the "AS" keyword is not allowed by Oracle with the JOIN syntax)

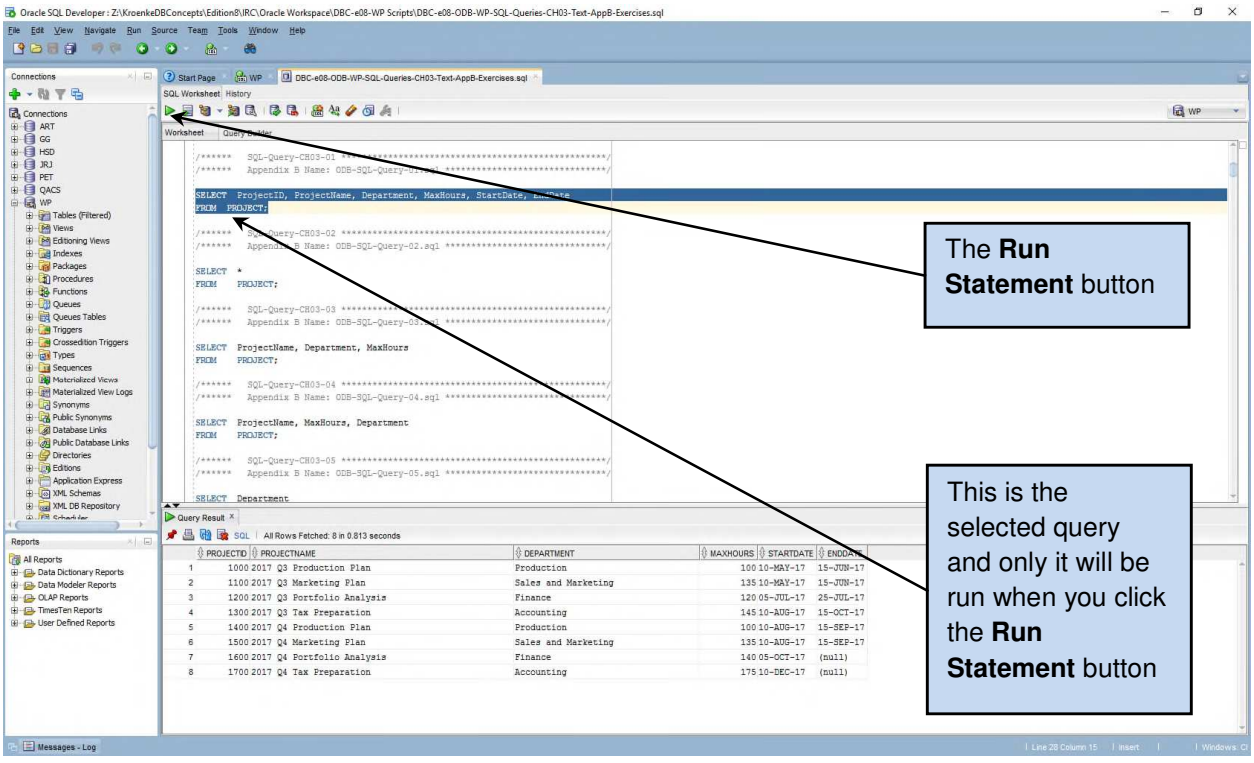
Save each query as follows:

- *Create and run each query in Oracle SQL Developer.*
- *After you have run each query, save the query. By default, Oracle Database XE saves each file as an SQL file with the file extension ***.sql**. Use this default setting unless your instructor tells you to use a different extension. Name your queries in* numerical sequence, *starting with the file name **ODB-SQL-Query-01.sql**.*

The solutions are in the script file:

DBC-e08-ODB-WP-SQL-Queries-CH03-Text-AppB-Exercises.sql

DO NOT RUN THIS FILE AS A SCRIPT! You can run individual queries from the script file by highlighting them or placing the cursor in them and then clicking the Run Statement button, as shown in the following screen shot.



B.25 Use Oracle SQL Developer to run one or more of the saved *SQL queries you created in the previous question:*

- *Open a query using the **Open File** button (or by selecting the **File** | **Open File** menu command). Note that the query is opened in a tabbed query window. Run the query.*
- *Use the **Open File** button (or the **File** | **Open File** menu command) to open and run another query in another tabbed window.*
- *Experiment with opening and closing windows and running various queries in these windows.*

These questions are self-explanatory and do not require separate solutions.

*B.26 Complete exercise 3.59 using Oracle Database XE and Oracle SQL Developer. Start each saved query name with **ODB-** and use the default **.sql** file extension. (The first saved query name should be **ODB-SQL-Query-AWE-3-1-A.sql**.)*

The solution for this Exercise is the same as the solution to Exercise 3.59. See the Instructor's Manual for Chapter 3 and the solutions in the file:

DBC-e08-ODB-WP-SQL-Queries-CH03-Exercises.sql

*B.27 Complete Exercise 3.60 using Oracle Database XE and Oracle SQL Developer. Start each saved script or query name with **ODB-** and use the default **.sql** file extension. Create as many scripts as you need for Parts A-D. The saved query name will be **ODB-SQL-Query-AWE-3-3-E.sql**.*

The solution for this Exercise is the same as the solution to Exercise 3.60. See the Instructor's Manual for Chapter 3 and the solutions in the file:

DBC-e08-ODB-WP-SQL-Queries-CH03-Exercises.sql

*B.28 If you have not already done so, import the **COMPUTER** table from Microsoft Excel into the Oracle Database XE WP database as explained in the text.*

This is self-explanatory. Follow the instructions on pages B-34 through B-45. This exercise is intended to make sure that the student is prepared to answer exercises B.29 and B.30.

*B.29 Import the **COMPUTER_ASSIGNMENT** table from Microsoft Excel into the Oracle Database XE WP database as explained in the text. How should this table be linked to the EMPLOYEE table and the COMPUTER table by foreign keys? Be sure to include these foreign keys in your final COMPUTER_ASSIGNMENT table structure when you create it in Oracle Database XE. Note that this may require using some ALTER TABLE statements as discussed in Appendix E.*

We will first import the table then create foreign keys in the new table linking it to COMPUTER and to EMPLOYEE. We will use the following column characteristics, which reflect Access data types and are copied from Figure A-65:

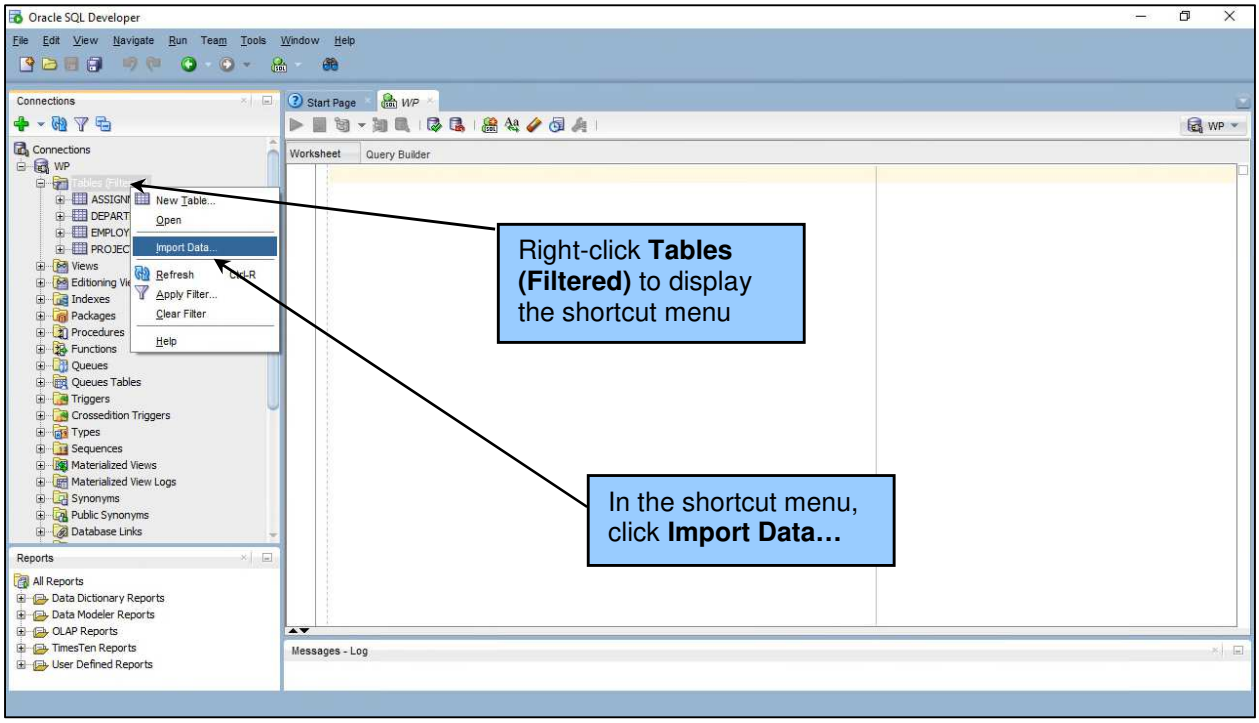
Database Column Characteristics for the WP COMPUTER_ASSIGNMENT Table

Column Name	Type	Key	Required	Remarks
SerialNumber	Number	Primary Key, Foreign Key	Yes	Long Integer
EmployeeNumber	Number	Primary Key, Foreign Key	Yes	Long Integer
DateAssigned	Date/Time	Primary Key	Yes	Medium Date

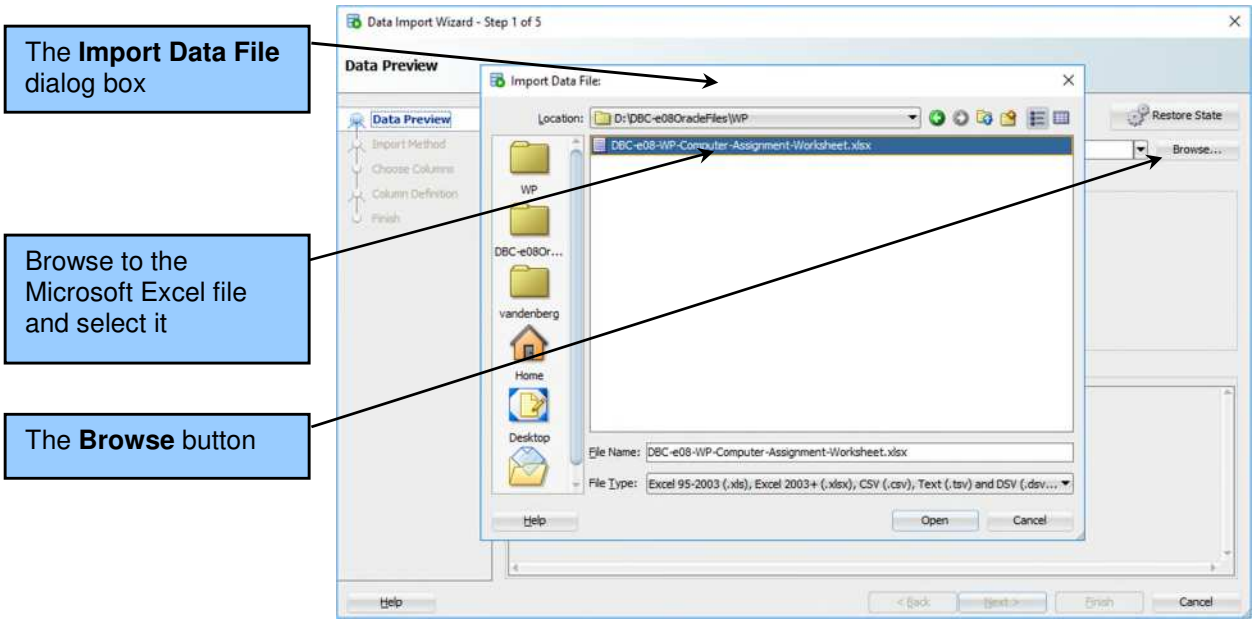
To Import the COMPUTER_ASSIGNMENT table:

1. In Oracle SQL Developer, expand the WP Database.

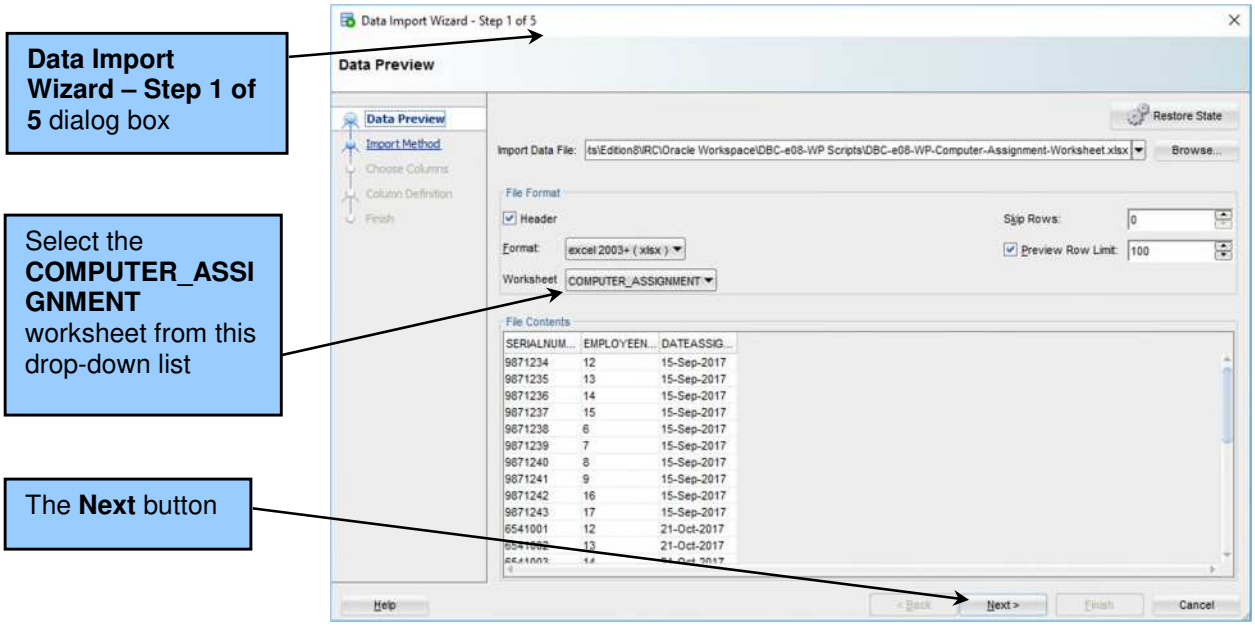
2. Right-click on the **Tables (Filtered)** WP database object to display a shortcut menu, and in the shortcut menu click on the **Import Data** command:



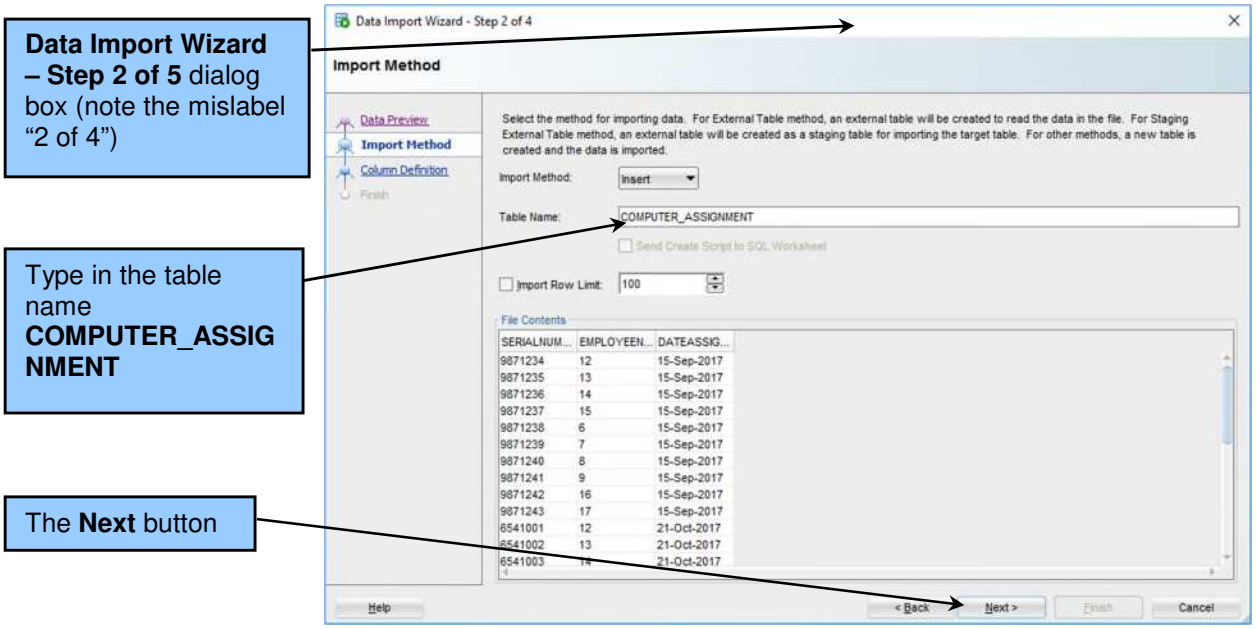
3. The *Data Import Wizard-Step 1 of 5* (Data Preview) dialog box is displayed. Use the Browse button to open the Import Data File dialog box and locate the Excel workbook containing the WP COMPUTER_ASSIGNMENT data. Click the Open button.



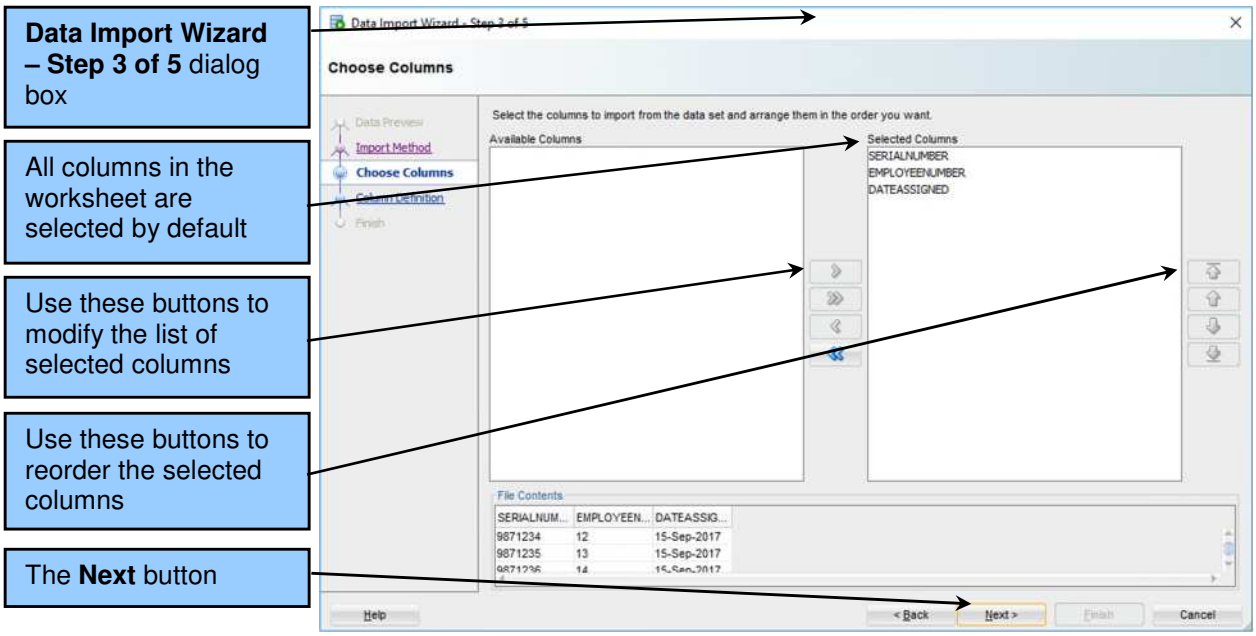
4. The Data Preview dialog box is redisplayed, this time with information from the Excel workbook. Select the COMPUTER_ASSIGNMENT worksheet from the Worksheet drop-down list as shown below. Also, make sure the Header checkbox is checked and that the Excel 2003+ (xlsx) format is selected from the Format drop-down list.



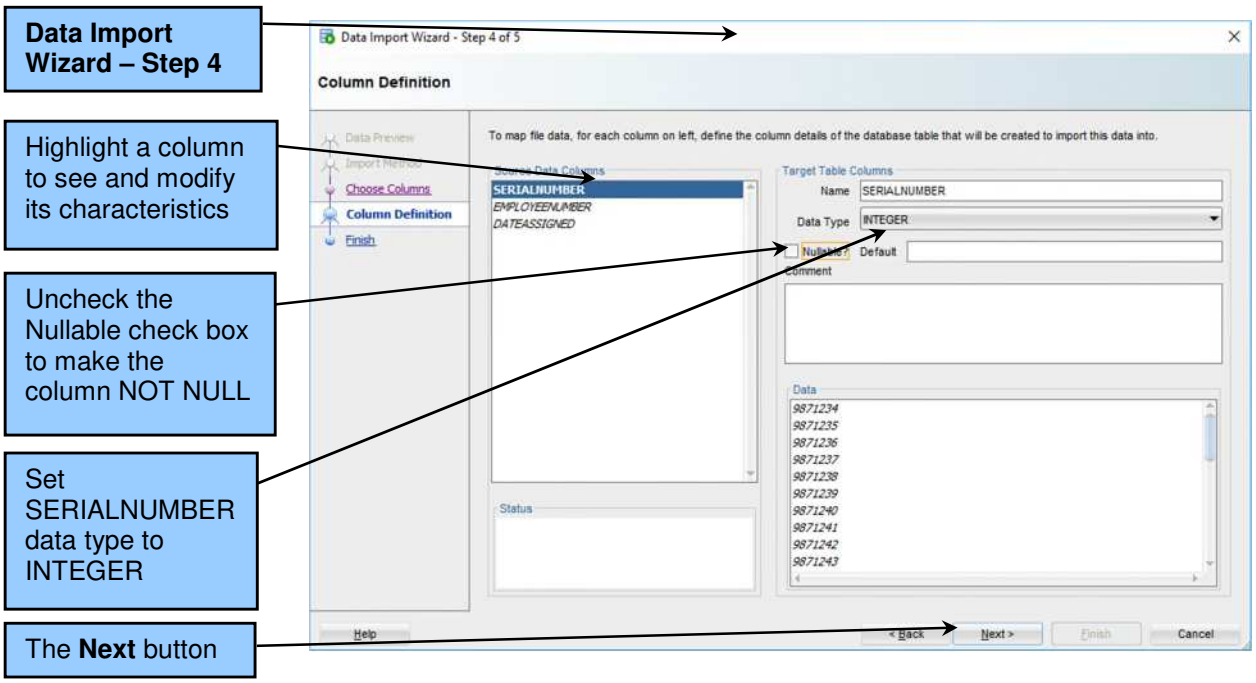
5. Click the Next button as shown above. The *Data Import Wizard – Step 2 of 5* (Import Method) dialog box is displayed (and mislabeled as *Step 2 of 4*). Type in the Table Name COMPUTER_ASSIGNMENT and leave the rest of the settings as they are:



6. Click the Next button as shown above. The *Data Import Wizard – Step 3 of 5* (Choose Columns) dialog box is displayed as shown below. This step allows us to choose which spreadsheet columns to import. Note that all are currently selected, and that is what we want, so no changes are made.



7. Click the Next button as shown above to display the *Data Import Wizard – Step 4 of 5* (Column Definition) dialog box as shown below. Here we can select column characteristics for the COMPUTER_ASSIGNMENT table. Note that in the figure we show the proper settings for the SERIALNUMBER column (which will become part of the primary key of the COMPUTER_ASSIGNMENT table). In particular, we set the column to INTEGER and NOT NULL.



8. Complete the specifications for the EmployeeNumber and DateAssigned columns as follows: EmployeeNumber, like SerialNumber, should be type INTEGER and not nullable. DateAssigned should be imported as VARCHAR2(11). If desired, it can be transformed after import into an Oracle DATE datatype, using the TO_DATE function and other techniques covered in this book (that step is not shown in these solutions). DateAssigned, which will also be part of the primary key, should also not be nullable.

9. The *Data Import Wizard – Step 5 of 5* (Finish) dialog box is displayed (not shown here). This window allows you to examine the choices made during the import dialogs and to save the settings to a file for future use (perhaps with another, similar table or spreadsheet). Click the Finish button.

10. The Import Data results box appears (not shown here), indicating successful completion of the importing. Click OK to complete the import process.

11. In the WP SQL Worksheet window, type and run the following SQL query to verify the proper importing of the COMPUTER_ASSIGNMENT data:

```
SELECT   *  FROM  COMPUTER_ASSIGNMENT;
```

12. Next we need to create the foreign keys using SQL ALTER TABLE statements. Note that we link to both the EMPLOYEE table using EmployeeNumber and to the COMPUTER table using SerialNumber. What referential integrity constraints should we use? For EMPLOYEE, EmployeeNumber is a surrogate key and is never updated. Further, WP never drops employee records because of record keeping requirements. Therefore, we will never cascade updates or deletes for this primary key. FOR COMPUTER, SerialNumber never changes. However, when we remove a computer from the WP computer inventory, we do not need to keep historical records of having had that computer. Therefore, while we do not cascade updates, we will cascade deletions for this primary key. The SQL for this is covered in Appendix E in more detail:

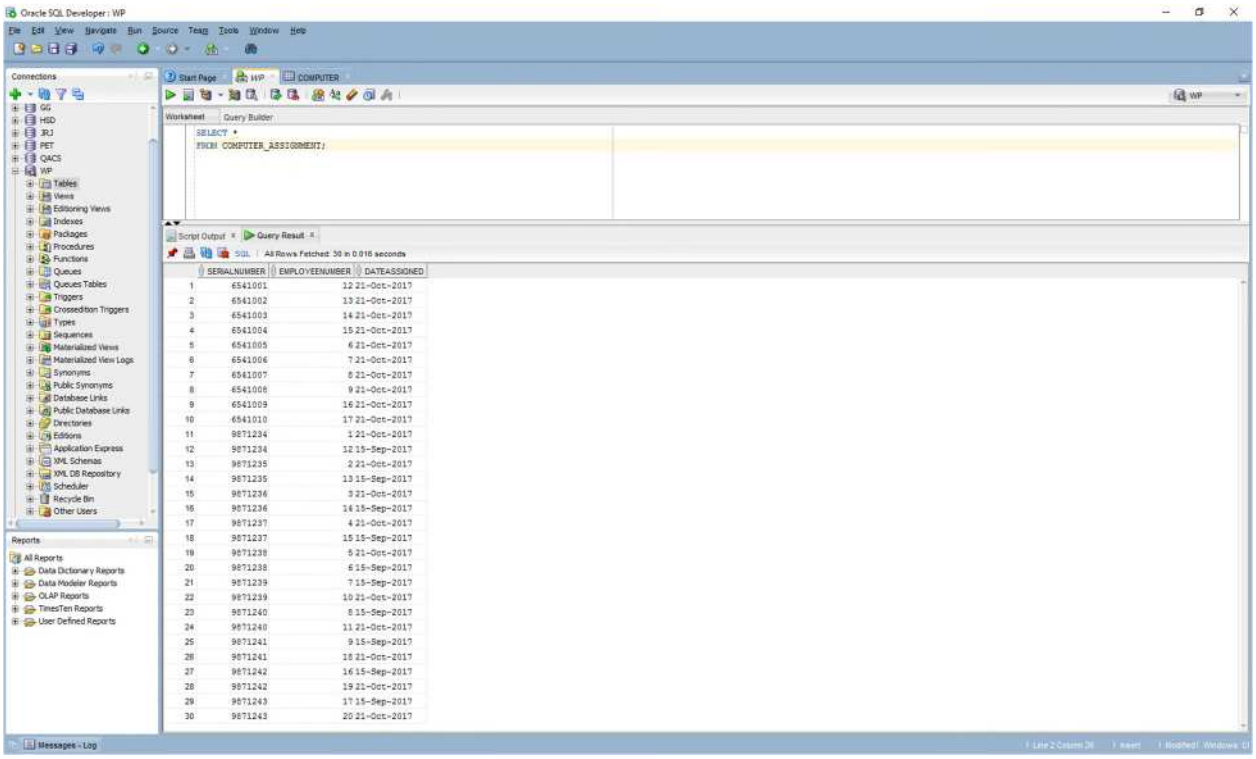
```
ALTER TABLE COMPUTER_ASSIGNMENT
  ADD CONSTRAINT CA_COMP_FK FOREIGN KEY (SerialNumber)
    REFERENCES COMPUTER (SerialNumber)
    ON DELETE CASCADE;

ALTER TABLE COMPUTER_ASSIGNMENT
  ADD CONSTRAINT CA_EMP_FK FOREIGN KEY (EmployeeNumber)
    REFERENCES EMPLOYEE (EmployeeNumber);
```

13. The one remaining task for the COMPUTER_ASSIGNMENT table is to set the primary key. The ALTER TABLE syntax for accomplishing this will be covered in Appendix E, but here is the solution:

```
ALTER TABLE COMPUTER_ASSIGNMENT
  ADD CONSTRAINT CA_PK PRIMARY KEY
      (SerialNumber, EmployeeNumber, DateAssigned);
```

14. We can verify correct importing by running an SQL query, whose results are shown below:

```
SELECT *
FROM  COMPUTER_ASSIGNMENT;
```



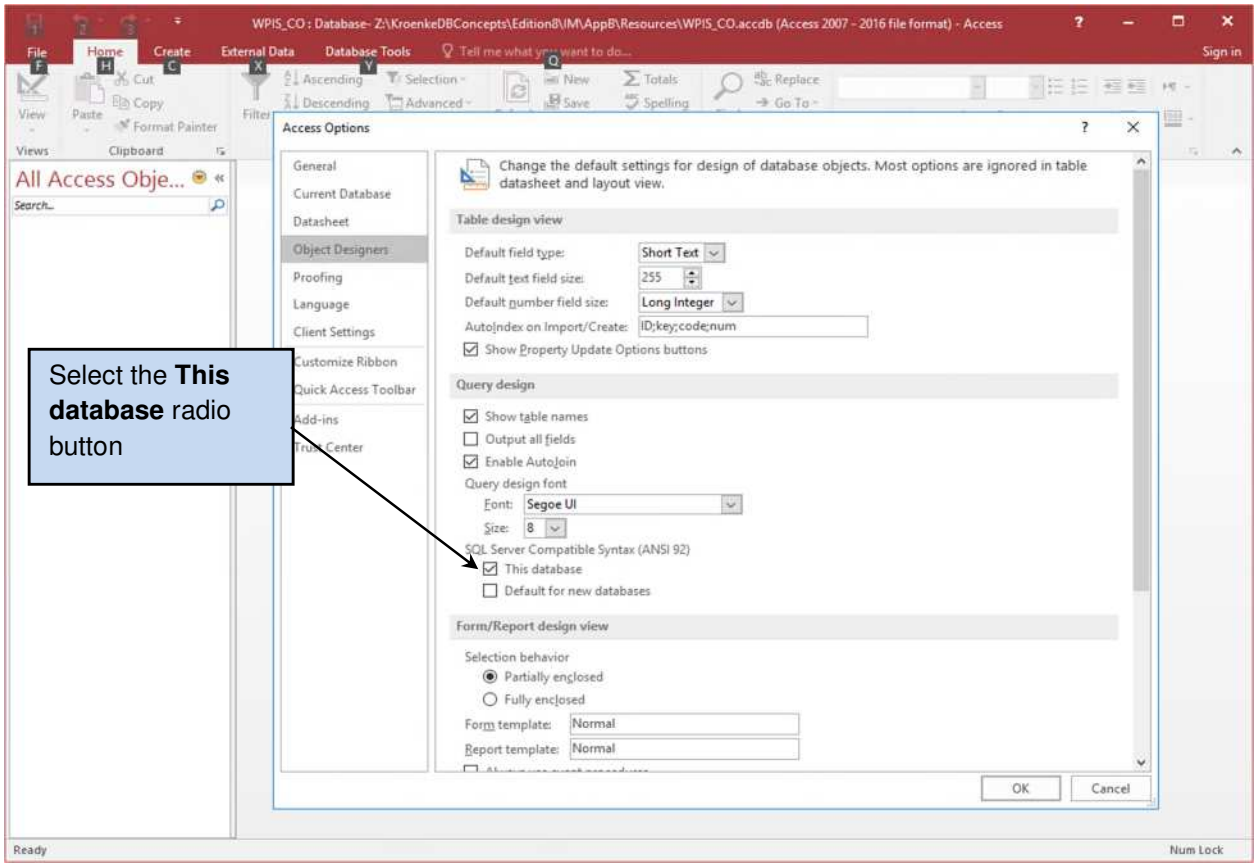
*B.30 Create a Microsoft Access 2016 database named **WPIS_CO.accdb** where CO stands for "computer-only." This database will be an application for the Oracle Database XE WP database which will allow users to read, query, and alter the data in the WP database COMPUTER table but will provide no access to other WP database tables. Once you have created the database:*

1. *If you have not completed exercise B.28, do so now.*

   This is self-explanatory.

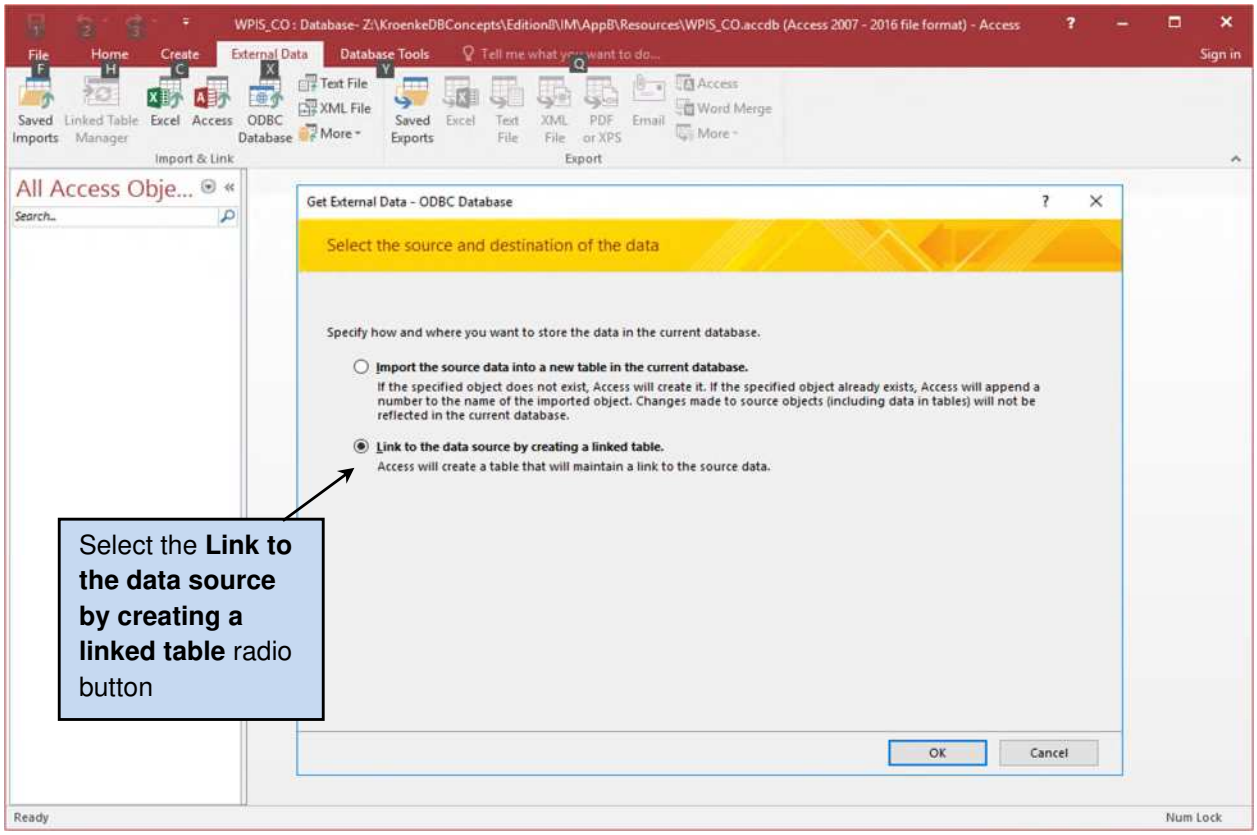
2. *Set the WPIS_CO.accdb database to use **SQL Server Compatible Syntax (ANSI 92)**.*

   See the image on the next page.



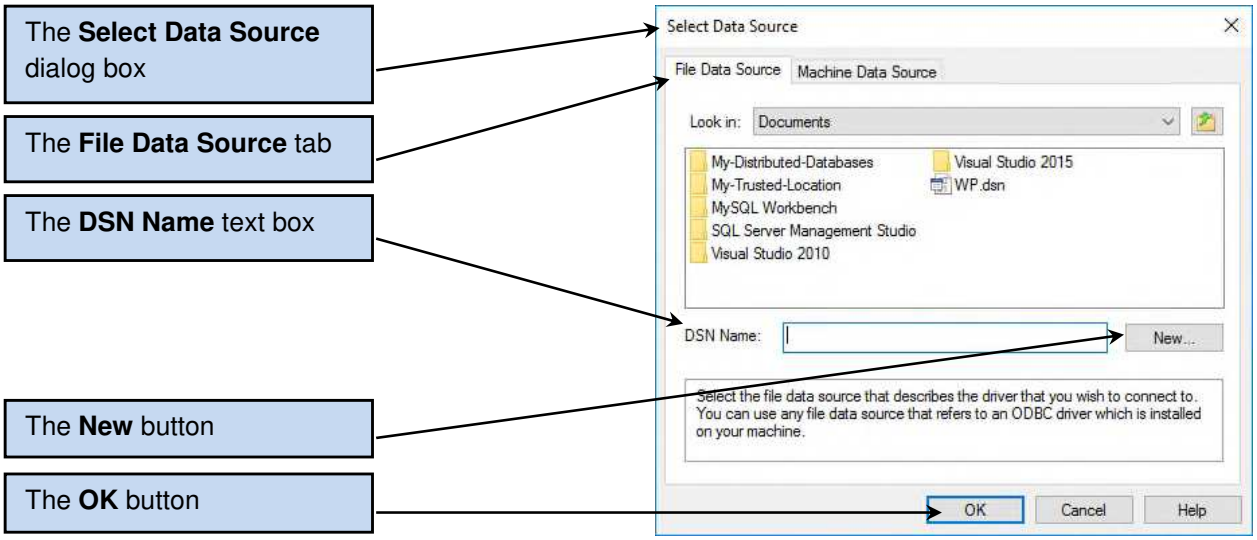
*3. Link the WPIS_CO.accdb database to the Oracle Database XE WP database using ODBC. When you create your File Data Source DSN, name it **WPCO**, and use the same WP_USER account for Oracle Database XE authentication.*

1. In the Microsoft Access 2016 WPIS.accdb database, click the **External Data** command tab, and then click the **ODBC Database** button in the Import & Link commands section.
2. The **Get External Data - ODBC Database** Wizard dialog box is displayed.
3. In the **Get External Data – ODBC Database Wizard** dialog box, the **Select the source and destination of the data** page is displayed. Click the **Link to the data source by creating a linked table** radio button, as shown in the screen shot on the next page.

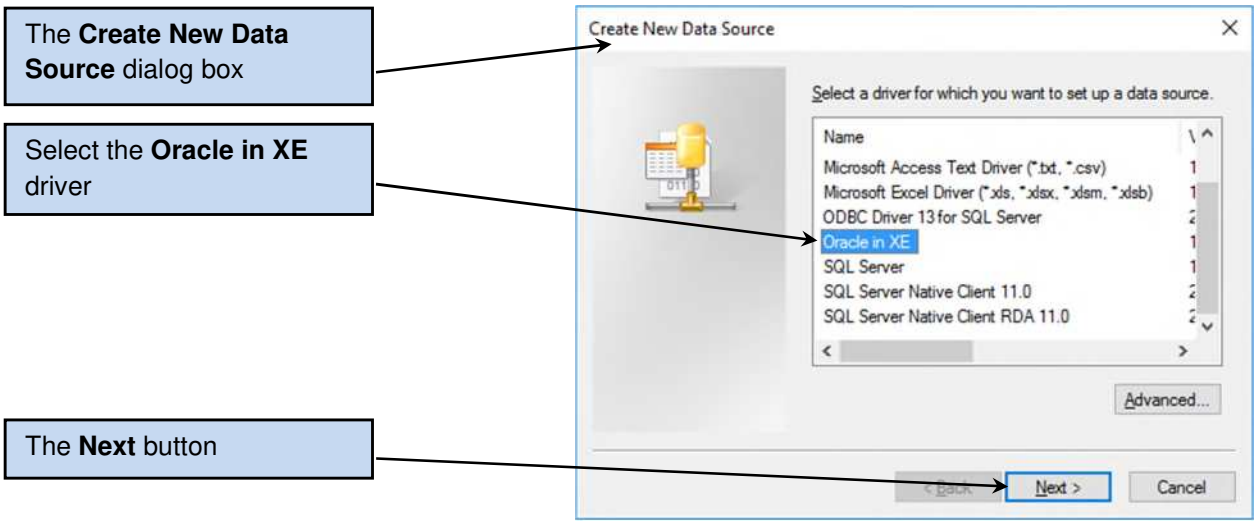


4. Click the **OK** button.

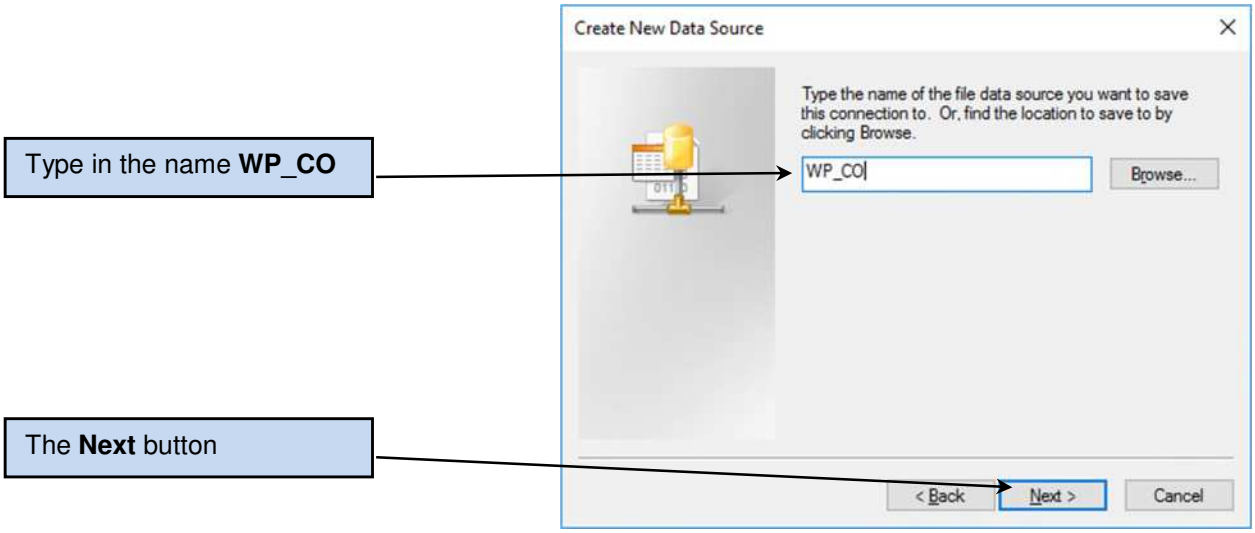
5. The **Select Data Source** dialog box is displayed. This is the dialog box that we will use to create the needed OBDC DSN. In the Select Data Source dialog box, make sure the **File Data Source** tab is selected, as shown below.



6. Click the **New** button to display the **Create New Data Source** dialog box.
7. In the Create New Data Source dialog box, scroll down through the list of drivers until you can see the driver named **Oracle in XE**. Click this driver name to select it.

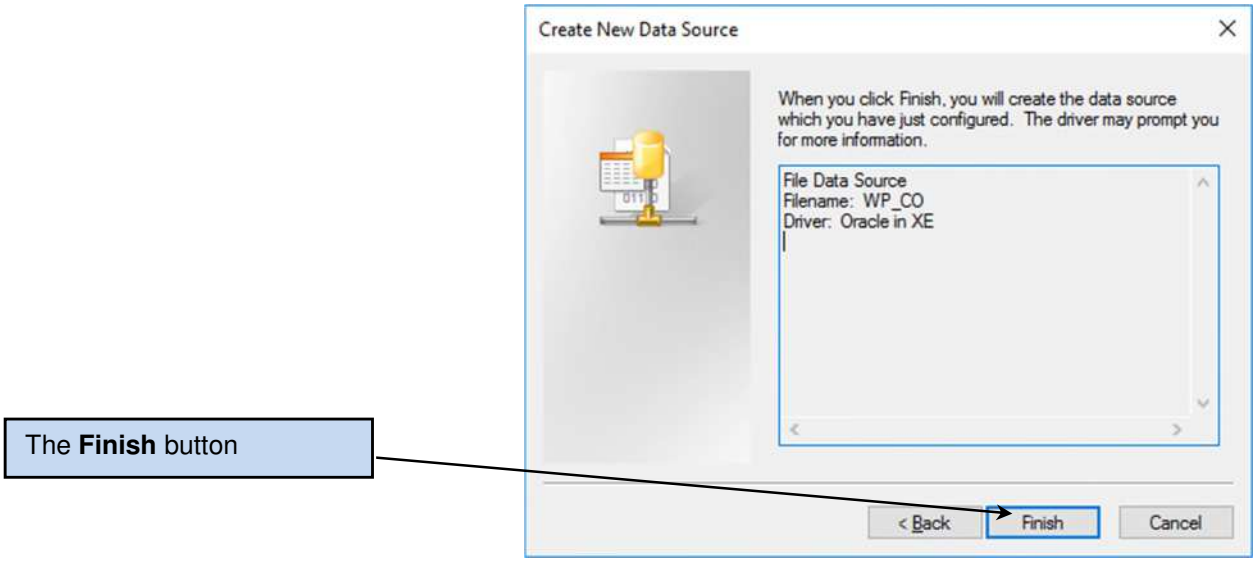


8. Click the **Next** button.
9. The next page of the Create New Data Source dialog box provides a text box for naming the new DSN. Type in **WP_CO**.



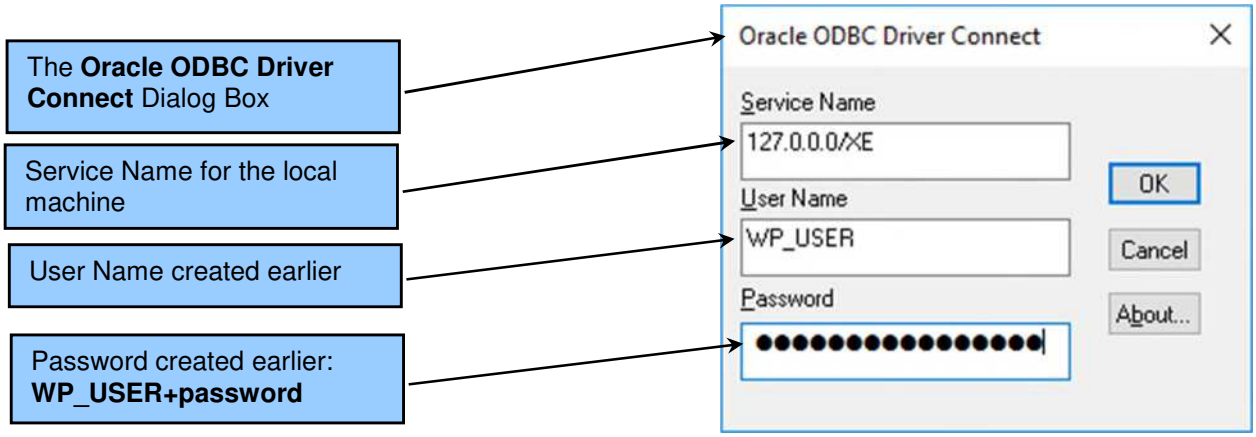
10. Click the **Next** button.

11. The next page of the **Create New Data Source** dialog box provides a summary of the settings that will be used to create the new DSN.

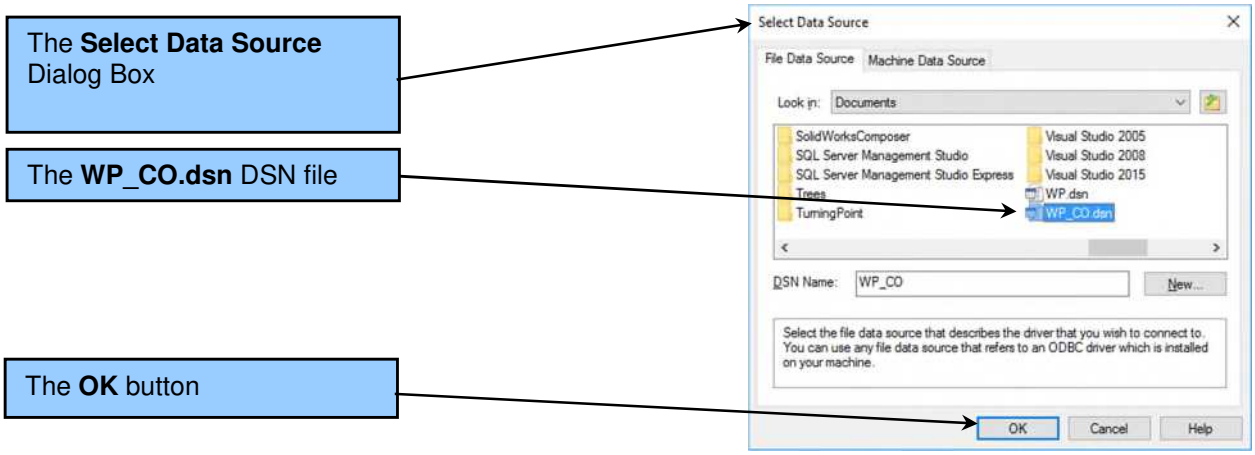


10. Click the **Finish** button.

11. The **Oracle ODBC Driver Connect** dialog box is displayed, as shown below. Enter the **Service Name**, **User Name**, and **Password** as shown in the figure (the User Name and Password were created earlier in this appendix). Click the **OK** button.



12. The WP DSN file data source is created and displayed in the Select Data Source dialog box as shown on the next page. Now that the DSN is completed, we can finish linking the Microsoft Access 2016 database to the Oracle Database XE database.



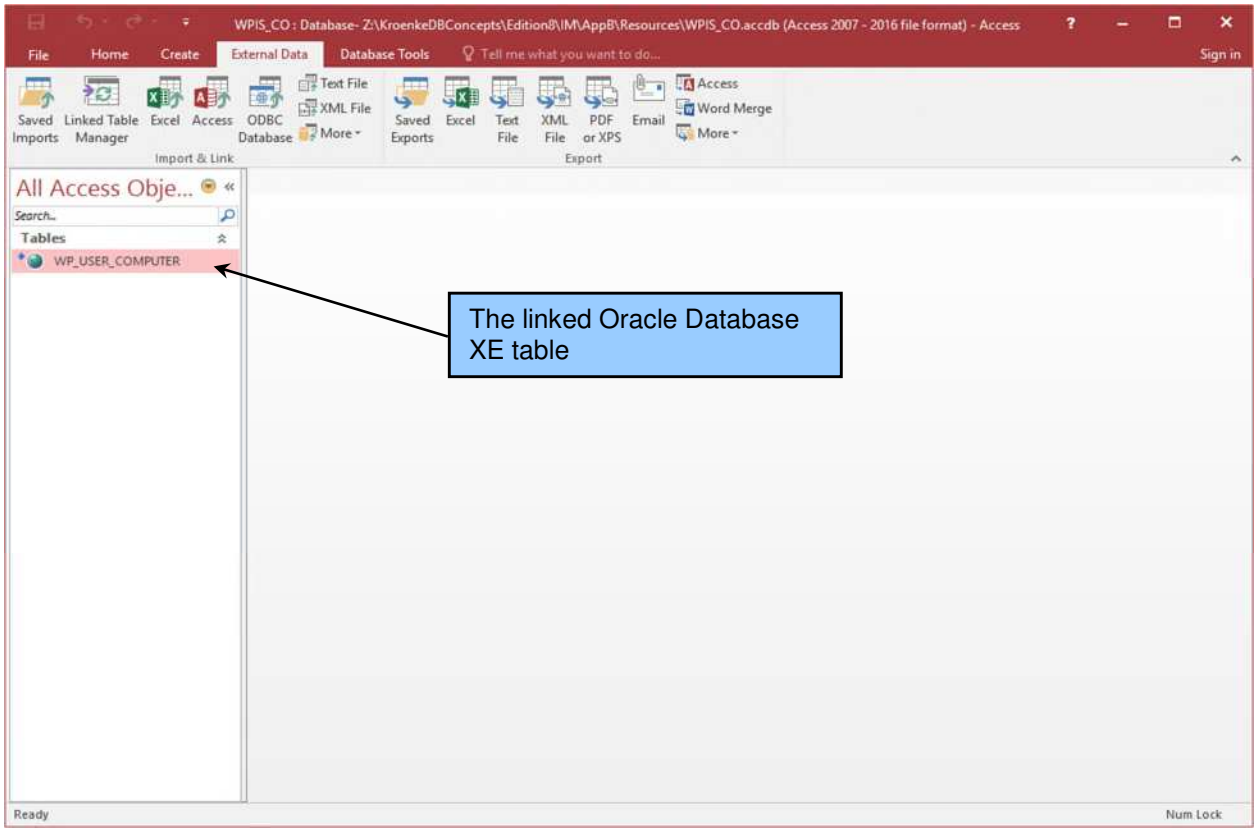
13. In the **Select Data Source** dialog box, click the **OK** button. The **Oracle ODBC Driver Connect** dialog box is displayed again, as shown on the previous page. Enter the password **WP_USER+password** in the **Password** text box, and then click the **OK** button.

4. *Import the COMPUTER table. You may be asked by Access to specify a primary key; if so, use SERIALNUMBER.*

Note that the steps in this question are a continuation of the steps in question 3—the Microsoft Access 2016 Get External Data – ODBC Database Wizard continues to run after we close the Select Data Source dialog box!

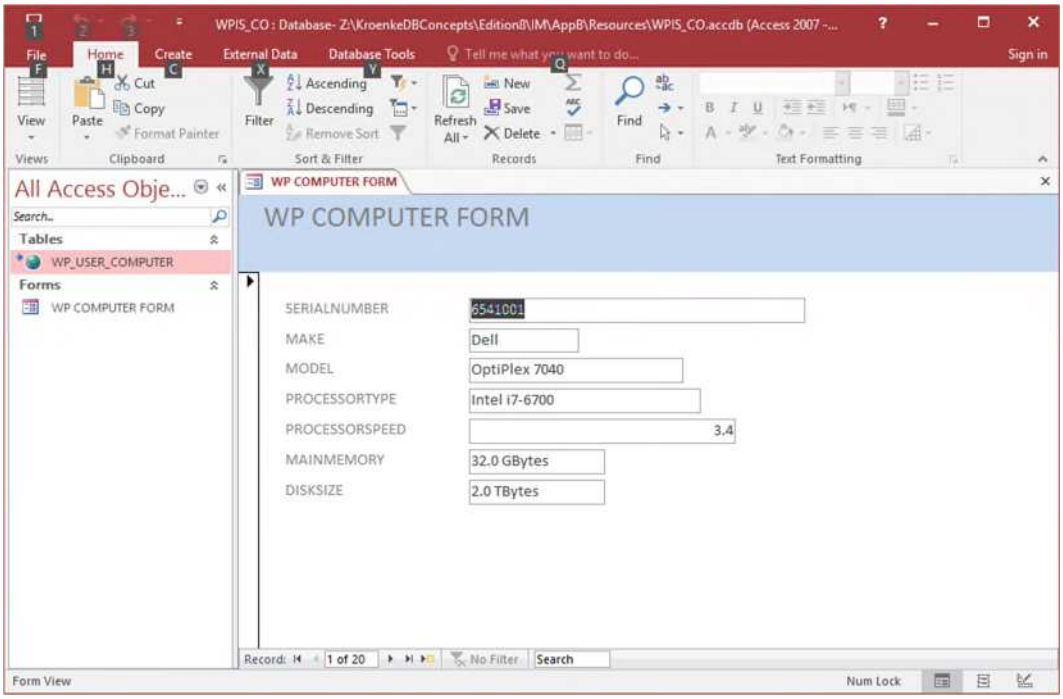
1. The **Link Tables** dialog box is displayed, as shown below. Scroll down until you find the **WP_USER.COMPUTER** table. To select this table, **click** on it.

2. Click the **OK** button. The ODBC links between the WP.accdb Microsoft Access 2016 database and the WP database in Oracle Database XE are completed, as shown below.



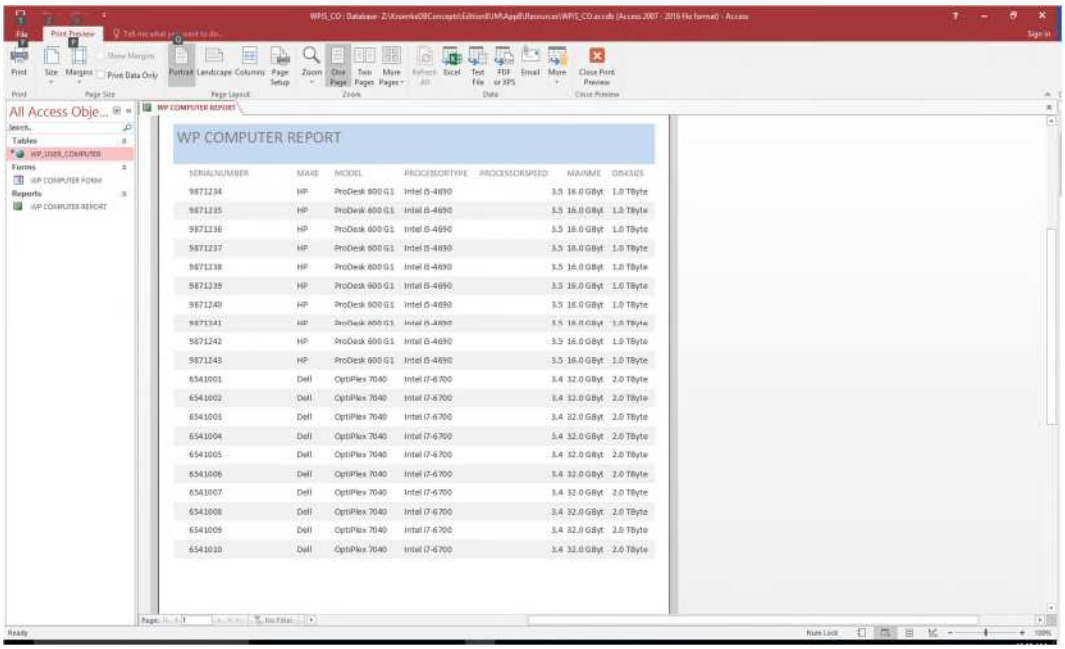
5. *Create a form to show all the data in the COMPUTER table named **WP Computer Form**.*

This is done using Microsoft Access 2016 techniques as discussed in the Chapter 1 section of "The Access Workbench." An image of the form is on the next page.



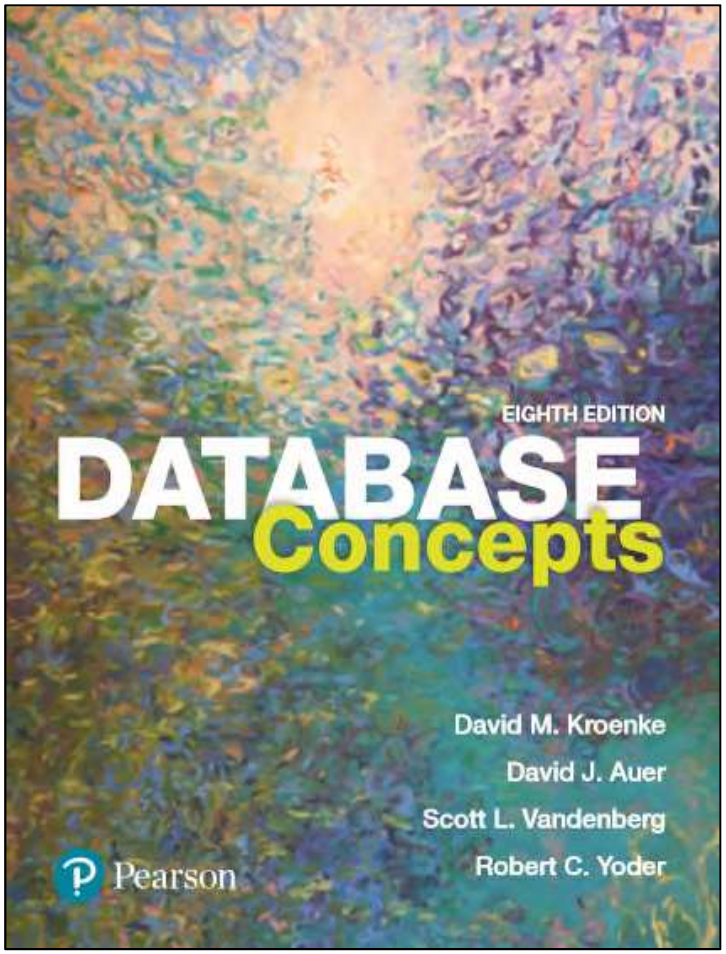
6. *Create a report to show all the data in the COMPUTER table named* ***WP Computer Report****.*

This is done using Microsoft Access 2016 techniques as discussed in the Chapter 4 section of "The Access Workbench." An image of the report is shown below.



Database Concepts

Eighth Edition



Chapter 2

The Relational Model

Chapter Objectives

2.1 Learn the conceptual foundation of the relational model

2.2 Understand how relations differ from nonrelational tables

2.3 Learn basic relational terminology

2.4 Learn the meaning and importance of keys, foreign keys, and related terminology

2.5 Understand how foreign keys represent relationships

2.6 Learn the purpose and use of surrogate keys

2.7 Learn the meaning of functional dependencies

2.8 Learn to apply a process for normalizing relations

Entity

- An entity is something of importance to a user that needs to be represented in a database.
- An entity represents one theme or topic.
- In an entity-relationship model (discussed in Chapter 4), entities are restricted to things that can be represented by a single table.

Relation

- A **relation** is a two-dimensional table that has specific characteristics.
- The table dimensions, like a matrix, consist of rows and columns

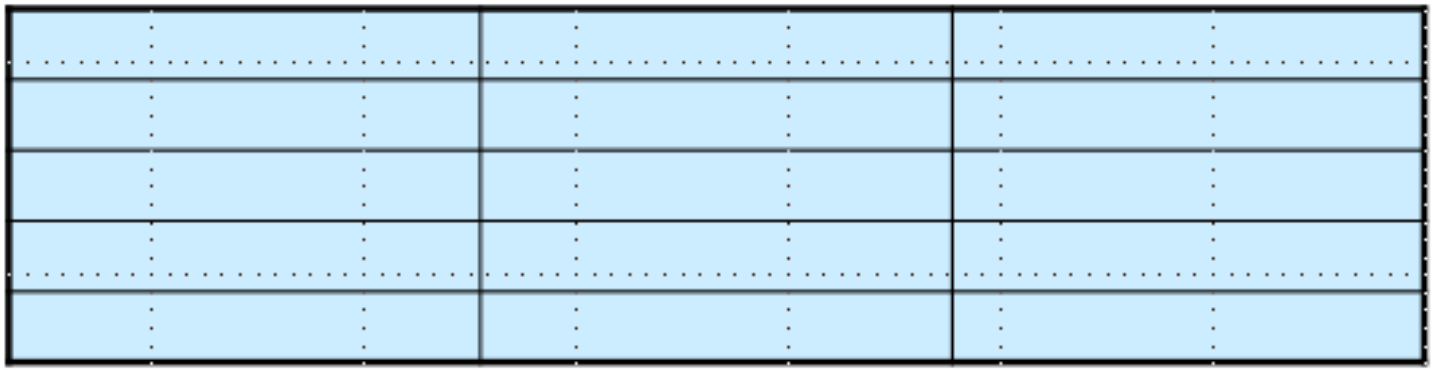


Figure 2-1 Characteristics of a Relation

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

A Sample Relation

Figure 2-2 Sample EMPLOYEE Relation

Employee Number	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420

Nonrelation Example 1

Figure 2-3 Nonrelational Table—Multiple Entries per Cell

Cells of the table hold multiple values

Employee Number	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310, 360-285-8391, 206-723-8392
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420

Nonrelation Example 2

Figure 2-4 Nonrelational Table—Order of Rows Matters and Kind of Column Entries Differs in Email

Employee Number	FirstName	LastName	Department	EmailAddress	Phone
101	Mary	Jacobs	Administration	Mary.Jacobs@ourcompany.com	360-285-8110
102	Rosalie	Jackson	Administration	Rosalie.Jackson@ourcompany.com	360-285-8120
103	Richard	Bandalone	Legal	Richard.Bandalone@ourcompany.com	360-285-8210
104	George	Smith	Human Resources	George.Smith@ourcompany.com	360-285-8310
				Fax:	360-285-8391
				Home:	206-723-8392
105	Alan	Adams	Human Resources	Alan.Adams@ourcompany.com	360-285-8320
106	Ken	Evans	Finance	Ken.Evans@ourcompany.com	360-285-8410
107	Mary	Abernathy	Finance	Mary.Abernathy@ourcompany.com	360-285-8420
				Fax:	360-285-8491

Terminology

Figure 2-6 Equivalent Sets of Terms

Table	Row	Column
File	Record	Field
Relation	Tuple	Attribute

A Key

- A **key** is one or more columns of a relation that is used to identify a row.
- A key can be unique (primary key) or nonunique (foreign key).

A Composite Key

- A **composite key** is a key that contains two or more attributes.
- For a key to be a unique identifier, it must often become a composite key.

Candidate and Primary Keys

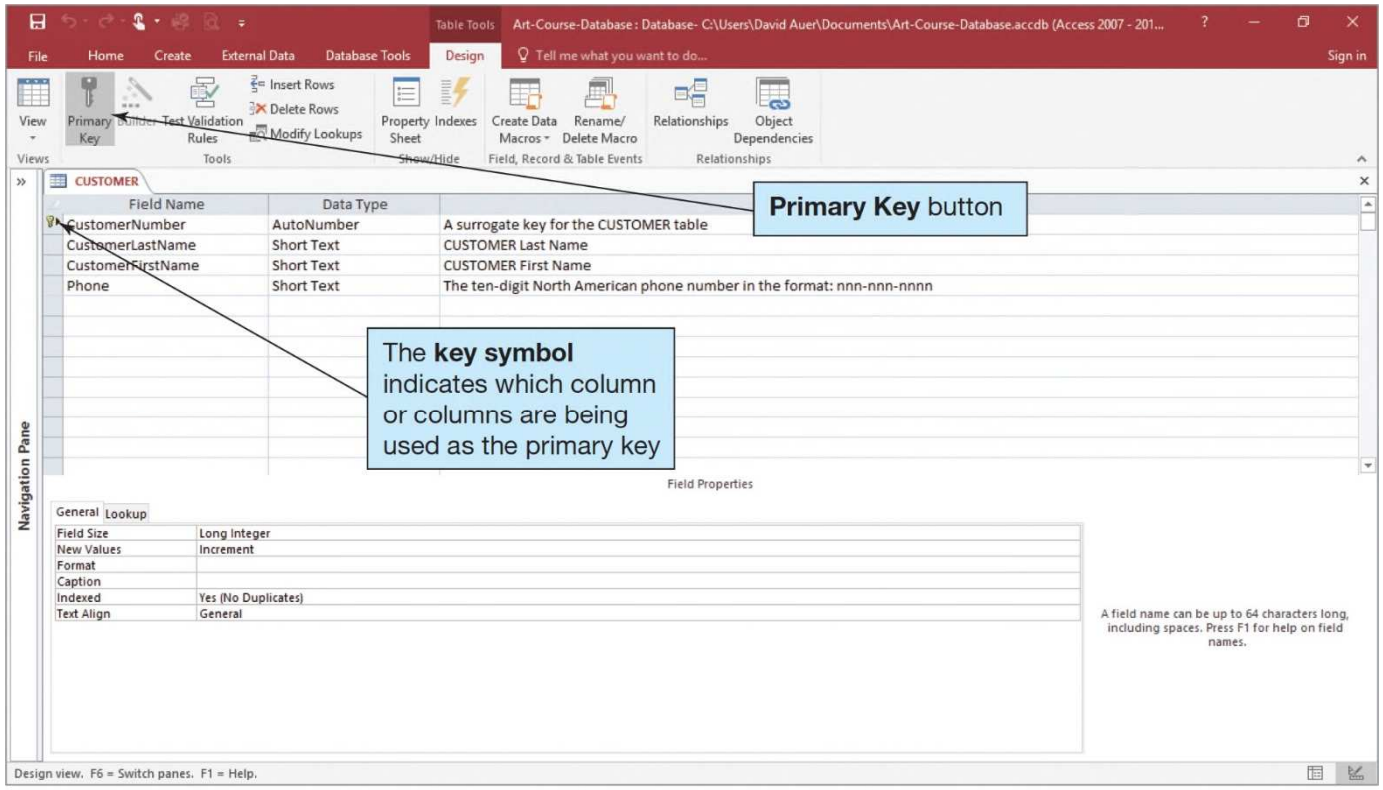
- **Candidate keys** are keys that uniquely identify each row in a relation.
- A relation can have multiple candidate keys.
- A **primary key** is a candidate key that is chosen as the key the DBMS will use to uniquely identify each row in a relation.

Primary Key Example

- We will underline the primary key(s) as an identification to indicate the unique key shown in the example below.

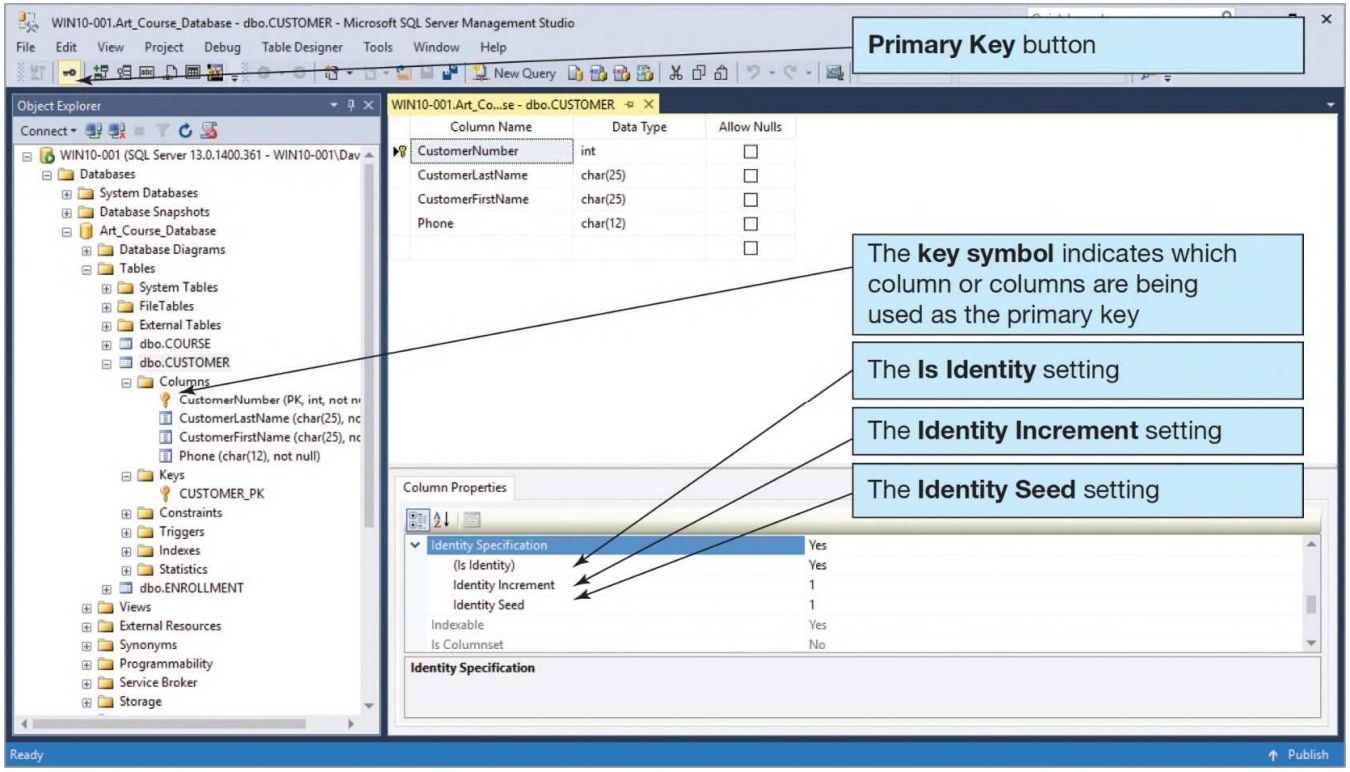
EMPLOYEE (EmployeeNumber, FirstName, LastName, Department, EmailAddress, Phone)

Figure 2-7 Defining the Primary Key in Microsoft Access 2016



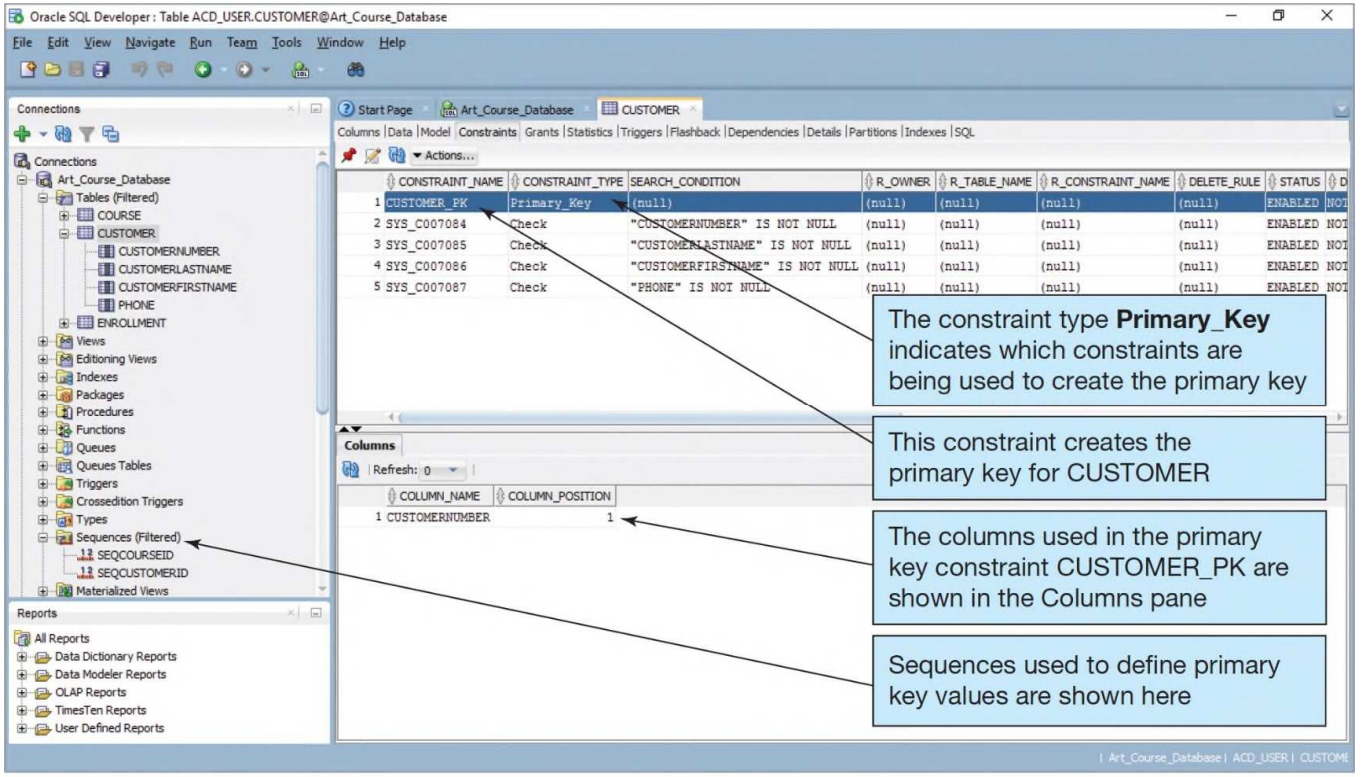
Access 2016, Windows 10, Microsoft Corporation.

Figure 2-8 Defining the Primary Key in Microsoft SQL Server 2016



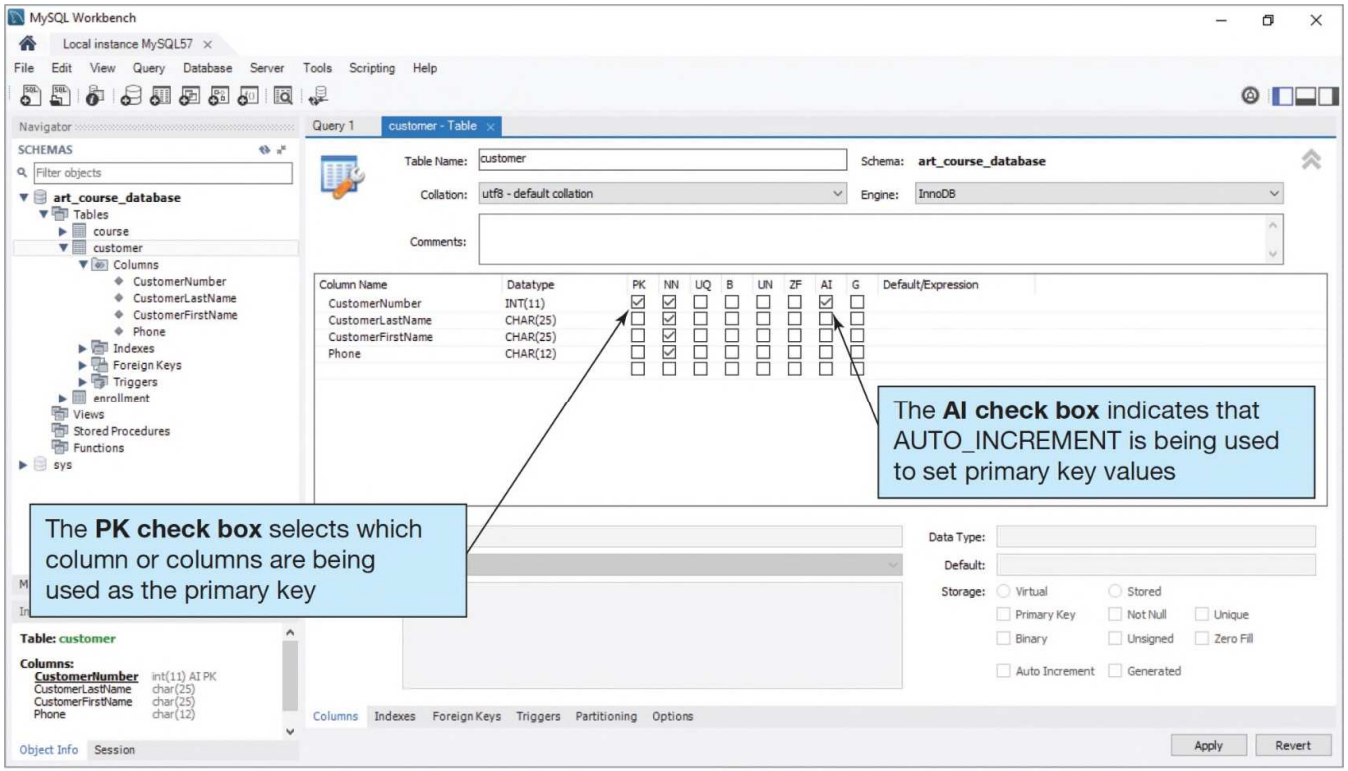
SQL Server 2016, Windows 10, Microsoft Corporation.

Figure 2-9 Defining the Primary Key in Oracle Database XE



Oracle SQL Developer 4.01, Oracle Corporation.

Figure 2-10 Defining the Primary Key in My SQL 5.7



Oracle MySQL Community Server 5.7, Oracle Corporation.

A Surrogate Key

- A **surrogate key** is a column with a unique, DBMS-assigned identifier that has been added to a table to be the primary key.
- The ideal surrogate key is short and numeric and never changes.
- Surrogate key values will have no inherent meaning to users, thus they are often hidden in forms, query results, and reports.

Surrogate Key Example

- In the example below, notice that it takes the Street, City, State, and ZIP code to serve as a composite primary key. The second example will show how the addition of PropertyID serves as the surrogate primary key.

PROPERTY (Street, City, State, ZIP, OwnerID)

PROPERTY (PropertyID, Street, City, State, ZIP, OwnerID)

A Foreign Key

- The primary key of one relation is placed into a second relation as a **foreign key** to represent a relationship.
- Foreign keys are represented in a relationship by italics as seen in the example below.

EMPLOYEE (EmployeeNumber, FirstName, LastName, *Department*, EmailAddress, Phone)

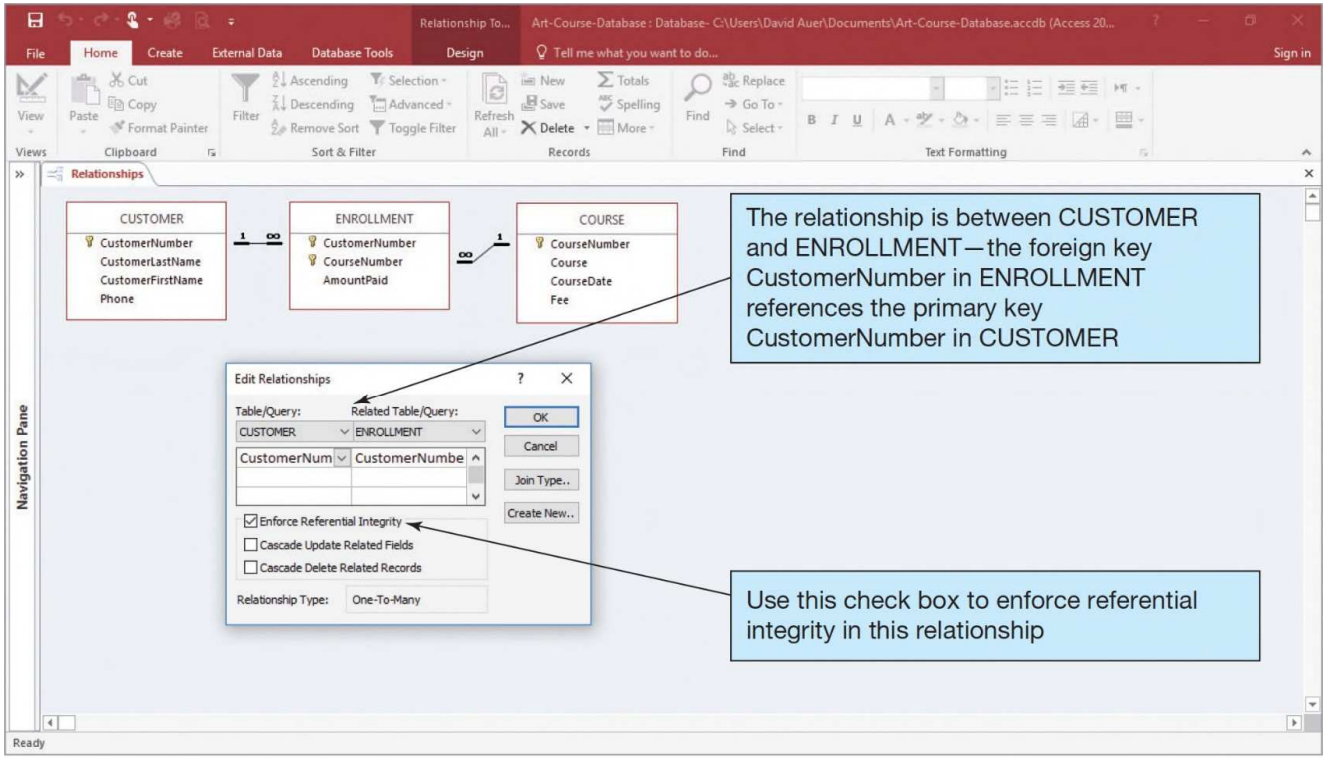
and:

DEPARTMENT (DepartmentName, BudgetCode, OfficeNumber, DepartmentPhone)

Referential Integrity

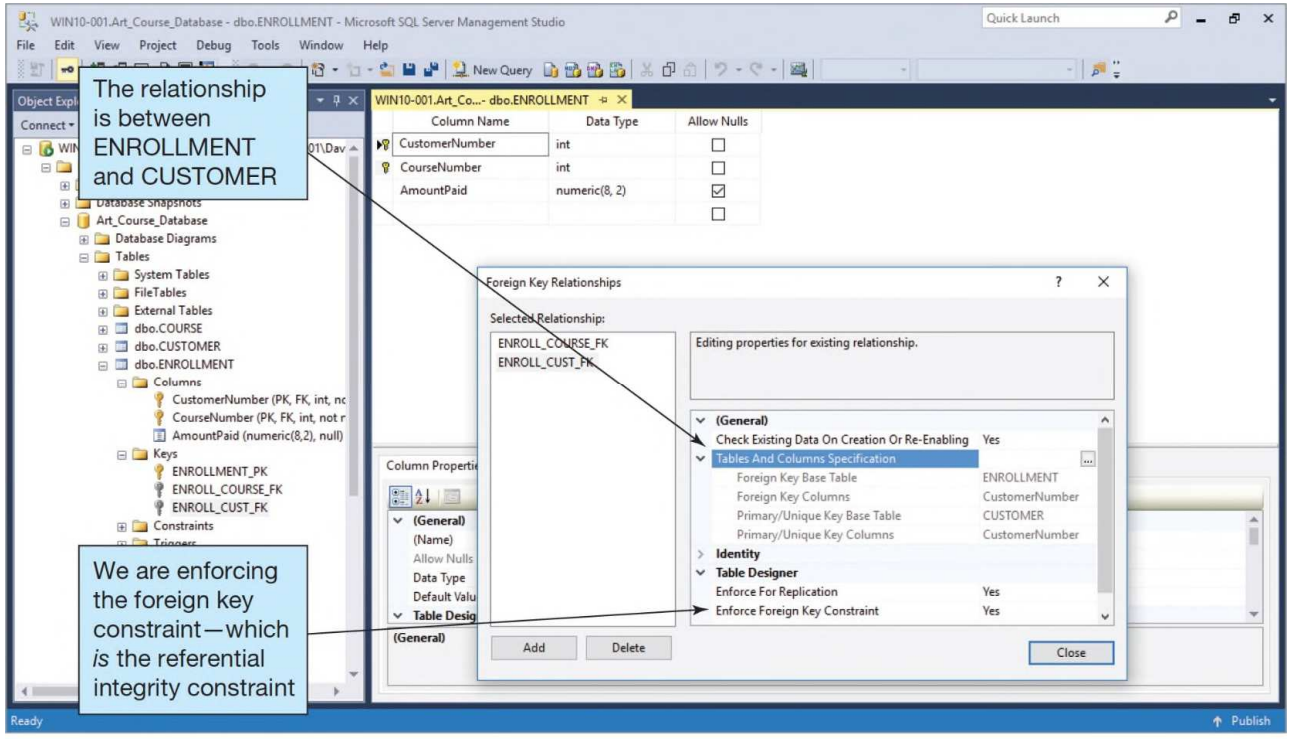
- Referential integrity states that every value of a foreign key must match a value of an existing primary key.
- In the relationship between EMPLOYEE and DEPARTMENT seen on the previous slide, the department attribute located in the EMPLOYEE table is the foreign key and whatever value is placed in that column, the same value MUST exist in the Department attribute in the DEPARTMENT table.

Figure 2-11 Enforcing Referential Integrity in Microsoft Access 2016



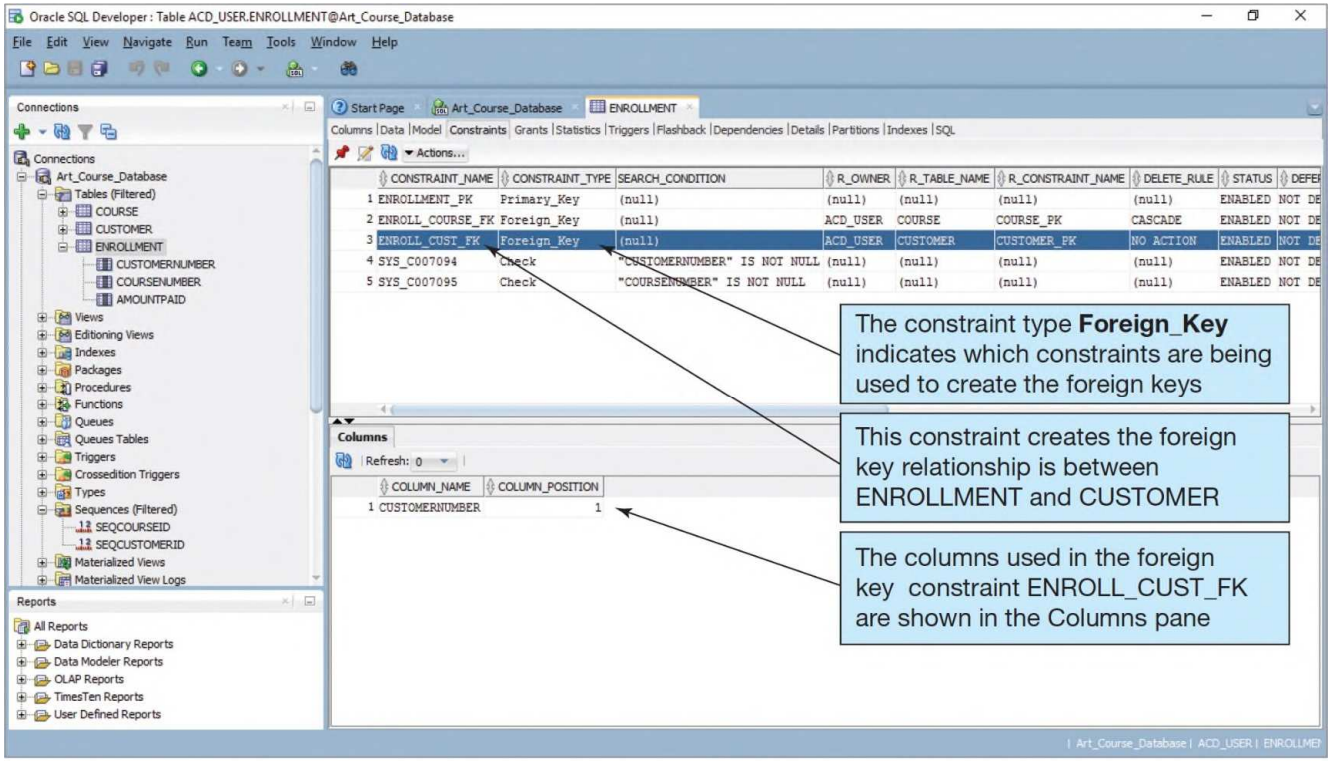
Access 2016, Windows 10, Microsoft Corporation.

Figure 2-12 Enforcing Referential Integrity in Microsoft SQL Server 2016



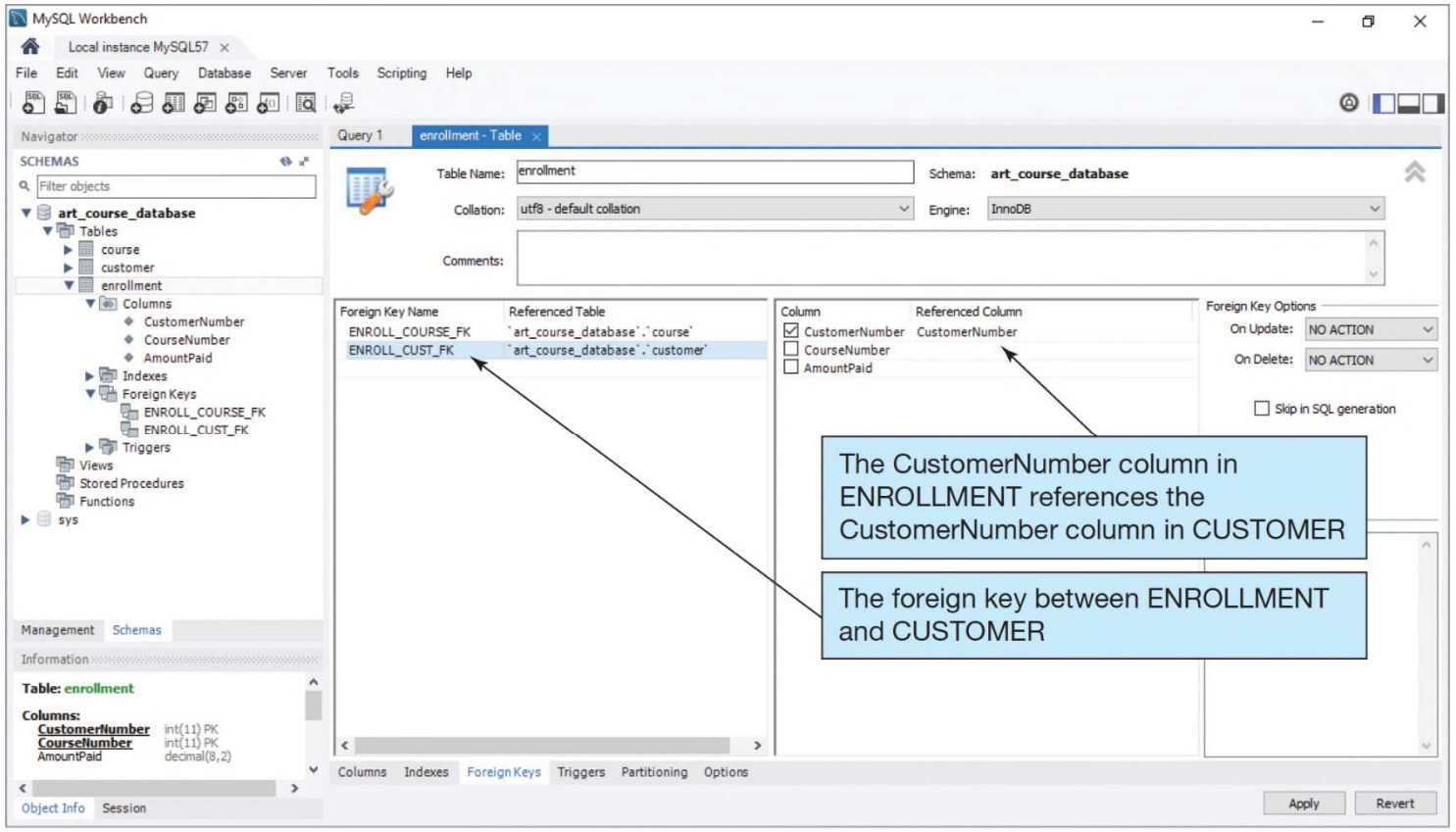
SQL Server 2016, Windows 10, Microsoft Corporation.

Figure 2-13 Enforcing Referential Integrity in Oracle Database XE



Oracle SQL Developer 4.01, Oracle Corporation.

Figure 2-14 Enforcing Referential Integrity in My SQL 5.7



Oracle MySQL Community Server 5.7, Oracle Corporation.

The Null Value

- A **null value** is a missing value in a cell in a relation.
- This is different from a zero, space, character, or tab character.
- You can eliminate null values by requiring an attribute value.
- DBMS products allow you to specify whether a null value can occur in a column.

The Problem of Null Values

- A null value is often ambiguous. It could mean:
  - the column value is not appropriate for the specific row
  - the column value is not decided
  - the column value is unknown

Determinants

- Suppose the cost of cookies is determined by the number of boxes we buy. We would say that the number of boxes would be a **determinant** because it determines the cost of the cookies. The determinant is always shown on the left side of the relationship as seen below.

NumberOfBoxes → CookieCost

Determinants and Functional Dependencies

Object Color	Weight	Shape
Red	5	Ball
Blue	5	Cube
Yellow	7	Cube

If we know the object color (determinant) above then we know the weight and shape of the object. This would be written as a functional dependency as shown below:

Figure 2-16 Sample OBJECT Relation and Data

$$\text{ObjectColor} \rightarrow (\text{Weight, Shape})$$

Candidate/Primary Keys and Functional Dependency

- A candidate key of a relation will functionally determine all other attributes in the row.
- Likewise, a primary key of a relation will functionally determine all other attributes in the row.

Normalization

- **Normalization** is the process of (or set of steps for) breaking a table or relation with more than one theme into a set of tables such that each has only one theme.
- Relational design principles for normalized relations:
  - to be a well formed relation, every determinant must be a candidate key
  - any relation that is not well formed should be broken into two or more relations that are well formed

Normalization Process (1 of 2)

- Steps in the normalization process are as follows:

1. Identify all the candidate keys of the relation.
2. Identify all the functional dependencies in the relation.
3. Examine the determinants of the functional dependencies. If any determinant is not a candidate key, the relation is not well formed. In this case:
    a. Place the columns of the functional dependency in a new relation of their own.
    b. Make the determinant of the functional dependency the primary key of the new relation.

Normalization Process (2 of 2)

c. Leave a copy of the determinant as a foreign key in the original relation.
d. Create a referential integrity constraint between the original and the new relation.

4. Repeat Step 3 until every determinant of every relation is a candidate key.

Normalization Example PRESCRIPTION Relation 1

Figure 2-17 Sample PRESCRIPTION Relation and Data

Prescription Number	Date	Drug	Dosage	CustomerName	CustomerPhone	CustomerEmailAddress
P10001	10/17/2017	DrugA	10mg	Smith, Alvin	575-523-2233	ASmith@somewhere.com
P10003	10/17/2017	DrugB	35mg	Rhodes, Jeff	575-645-3455	JRhodes@somewhere.com
P10004	10/17/2017	DrugA	20mg	Smith, Sarah	575-523-2233	SSmith@somewhere.com
P10007	10/18/2017	DrugC	20mg	Frye, Michael	575-645-4566	MFrye@somewhere.com
P10010	10/18/2017	DrugB	30mg	Rhodes, Jeff	575-645-3455	JRhodes@somewhere.com

- In the figure above there are two themes: prescription and customer.
- These two themes should be broken into two separate tables as seen on the next slide.

Normalization Example PRESCRIPTION Relation 2

Note that the CustomerEmailAddress is the foreign key in the PRESCRIPTION table below.

Figure 2-18 Normalized **Customer** and **Prescription** Relations and Data

CustomerEmailAddress	CustomerName	CustomerPhone
ASmith@somewhere.com	Smith, Alvin	575-523-2233
JRhodes@somewhere.com	Rhodes, Jeff	575-645-3455
MFrye@somewhere.com	Frye, Michael	575-645-4566
SSmith@somewhere.com	Smith, Sarah	575-523-2233

PrescriptionNumber	Date	Drug	Dosage	CustomerEmailAddress
P10001	10/17/2017	DrugA	10mg	ASmith@somewhere.com
P10003	10/17/2017	DrugB	35mg	JRhodes@somewhere.com
P10004	10/17/2017	DrugA	20mg	SSmith@somewhere.com
P10007	10/18/2017	DrugC	20mg	MFrye@somewhere.com
P10010	10/18/2017	DrugB	30mg	JRhodes@somewhere.com

Normalization Example STU_DORM Relation 1

- Notice the two different themes in the relation above. Once they are split out using normalization you will see the correct tables in the next slide.

Figure 2-19 Sample STU_DORM Relation and Data

StudentNumber	LastName	FirstName	DormName	DormCost
100	Smith	Terry	Stephens	3,500.00
200	Johnson	Jeff	Alexander	3,800.00
300	Abernathy	Susan	Horan	4,000.00
400	Smith	Susan	Alexander	3,800.00
500	Wilcox	John	Stephens	3,500.00
600	Webber	Carl	Horan	4,000.00
700	Simon	Carol	Stephens	3,500.00

Normalization Example STU_DORM Relation 2

Figure 2-20 Normalized **Student** and **Dorm** Relations and Data

StudentNumber	LastName	FirstName	DormName
100	Smith	Terry	Stephens
200	Johnson	Jeff	Alexander
300	Abernathy	Susan	Horan
400	Smith	Susan	Alexander
500	Wilcox	John	Stephens
600	Webber	Carl	Horan
700	Simon	Carol	Stephens

DormName	DormCost
Alexander	3,800.00
Horan	4,000.00
Stephens	3,500.00

Eliminating Anomalies from Multivalued Dependencies

- When modification problems are due to functional dependencies and we then normalize relations to BCNF, we eliminate these anomalies.
- Anomalies can also arise from another kind of dependency—the multivalued dependency.
  - A multivalued dependency occurs when a determinant is matched with a particular set of values as seen below.

EmployeeName $\rightarrow\!\!\rightarrow$ EmployeeDegree

EmployeeName $\rightarrow\!\!\rightarrow$ EmployeeSibling

PartKitName $\rightarrow\!\!\rightarrow$ Part

A Multivalued Dependency Example (1 of 2)

Figure 2-25 Three Examples of Multivalued Dependencies

EMPLOYEE_DEGREE

EmployeeName	EmployeeDegree
Chau	BS
Green	BS
Green	MS
Green	PhD
Jones	AA
Jones	BA

EMPLOYEE_SIBLING

EmployeeName	EmployeeSibling
Chau	Eileen
Chau	Jonathan
Green	Nikki
Jones	Frank
Jones	Fred
Jones	Sally

A Multivalued Dependency Example (2 of 2)

Figure 2-25 [continued]

PARTKIT_PART

PartKitName	Part
Bike Repair	Screwdriver
Bike Repair	Tube Fix
Bike Repair	Wrench
First Aid	Aspirin
First Aid	Bandaids
First Aid	Elastic Bands
First Aid	Ibuprofin
Toolbox	Drill and drill bits
Toolbox	Hammer
Toolbox	Saw
Toolbox	Screwdriver
Toolbox	Wrench

The Normal Forms 1

- First Normal Form (1NF)
  - each cell has only one value, and all entries in a column are of the same kind
- Second Normal Form (2NF)
  - each table is in 1NF and all non-key attributes are determined by only the entire primary key
- Third Normal Form (3NF)
  - each table is in 2NF and no non-key attributes are determined by another non-key attribute

The Normal Forms 2

- Boyce Codd Normal Form (BCNF)
  - each table is in 3NF and all determinants are candidate keys
- Fourth Normal Form (4NF)
  - each table is in BCNF and all multivalued dependencies have been moved to their own table
- Fifth Normal Form (5NF)
- Domain/Key Normal Form (DK/NF)

Copyright