14e

PROBLEM-SOLVING CASES IN MICROSOFT® ACCESS™ & EXCEL®

Monk

Brady

Mendelsohn

PROBLEM-SOLVING CASES IN MICROSOFT® ACCESS™ AND EXCEL®

PROBLEM-SOLVING CASES IN MICROSOFT® ACCESS™ AND EXCEL®

Fourteenth Annual Edition

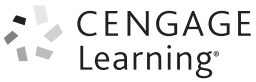
Ellen F. Monk

Joseph A. Brady

Emilio I. Mendelsohn

CENGAGE Learning®

Australia • Brazil • Mexico • Singapore • United Kingdom • United States



Problem-Solving Cases in Microsoft® Access™ and Excel®, 14th Annual Edition

Ellen F. Monk, Joseph A. Brady, Emilio I. Mendelsohn

Vice President, General Manager, Science, Math & Quantitative Business: Balraj Kalsi

Product Director: Mike Schenk

Senior Product Team Manager: Joe Sabatino

Content Developer: Dan Seiter

Marketing Director: Michele McTighe

Senior Marketing Manager: Eric La Scola

Marketing Coordinator: Will Guiliani

Art and Cover Direction, Production Management, and Composition: Lumina Datamatics, Inc.

Media Developer: Dan Seiter

Intellectual Property

Analyst: Brittani Morgan

Project Manager: Kathryn Kucharek

Manufacturing Planner: Ron Montgomery

Cover Image(s):

yienkeat/Shutterstock.com; wavebreakmedia/Shutterstock.com; Jacob Lund/Shutterstock.com; Gaudilab/Shutterstock.com

Library of Congress Control Number: 2016932029

Student Edition ISBN: 978-1-305-86862-5

Cengage Learning
20 Channel Center Street
Boston, MA 02210
USA

Cengage Learning is a leading provider of customized learning solutions, with employees residing in nearly 40 different countries and with sales in more than 125 countries around the world. Find your local representative at **www.cengage.com**.

Cengage Learning products are represented in Canada by Nelson Education, Ltd.

To learn more about Cengage Learning, visit **www.cengage.com.**

Purchase any of our products at your local college store or at our preferred online store: **www.cengagebrain.com.**

Printed in the United States of America
Print Number: 01 Print Year: 2016

To Karen—thank you for the support.
—JAB
To my problem-solving students.
—EFM
To Laurie, the ultimate fighter.
—EIM

BRIEF CONTENTS

Part 3: Decision Support Cases Using Microsoft Excel Solver

Part 4: Decision Support Case Using Basic Excel Functionality

Part 5: Integration Cases Using Microsoft Access and Excel

Part 6: Advanced Skills Using Excel

Part 7: Presentation Skills

PREFACE

For more than two decades, we have taught MIS courses at the university level. From the start, we wanted to use good computer-based case studies for the database and decision-support portions of our courses.

At first, we could not find a casebook that met our needs! This surprised us because we thought our requirements were not unreasonable. First, we wanted cases that asked students to think about real-world business situations. Second, we wanted cases that provided students with hands-on experience, using the kind of software that they had learned to use in their computer literacy courses—and that they would later use in business. Third, we wanted cases that would strengthen students' ability to analyze a problem, examine alternative solutions, and implement a solution using software. Undeterred by the lack of casebooks, we wrote our own for Cengage Learning.

This is the fourteenth casebook we have written for Cengage Learning. The cases are all new, and the tutorials have been updated using Microsoft Office 2013.

As with our prior casebooks, we include tutorials that prepare students for the cases, which are challenging but doable. The cases are organized to help students think about the logic of each case's business problem and then about how to use the software to solve the business problem. The cases fit well in an undergraduate MIS course, an MBA information systems course, or a computer science course devoted to business-oriented programming.

BOOK ORGANIZATION

The book is organized into seven parts:

- Database cases using Access
- Decision support cases using the Excel Scenario Manager
- Decision support cases using the Excel Solver
- A decision support case using basic Excel functionality
- Integration cases using Access and Excel
- Advanced Excel skills
- Presentation skills

Part 1 begins with two tutorials that prepare students for the Access case studies. Parts 2 and 3 each begin with a tutorial that prepares students for the Excel case studies. All four tutorials provide students with hands-on practice in using the software's more advanced features—the kind of support that other books about Access and Excel do not provide. Part 4 asks students to use Excel's basic functionality for decision support. Part 5 challenges students to use both Access and Excel to find a solution to a business problem. Part 6 is a tutorial about advanced skills students might need to complete some of the Excel cases. Part 7 is a tutorial that hones students' skills in creating and delivering an oral presentation to business managers. The next sections explore these parts of the book in more depth.

Part 1: Database Cases Using Access

This section begins with two tutorials and then presents five case studies.

Tutorial A: Database Design

This tutorial helps students understand how to set up tables to create a database, without requiring students to learn formal analysis and design methods, such as data normalization.

Tutorial B: Microsoft Access

The second tutorial teaches students the more advanced features of Access queries and reports—features that students will need to know to complete the cases.

Cases 1–5

Five database cases follow Tutorials A and B. The students must use the Access database in each case to create forms, queries, and reports that help management. The first case is an easier "warm-up" case. The next four cases require more effort to design the database and implement the results.

Part 2: Decision Support Cases Using Excel Scenario Manager

This section has one tutorial and two decision support cases that require the use of the Excel Scenario Manager.

Tutorial C: Building a Decision Support System in Excel

This section begins with a tutorial that uses Excel to explain decision support and fundamental concepts of spreadsheet design. The case emphasizes the use of Scenario Manager to organize the output of multiple "what-if" scenarios.

Cases 6–7

Students can complete these two cases with Scenario Manager. In each case, students must use Excel to model two or more solutions to a problem. Students then use the model outputs to identify and document the preferred solution in a memo.

Part 3: Decision Support Cases Using Microsoft Excel Solver

This section has one tutorial and two decision support cases that require the use of Excel Solver.

Tutorial D: Building a Decision Support System Using Microsoft Excel Solver

This section begins with a tutorial for using Excel Solver, a powerful decision support tool for solving optimization problems.

Cases 8–9

Students use the Excel Solver tool in each case to analyze alternatives and identify and document the preferred solution.

Part 4: Decision Support Case Using Basic Excel Functionality

Case 10

The book continues with a case that uses basic Excel functionality. The case does not require Scenario Manager or Solver. Excel is used to test students' analytical skills in a "what-if" analysis.

Part 5: Integration Cases Using Microsoft Access and Excel

Cases 11 and 12

These cases integrate Access and Excel. The cases show students how to share data between Access and Excel to solve problems.

Part 6: Advanced Skills Using Excel

This part contains one tutorial that focuses on using advanced techniques in Excel.

Tutorial E: Guidance for Excel Cases

Some cases may require the use of Excel techniques that are not discussed in other tutorials or cases in this casebook. For example, techniques for using data tables and pivot tables are explained in Tutorial E rather than in the cases themselves.

Part 7: Presentation Skills

Tutorial F: Giving an Oral Presentation

Each case may include an optional assignment that lets students practice making a presentation to management to summarize the results of their case analysis. This tutorial gives advice for creating oral presentations. It also includes technical information on charting, a technique that is useful in case analyses or as support for presentations. This tutorial will help students to organize their recommendations, to present their solutions both in words and graphics, and to answer questions from the audience. For larger classes, instructors may want to have students work in teams to create and deliver their presentations, which would model the team approach used by many corporations.

INDIVIDUAL CASE DESIGN

The format of the cases uses the following template:

- Each case begins with a *Preview* and an overview of the tasks.
- The next section, *Preparation*, tells students what they need to do or know to complete the case successfully. Again, the tutorials also prepare students for the cases.
- The third section, *Background*, provides the business context that frames the case. The background of each case models situations that require the kinds of thinking and analysis that students will need in the business world.
- The *Assignment* sections are generally organized to help students develop their analyses.
- The last section, *Deliverables*, lists the finished materials that students must hand in: printouts, a memorandum, a presentation, and files. The list is similar to the deliverables that a business manager might demand.

USING THE CASES AND TUTORIALS

We have successfully used cases like these in our undergraduate MIS courses. We usually begin the semester with Access database instruction. We assign the Access database tutorials and then a case to each student. Then, to teach students how to use the Excel decision support system, we do the same thing: We assign a tutorial and then a case.

Some instructors have asked for access to extra cases, especially in the second semester of a school year. For example, they assigned the integration case in the fall, and they need another one for the spring. To meet this need, we have set up an online "Hall of Fame" that features some of our favorite cases from prior editions. These password-protected cases are available to instructors on the Cengage Learning Web site. Go to *www.cengage.com/login* and search for this textbook by title, author, or ISBN. Note that the cases are in Microsoft Office 2013 format.

Tutorials C and D, which describe the use of Scenario Manager and Solver, have changed significantly in the current edition. If you prefer to continue using Tutorials C and D from prior editions, note that they are available with the Hall of Fame cases described in the previous paragraph.

TECHNICAL INFORMATION

The cases in this textbook were written using Microsoft Office 2013, and the textbook was tested for quality assurance using the Windows 10 operating system, Microsoft Access 2013, and Microsoft Excel 2013.

Data Files and Solution Files

We have created "starter" data files for the Excel cases, so students need not spend time typing in the spreadsheet skeleton. Cases 11 and 12 also ask students to load Access and Excel starter files. All these files are on the Cengage Learning Web site, which is available both to students and instructors. Instructors should go to *www.cengage.com/login* and search for this textbook by title, author, or ISBN. Students will find the files at *www.cengagebrain.com/login.* You are granted a license to copy the data files to any computer or computer network used by people who have purchased this textbook.

Solutions to the material in the text are available to instructors at *www.cengage.com/login*. Search for this textbook by title, author, or ISBN. The solutions are password protected.

ACKNOWLEDGMENTS

We would like to give many thanks to the team at Cengage Learning, including our Content Developer, Dan Seiter; and our Content Project Manager, Jyotsna Ojha. As always, we acknowledge our students' diligent work.

PART 1

DATABASE CASES USING ACCESS

TUTORIAL A

DATABASE DESIGN

This tutorial has three sections. The first section briefly reviews basic database terminology. The second section teaches database design. The third section features a database design problem for practice.

REVIEW OF TERMINOLOGY

You will begin by reviewing some basic terms that will be used throughout this textbook. In Access, a **database** is a group of related objects that are saved in one file. An Access **object** can be a table, form, query, or report. You can identify an Access database file by its suffix, .accdb.

A **table** consists of data that is arrayed in rows and columns. A **row** of data is called a **record**. A **column** of data is called a **field**. Thus, a record is a set of related fields. The fields in a table should be related to one another in some way. For example, a company might want to keep its employee data together by creating a database table called Employee. That table would contain data fields about employees, such as their names and addresses. It would not have data fields about the company's customers; that data would go in a Customer table.

A field's values have a **data type** that is declared when the table is defined. Thus, when data is entered into the database, the software knows how to interpret each entry. Data types in Access include the following:

- *Text* for words
- *Integer* for whole numbers
- *Double* for numbers that have a decimal value
- *Currency* for numbers that represent dollars and cents
- *Yes/No* for variables that have only two values (such as 1/0, on/off, yes/no, and true/false)
- *Date/Time* for variables that are dates or times

Each database table should have a **primary key** field—a field in which each record has a *unique* value. For example, in an Employee table, a field called Employee Identification Number (EIN) could serve as a primary key. (This assumes that each employee is given a number when hired, and that these numbers are not reused later.) Sometimes, a table does not have a single field whose values are all different. In that case, two or more fields are combined into a **compound primary key**. The combination of the fields' values is unique.

Database tables should be logically related to one another. For example, suppose a company has an Employee table with fields for EIN, Name, Address, and Telephone Number. For payroll purposes, the company has an Hours Worked table with a field that summarizes Labor Hours for individual employees. The relationship between the Employee table and Hours Worked table needs to be established in the database so you can determine the number of hours worked by any employee. To create this relationship, you include the primary key field from the Employee table (EIN) as a field in the Hours Worked table. In the Hours Worked table, the EIN field is then called a **foreign key** because it's from a "foreign" table.

In Access, data can be entered directly into a table or it can be entered into a form, which then inserts the data into a table. A **form** is a database object that is created from an existing table to make the process of entering data more user-friendly.

A **query** is the database equivalent of a question that is posed about data in a table (or tables). For example, suppose a manager wants to know the names of employees who have worked for the company for more than five years. A query could be designed to search the Employee table for the information. The query would be run, and its output would answer the question.

Queries can be designed to search multiple tables at a time. For this to work, the tables must be connected by a **join** operation, which links tables on the values in a field that they have in common. The common field acts as a "hinge" for the joined tables; when the query is run, the query generator treats the joined tables as one large table.

In Access, queries that answer a question are called *select queries* because they select relevant data from the database records. Queries also can be designed to change data in records, add a record to the end of a table, or delete entire records from a table. These queries are called **update**, **append**, and **delete** queries, respectively.

Access has a **report** generator that can be used to format a table's data or a query's output.

DATABASE DESIGN

Designing a database involves determining which tables belong in the database and then creating the fields that belong in each table. This section begins with an introduction to key database design concepts, then discusses design rules you should use when building a database. First, the following key concepts are defined:

- Entities
- Relationships
- Attributes

Database Design Concepts

Computer scientists have highly formalized ways of documenting a database's logic. Learning their notations and mechanics can be time-consuming and difficult. In fact, doing so usually takes a good portion of a systems analysis and design course. This tutorial will teach you database design by emphasizing practical business knowledge; the approach should enable you to design serviceable databases quickly. Your instructor may add more formal techniques.

A database models the logic of an organization's operation, so your first task is to understand the operation. You can talk to managers and workers, make your own observations, and look at business documents, such as sales records. Your goal is to identify the business's "entities" (sometimes called *objects*). An **entity** is a thing or event that the database will contain. Every entity has characteristics, called **attributes**, and one or more **relationships** to other entities. Let's take a closer look.

Entities

As previously mentioned, an entity is a tangible thing or an event. The reason for identifying entities is that *an entity eventually becomes a table in the database*. Entities that are things are easy to identify. For example, consider a video store. The database for the video store would probably need to contain the names of DVDs and the names of customers who rent them, so you would have one entity named Video and another named Customer.

In contrast, entities that are events can be more difficult to identify, probably because they are more conceptual. However, events are real, and they are important. In the video store example, one event would be Video Rental and another event would be Hours Worked by employees.

In general, your analysis of an organization's operations is made easier when you realize that organizations usually have physical entities such as these:

- Employees
- Customers
- Inventory (products or services)
- Suppliers

Thus, the database for most organizations would have a table for each of these entities. Your analysis also can be made easier by knowing that organizations engage in transactions internally (within the company) and externally (with the outside world). Such transactions are explained in an introductory accounting course, but most people understand them from events that occur in daily life. Consider the following examples:

- Organizations generate revenue from sales or interest earned. Revenue-generating transactions include event entities called Sales and Interest Earned.
- Organizations incur expenses from paying hourly employees and purchasing materials from suppliers. Hours Worked and Purchases are event entities in the databases of most organizations.

Thus, identifying entities is a matter of observing what happens in an organization. Your powers of observation are aided by knowing what entities exist in the databases of most organizations.

Relationships

As an analyst building a database, you should consider the relationship of each entity to the other entities you have identified. For example, a college database might contain entities for Student, Course, and Section to contain data about each. A relationship between Student and Section could be expressed as "Students enroll in sections."

An analyst also must consider the **cardinality** of any relationship. Cardinality can be one-to-one, one-to-many, or many-to-many:

- In a one-to-one relationship, one instance of the first entity is related to just one instance of the second entity.
- In a one-to-many relationship, one instance of the first entity is related to many instances of the second entity, but each instance of the second entity is related to only one instance of the first.
- In a many-to-many relationship, one instance of the first entity is related to many instances of the second entity, and one instance of the second entity is related to many instances of the first.

For a more concrete understanding of cardinality, consider again the college database with the Student, Course, and Section entities. The university catalog shows that a course such as Accounting 101 can have more than one section: 01, 02, 03, 04, and so on. Thus, you can observe the following relationships:

- The relationship between the entities Course and Section is one-to-many. Each course has many sections, but each section is associated with just one course.
- The relationship between Student and Section is many-to-many. Each student can be in more than one section, because each student can take more than one course. Also, each section has more than one student.

Thinking about relationships and their cardinalities may seem tedious to you. However, as you work through the cases in this text, you will see that this type of analysis can be valuable in designing databases. In the case of many-to-many relationships, you should determine the tables a given database needs; in the case of one-to-many relationships, you should decide which fields the tables need to share.

Attributes

An attribute is a characteristic of an entity. You identify attributes of an entity because *attributes become a table's fields*. If an entity can be thought of as a noun, an attribute can be considered an adjective that describes the noun. Continuing with the college database example, consider the Student entity. Students have names, so Last Name would be an attribute of the Student entity and therefore a field in the Student table. First Name would be another attribute, as well as Address, Phone Number, and other descriptive fields.

Sometimes it can be difficult to tell the difference between an attribute and an entity, but one good way is to ask whether more than one attribute is possible for each entity. If more than one instance is possible, but you do not know the number in advance, you are working with an entity. For example, assume that a student could have a maximum of two addresses—one for home and one for college. You could specify attributes Address 1 and Address 2. Next, consider that you might not know the number of student addresses in advance, meaning that all addresses have to be recorded. In that case, you would not know how many fields to set aside in the Student table for addresses. Therefore, you would need a separate Student Addresses table (entity) that would show any number of addresses for a given student.

Database Design Rules

As described previously, your first task in database design is to understand the logic of the business situation. Once you understand this logic, you are ready to build the database. To create a context for learning about database design, look at a hypothetical business operation and its database needs.

Example: The Talent Agency

Suppose you have been asked to build a database for a talent agency that books musical bands into nightclubs. The agent needs a database to keep track of the agency's transactions and to answer day-to-day questions. For example, a club manager often wants to know which bands are available on a certain date at a certain time, or wants to know the agent's fee for a certain band. The agent may want to see a list of all band members and the instrument each person plays, or a list of all bands that have three members.

Suppose that you have talked to the agent and have observed the agency's business operation. You conclude that your database needs to reflect the following facts:

1. A booking is an event in which a certain band plays in a particular club on a particular date, starting and ending at certain times, and performing for a specific fee. A band can play more than once a day. The Heartbreakers, for example, could play at the East End Cafe in the afternoon and then at the West End Cafe on the same night. For each booking, the club pays the talent agent. The agent keeps a five percent fee and then gives the remainder of the payment to the band.
2. Each band has at least two members and an unlimited maximum number of members. The agent notes a telephone number of just one band member, which is used as the band's contact number. No two bands have the same name or telephone number.
3. Band member names are not unique. For example, two bands could each have a member named Sally Smith.
4. The agent keeps track of just one instrument that each band member plays. For the purpose of this database, "vocals" are considered an instrument.
5. Each band has a desired fee. For example, the Lightmetal band might want $700 per booking, and would expect the agent to try to get at least that amount.
6. Each nightclub has a name, an address, and a contact person. The contact person has a telephone number that the agent uses to call the club. No two clubs have the same name, contact person, or telephone number. Each club has a target fee. The contact person will try to get the agent to accept that fee for a band's appearance.
7. Some clubs feed the band members for free; others do not.

Before continuing with this tutorial, you might try to design the agency's database on your own. Ask yourself: What are the entities? Recall that business databases usually have Customer, Employee, and Inventory entities, as well as an entity for the event that generates revenue transactions. Each entity becomes a table in the database. What are the relationships among the entities? For each entity, what are its attributes? For each table, what is the primary key?

Six Database Design Rules

Assume that you have gathered information about the business situation in the talent agency example. Now you want to identify the tables required for the database and the fields needed in each table. Observe the following six rules:

Rule 1: You do not need a table for the business. The database represents the entire business. Thus, in the example, Agent and Agency are not entities.

Rule 2: Identify the entities in the business description. Look for typical things and events that will become tables in the database. In the talent agency example, you should be able to observe the following entities:

- *Things*: The product (inventory for sale) is Band. The customer is Club.
- *Events*: The revenue-generating transaction is Bookings.

You might ask yourself: Is there an Employee entity? Isn't Instrument an entity? Those issues will be discussed as the rules are explained.

Rule 3: Look for relationships among the entities. Look for one-to-many relationships between entities. The relationship between those entities must be established in the tables, using a foreign key. For details, see the following discussion in Rule 4 about the relationship between Band and Band Member.

Look for many-to-many relationships between entities. Each of these relationships requires a third entity that associates the two entities in the relationship. Recall the many-to-many relationship from the college database scenario that involved Student and Section entities. To display the enrollment of specific students in specific sections, a third table would be required. The mechanics of creating such a table are described in Rule 4 during the discussion of the relationship between Band and Club.

Rule 4: Look for attributes of each entity and designate a primary key. As previously mentioned, you should think of the entities in your database as nouns. You should then create a list of adjectives that describe those nouns. These adjectives are the attributes that will become the table's fields. After you have identified fields for each table, you should check to see whether a field has unique

values. If such a field exists, designate it as the primary key field; otherwise, designate a compound primary key.

In the talent agency example, the attributes, or fields, of the Band entity are Band Name, Band Phone Number, and Desired Fee, as shown in Figure A-1. Assume that no two bands have the same name, so the primary key field can be Band Name. The data type of each field is shown.

BAND	
Field Name	**Data Type**
Band Name (primary key)	Text
Band Phone Number	Text
Desired Fee	Currency

Source: © 2016 Cengage Learning®

FIGURE A-1 The Band table and its fields

Two Band records are shown in Figure A-2.

Band Name (primary key)	Band Phone Number	Desired Fee
Heartbreakers	981 831 1765	$800
Lightmetal	981 831 2000	$700

Source: © 2016 Cengage Learning®

FIGURE A-2 Records in the Band table

If two bands might have the same name, Band Name would not be a good primary key, so a different unique identifier would be needed. Such situations are common. Most businesses have many types of inventory, and duplicate names are possible. The typical solution is to assign a number to each product to use as the primary key field. A college could have more than one faculty member with the same name, so each faculty member would be assigned an employee identification number. Similarly, banks assign a personal identification number (PIN) for each depositor. Each automobile produced by a car manufacturer gets a unique vehicle identification number (VIN). Most businesses assign a number to each sale, called an invoice number. (The next time you go to a grocery store, note the number on your receipt. It will be different from the number on the next customer's receipt.)

At this point, you might be wondering why Band Member would not be an attribute of Band. The answer is that, although you must record each band member, you do not know in advance how many members are in each band. Therefore, you do not know how many fields to allocate to the Band table for members. (Another way to think about band members is that they are the agency's employees, in effect. Databases for organizations usually have an Employee entity.) You should create a Band Member table with the attributes Member ID Number, Member Name, Band Name, Instrument, and Phone. A Member ID Number field is needed because member names may not be unique. The table and its fields are shown in Figure A-3.

BAND MEMBER	
Field Name	**Data Type**
Member ID Number (primary key)	Text
Member Name	Text
Band Name (foreign key)	Text
Instrument	Text
Phone	Text

Source: © 2016 Cengage Learning®

FIGURE A-3 The Band Member table and its fields

Note in Figure A-3 that the phone number is classified as a Text data type because the field values will not be used in an arithmetic computation. The benefit is that Text data type values take up fewer bytes than Numerical or Currency data type values; therefore, the file uses less storage space. You should also use the Text data type for number values, such as zip codes.

Five records in the Band Member table are shown in Figure A-4.

Member ID Number (primary key)	Member Name	Band Name	Instrument	Phone
0001	Pete Goff	Heartbreakers	Guitar	981 444 1111
0002	Joe Goff	Heartbreakers	Vocals	981 444 1234
0003	Sue Smith	Heartbreakers	Keyboard	981 555 1199
0004	Joe Jackson	Lightmetal	Sax	981 888 1654
0005	Sue Hoopes	Lightmetal	Piano	981 888 1765

Source: © 2016 Cengage Learning®

FIGURE A-4 Records in the Band Member table

You can include Instrument as a field in the Band Member table because the agent records only one instrument for each band member. Thus, you can use the instrument as a way to describe a band member, much like the phone number is part of the description. Phone could not be the primary key because two members might share a telephone and because members might change their numbers, making database administration more difficult.

You might ask why Band Name is included in the Band Member table. The common-sense reason is that you did not include the Member Name in the Band table. You must relate bands and members somewhere, and the Band Member table is the place to do it.

To think about this relationship in another way, consider the cardinality of the relationship between Band and Band Member. It is a one-to-many relationship: one band has many members, but each member in the database plays in just one band. You establish such a relationship in the database by using the primary key field of one table as a foreign key in the other table. In Band Member, the foreign key Band Name is used to establish the relationship between the member and his or her band.

The attributes of the Club entity are Club Name, Address, Contact Name, Club Phone Number, Preferred Fee, and Feed Band?. The Club table can define the Club entity, as shown in Figure A-5.

CLUB	
Field Name	**Data Type**
Club Name (primary key)	Text
Address	Text
Contact Name	Text
Club Phone Number	Text
Preferred Fee	Currency
Feed Band?	Yes/No

Source: © 2016 Cengage Learning®

FIGURE A-5 The Club table and its fields

Two records in the Club table are shown in Figure A-6.

Club Name (primary key)	Address	Contact Name	Club Phone Number	Preferred Fee	Feed Band?
East End	1 Duce St.	Al Pots	981 444 8877	$600	Yes
West End	99 Duce St.	Val Dots	981 555 0011	$650	No

Source: © 2016 Cengage Learning®

FIGURE A-6 Records in the Club table

You might wonder why Bands Booked into Club (or a similar name) is not an attribute of the Club table. There are two reasons. First, you do not know in advance how many bookings a club will have, so the value cannot be an attribute. Second, Bookings is the agency's revenue-generating transaction, an event entity, and you need a table for that business transaction. Consider the booking transaction next.

You know that the talent agent books a certain band into a certain club for a specific fee on a certain date, starting and ending at a specific time. From that information, you can see that the attributes of the Bookings entity are Band Name, Club Name, Date, Start Time, End Time, and Fee. The Bookings table and its fields are shown in Figure A-7.

BOOKINGS	
Field Name	**Data Type**
Band Name (foreign key)	Text
Club Name (foreign key)	Text
Date	Date/Time
Start Time	Date/Time
End Time	Date/Time
Fee	Currency

Source: © 2016 Cengage Learning®

FIGURE A-7 The Bookings table and its fields—and no designation of a primary key

Some records in the Bookings table are shown in Figure A-8.

Band Name	Club Name	Date	Start Time	End Time	Fee
Heartbreakers	East End	11/21/16	21:30	23:30	$800
Heartbreakers	East End	11/22/16	21:00	23:30	$750
Heartbreakers	West End	11/28/16	19:00	21:00	$500
Lightmetal	East End	11/21/16	18:00	20:00	$700
Lightmetal	West End	11/22/16	19:00	21:00	$750

Source: © 2016 Cengage Learning®

FIGURE A-8 Records in the Bookings table

Note that no single field is guaranteed to have unique values, because each band is likely to be booked many times and each club might be used many times. Furthermore, each date and time can appear more than once. Thus, no one field can be the primary key.

If a table does not have a single primary key field, you can make a compound primary key whose field values will be unique when taken together. Because a band can be in only one place at a time, one possible solution is to create a compound key from the Band Name, Date, and Start Time fields. An alternative solution is to create a compound primary key from the Club Name, Date, and Start Time fields.

If you don't want a compound key, you could create a field called Booking Number. Each booking would then have its own unique number, similar to an invoice number.

You can also think about this event entity in a different way. Over time, a band plays in many clubs, and each club hires many bands. Thus, Band and Club have a many-to-many relationship, which signals the need for a table between the two entities. A Bookings table would associate the Band and Club tables. You implement an associative table by including the primary keys from the two tables that are associated. In this case, the primary keys from the Band and Club tables are included as foreign keys in the Bookings table.

Rule 5: Avoid data redundancy. You should not include extra (redundant) fields in a table. Redundant fields take up extra disk space and lead to data entry errors because the same value must be entered in multiple tables, increasing the chance of a keystroke error. In large databases, keeping track of multiple instances of the same data is nearly impossible, so contradictory data entries become a problem.

Consider this example: Why wouldn't Club Phone Number be included in the Bookings table as a field? After all, the agent might have to call about a last-minute booking change and could quickly look up the number in the Bookings table. Assume that the Bookings table includes Booking Number as the primary key and Club Phone Number as a field. Figure A-9 shows the Bookings table with the additional field.

BOOKINGS	
Field Name	**Data Type**
Booking Number (primary key)	Text
Band Name (foreign key)	Text
Club Name (foreign key)	Text
Club Phone Number	Text
Date	Date/Time
Start Time	Date/Time
End Time	Date/Time
Fee	Currency

Source: © 2016 Cengage Learning®

FIGURE A-9 The Bookings table with an unnecessary field—Club Phone Number

The fields Date, Start Time, End Time, and Fee logically depend on the Booking Number primary key—they help define the booking. Band Name and Club Name are foreign keys and are needed to establish the relationship between the Band, Club, and Bookings tables. But what about Club Phone Number? It is not defined by the Booking Number. It is defined by Club Name—*in other words, it is a function of the club, not of the booking*. Thus, the Club Phone Number field does not belong in the Bookings table. It is already in the Club table.

Perhaps you can see the practical data-entry problem of including Club Phone Number in Bookings. Suppose a club changed its contact phone number. The agent could easily change the number one time, in the Club table. However, the agent would need to remember which other tables contained the field and change the values there too. In a small database, this task might not be difficult, but in larger databases, having redundant fields in many tables makes such maintenance difficult, which means that redundant data is often incorrect.

You might object by saying, "What about all of those foreign keys? Aren't they redundant?" In a sense, they are. But they are needed to establish the one-to-many relationship between one entity and another, as discussed previously.

Rule 6: Do not include a field if it can be calculated from other fields. A **calculated field** is made using the query generator. Thus, the agent's fee is not included in the Bookings table because it can be calculated by query (here, five percent multiplied by the booking fee).

PRACTICE DATABASE DESIGN PROBLEM

Imagine that your town library wants to keep track of its business in a database, and that you have been called in to build the database. You talk to the town librarian, review the old paper-based records, and watch people use the library for a few days. You learn the following about the library:

1. Any resident of the town can get a library card simply by asking for one. The library considers each cardholder a member of the library.
2. The librarian wants to be able to contact members by telephone and by mail. She calls members when books are overdue or when requested materials become available. She likes to mail a thank-you note to each patron on his or her anniversary of becoming a member of the library. Without a database, contacting members efficiently can be difficult; for example, multiple members can have the same name. Also, a parent and a child might have the same first and last name, live at the same address, and share a phone.
3. The librarian tries to keep track of each member's reading interests. When new books come in, the librarian alerts members whose interests match those books. For example, long-time member Sue Doaks is interested in reading Western novels, growing orchids, and baking bread. There must be some way to match her interests with available books. One complication is that, although the librarian wants to track all of a member's reading interests, she wants to classify each book as being in just one category of interest. For example, the classic gardening book *Orchids of France* would be classified as a book about orchids or a book about France, but not both.
4. The library stocks thousands of books. Each book has a title and any number of authors. Also, more than one book in the library might have the same title. Similarly, multiple authors might have the same name.
5. A writer could be the author of more than one book.
6. A book will be checked out repeatedly as time goes on. For example, *Orchids of France* could be checked out by one member in March, by another member in July, and by another member in September.
7. The library must be able to identify whether a book is checked out.
8. A member can check out any number of books in one visit. Also, a member might visit the library more than once a day to check out books.
9. All books that are checked out are due back in two weeks, with no exceptions. The librarian would like to have an automated way of generating an overdue book list each day so she can telephone offending members.
10. The library has a number of employees. Each employee has a job title. The librarian is paid a salary, but other employees are paid by the hour. Employees clock in and out each day. Assume that all employees work only one shift per day and that all are paid weekly. Pay is deposited directly into an employee's checking account—no checks are hand-delivered. The database needs to include the librarian and all other employees.

Design the library's database, following the rules set forth in this tutorial. Your instructor will specify the format of your work. Here are a few hints in the form of questions:

- A book can have more than one author. An author can write more than one book. How would you describe the relationship between books and authors?
- The library lends books for free, of course. If you were to think of checking out a book as a sales transaction for zero revenue, how would you handle the library's revenue-generating event?
- A member can borrow any number of books at one checkout. A book can be checked out more than once. How would you describe the relationship between checkouts and books?

TUTORIAL **B**

MICROSOFT ACCESS

Microsoft Access is a relational database package that runs on the Microsoft Windows operating system. There are many different versions of Access; this tutorial was prepared using Access 2013.

Before using this tutorial, you should know the fundamentals of Access and know how to use Windows. This tutorial explains advanced Access skills you will need to complete database case studies. The tutorial concludes with a discussion of common Access problems and how to solve them.

To prevent losing your work, always observe proper file-saving and closing procedures. To exit Access, click the File tab and select Close, then click the Close button in the upper-right corner. Always end your work with these steps. If you remove your USB key or other portable storage device when database forms and tables are shown on the screen, you will lose your work.

To begin this tutorial, you will create a new database called Employee.

AT THE KEYBOARD

Open Access. Click the Blank desktop database icon from the templates list. Name the database Employee. Click the file folder next to the filename to browse for the folder where you want to save the file. Click the new folder, click Open, and then click OK. Otherwise, your file will be saved automatically in the Documents folder. Click the Create button.

A portion of your opening screen should resemble the screen shown in Figure B-1.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-1 Entering data in Datasheet view

When you create a table, Access opens it in Datasheet view by default. Because you will use Design view to build your tables, close the new table by clicking the *X* in the upper-right corner of the table window that corresponds to Close "Table1." You are now on the Home tab in the Database window of Access, as shown in Figure B-2. From this screen, you can create or change objects.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-2 The Database window Home tab in Access

CREATING TABLES

Your database will contain data about employees, their wage rates, and the hours they worked.

Defining Tables

In the Database window, build three new tables using the following instructions.

AT THE KEYBOARD

Defining the Employee Table

This table contains permanent data about employees. To create the table, click the Create tab and then click Table Design in the Tables group. The table's fields are Last Name, First Name, Employee ID, Street Address, City, State, Zip, Date Hired, and US Citizen. The Employee ID field is the primary key field. Change the lengths of Short Text fields from the default 255 spaces to more appropriate lengths; for example, the Last Name field might be 30 spaces, and the Zip field might be 10 spaces. Your completed definition should resemble the one shown in Figure B-3.

Table1

Field Name	Data Type	Description (Optional)
Last Name	Short Text	
First Name	Short Text	
Employee ID	Short Text	
Street Address	Short Text	
City	Short Text	
State	Short Text	
Zip	Short Text	
Date Hired	Date/Time	
US Citizen	Yes/No	

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-3 Fields in the Employee table

When you finish, click the File tab, select Save As, select Save Object As, click the Save As button, and then enter a name for the table. In this example, the table is named Employee. (It is a coincidence that the Employee table has the same name as its database file.) After entering the name, click OK in the Save As window. Close the table by clicking the Close button (X) that corresponds to the Employee table.

Defining the Wage Data Table

This table contains permanent data about employees and their wage rates. The table's fields are Employee ID, Wage Rate, and Salaried. The Employee ID field is the primary key field. Use the data types shown in Figure B-4. Your definition should resemble the one shown in Figure B-4.

Table1

Field Name	Data Type	Description (Optional)
Employee ID	Short Text	
Wage Rate	Currency	
Salaried	Yes/No	

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-4 Fields in the Wage Data table

Click the File tab and then select Save As, select Save Object As, and click the Save As button to save the table definition. Name the table Wage Data.

Defining the Hours Worked Table

The purpose of this table is to record the number of hours that employees work each week during the year. The table's three fields are Employee ID (which has a Short Text data type), Week # (number–long integer), and Hours (number–double). The Employee ID and Week # are the compound keys.

In the following example, the employee with ID number 08965 worked 40 hours in Week 1 of the year and 52 hours in Week 2.

Employee ID	Week #	Hours
08965	1	40
08965	2	52

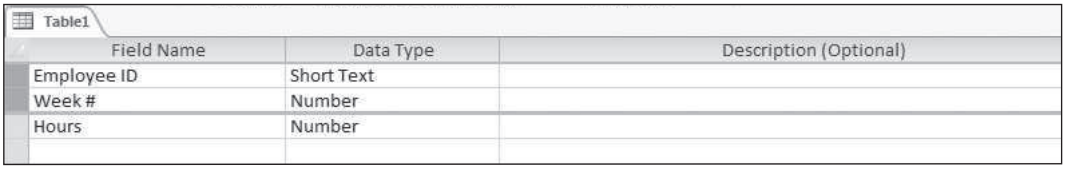
Note that no single field can be the primary key field because 08965 is an entry for each week. In other words, if this employee works each week of the year, 52 records will have the same Employee ID value at the end of the year. Thus, Employee ID values will not distinguish records. No other single field can distinguish these records either, because other employees will have worked during the same week number and some employees will have worked the same number of hours. For example, 40 hours—which corresponds to a full-time workweek—would be a common entry for many weeks.

All of this presents a problem because a table must have a primary key field in Access. The solution is to use a compound primary key; that is, use values from more than one field to create a combined field that will distinguish records. The best compound key to use for the current example consists of the Employee ID field and the Week # field, because as each person works each week, the week number changes. In other words, there is only *one* combination of Employee ID 08965 and Week # 1. Because those values *can occur in only one record*, the combination distinguishes that record from all others.

The first step of setting a compound key is to highlight the fields in the key. Those fields must appear one after the other in the table definition screen. (Plan ahead for that format.) As an alternative, you can highlight one field, hold down the Control key, and highlight the next field.

AT THE KEYBOARD

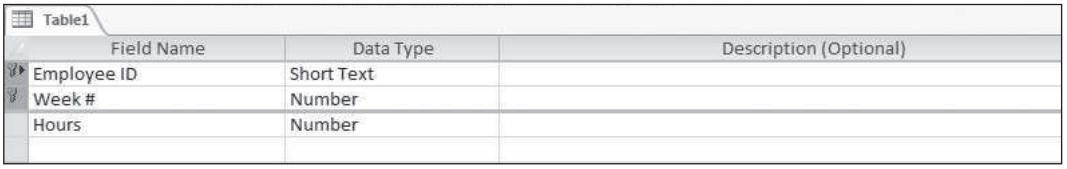
In the Hours Worked table, click the first field's left prefix area (known as the row selector), hold down the mouse button, and drag down to highlight the names of all fields in the compound primary key. Your screen should resemble the one shown in Figure B-5.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-5 Selecting fields for the compound primary key for the Hours Worked table

Now click the Key icon. Your screen should resemble the one shown in Figure B-6.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-6 The compound primary key for the Hours Worked table

You have created the compound primary key and finished defining the table. Click the File tab and then select Save As, select Save Object As, and click the Save As button to save the table as Hours Worked.

Adding Records to a Table

At this point, you have set up the skeletons of three tables. The tables have no data records yet. If you printed the tables now, you would only see column headings (the field names). The most direct way to enter data into a table is to double-click the table's name in the navigation pane at the left side of the screen and then type the data directly into the cells.

NOTE

To display and open the database objects, Access 2013 uses a navigation pane, which is on the left side of the Access window.

AT THE KEYBOARD

On the Home tab of the Database window, double-click the Employee table. Your data entry screen should resemble the one shown in Figure B-7.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-7 The data entry screen for the Employee table

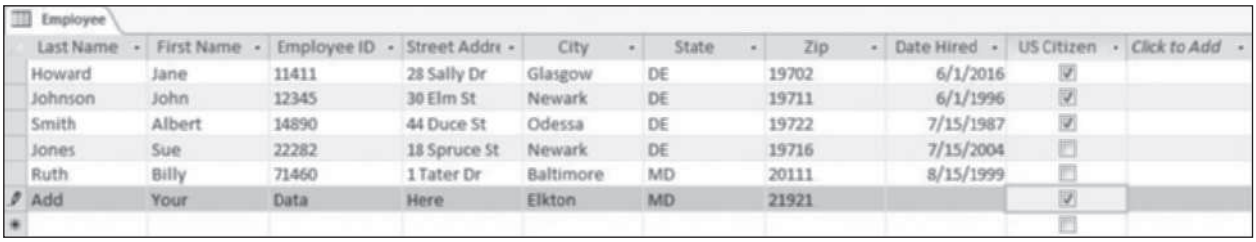
The Employee table has many fields, some of which may be off the screen to the right. Scroll to see obscured fields. (Scrolling happens automatically as you enter data.) Figure B-7 shows all of the fields on the screen.

Enter your data one field value at a time. Note that the first row is empty when you begin. Each time you finish entering a value, press Enter to move the cursor to the next cell. After you enter data in the last cell in a row, the cursor moves to the first cell of the next row *and* Access automatically saves the record. Thus, you do not need to click the File tab and then select Save after entering data into a table.

When entering data in your table, you should enter dates in the following format: 6/15/10. Access automatically expands the entry to the proper format in output.

Also note that Yes/No variables are clicked (checked) for Yes; otherwise, the box is left blank for No. You can change the box from Yes to No by clicking it.

Enter the data shown in Figure B-8 into the Employee table. If you make errors in data entry, click the cell, backspace over the error, and type the correction.

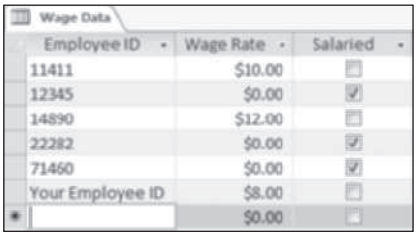


Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-8 Data for the Employee table

Note that the sixth record is *your* data record. Assume that you live in Elkton, Maryland, were hired on today's date (enter the date), and are a U.S. citizen. Make up a fictitious Employee ID number. For purposes of this tutorial, the sixth record has been created using the name of one of this text's authors and the employee ID 09911.

After adding records to the Employee table, open the Wage Data table and enter the data shown in Figure B-9.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-9 Data for the Wage Data table

In this table, you are again asked to create a new entry. For this record, enter your own employee ID. Also assume that you earn $8 an hour and are not salaried. Note that when an employee's Salaried box is not checked (in other words, Salaried = No), the implication is that the employee is paid by the hour. Because salaried employees are not paid by the hour, their hourly rate is 0.00.

When you finish creating the Wage Data table, open the Hours Worked table and enter the data shown in Figure B-10.

Hours Worked

Employee ID	Week #	Hours
11411	1	40
11411	2	50
12345	1	40
12345	2	40
14890	1	38
14890	2	40
22282	1	40
22282	2	40
71460	1	40
71460	2	40
Your Employee ID	1	60
Your Employee ID	2	55
	0	0

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-10 Data for the Hours Worked table

Notice that salaried employees are always given 40 hours. Nonsalaried employees (including you) might work any number of hours. For your record, enter your fictitious employee ID, 60 hours worked for Week 1, and 55 hours worked for Week 2.

CREATING QUERIES

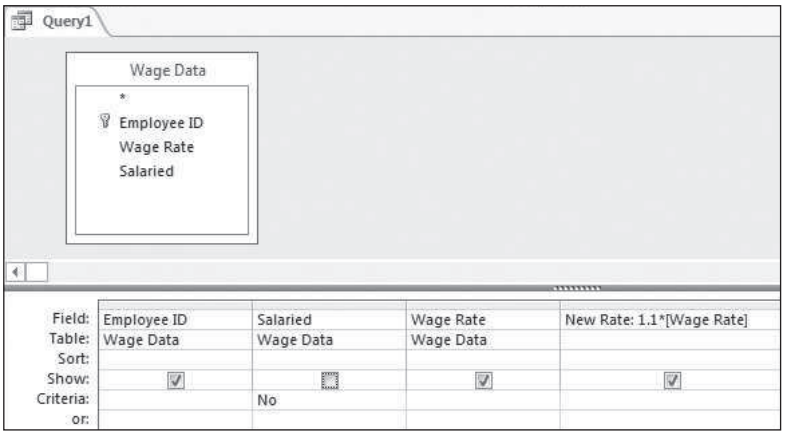
Because you know how to create basic queries, this section explains the advanced queries you will create in the cases in this book.

Using Calculated Fields in Queries

A **calculated field** is an output field made up of *other* field values. A calculated field can be a field in a table; here it is created in the query generator. The calculated field here does not become part of the table—it is just part of the query output. The best way to understand this process is to work through an example.

AT THE KEYBOARD

Suppose you want to see the employee IDs and wage rates of hourly workers, and the new wage rates if all employees were given a 10 percent raise. To view that information, show the employee ID, the current wage rate, and the higher rate, which should be titled New Rate in the output. Figure B-11 shows how to set up the query.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-11 Query setup for the calculated field

To set up this query, you need to select hourly workers by using the Salaried field with Criteria = No. Note in Figure B-11 that the Show box for the field is not checked, so the Salaried field values will not appear in the query output.

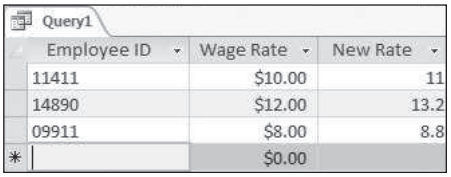
Note the expression for the calculated field, which you can see in the far-right field cell:

New Rate: 1.1 * [Wage Rate]

The term *New Rate:* merely specifies the desired output heading. (Don't forget the colon.) The rest of the expression, 1.1 * [Wage Rate], multiplies the old wage rate by 110 percent, which results in the 10 percent raise.

In the expression, the field name Wage Rate must be enclosed in square brackets. Remember this rule: *Any time an Access expression refers to a field name, the field name must be enclosed in square brackets*.

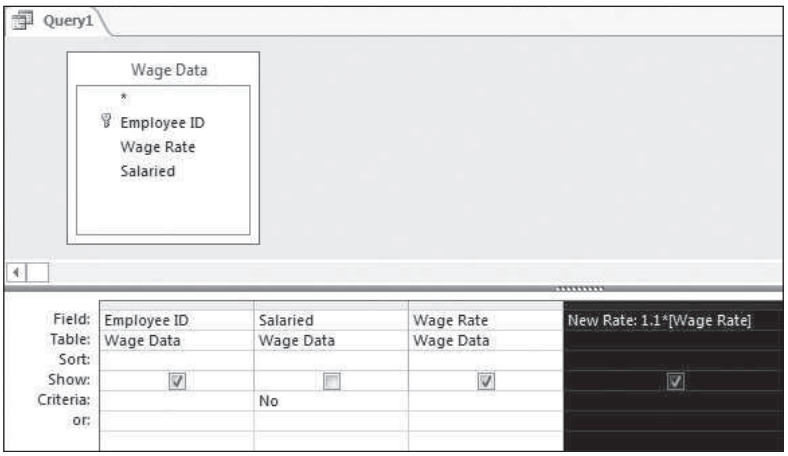
If you run this query, your output should resemble the one in Figure B-12.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-12 Output for a query with calculated field

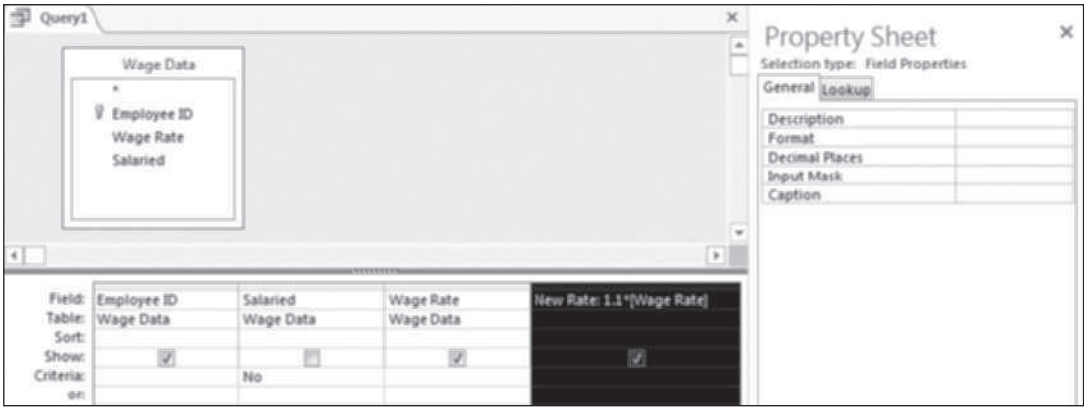
Notice that the calculated field output is not shown in Currency format, but as a Double—a number with digits after the decimal point. To convert the output to Currency format, select the output column by clicking the line above the calculated field expression. The column darkens to indicate its selection. Your data entry screen should resemble the one shown in Figure B-13.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-13 Activating a calculated field in query design

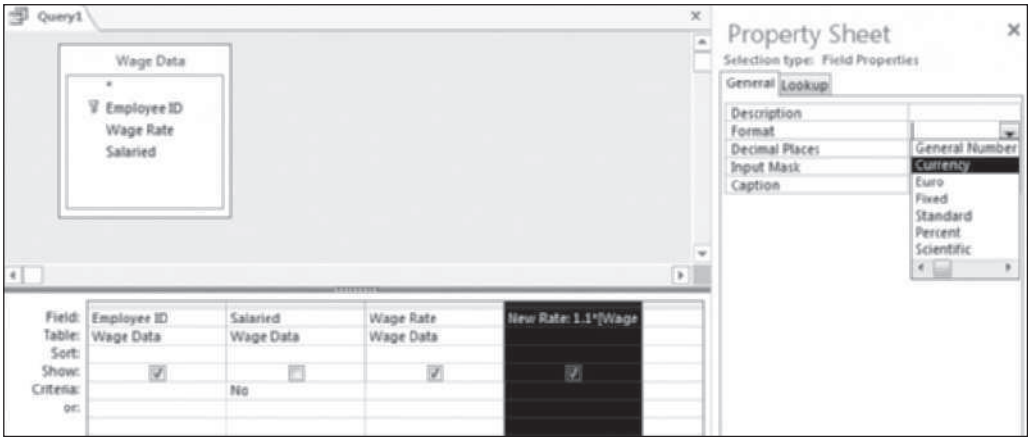
Then, on the Design tab, click Property Sheet in the Show/Hide group. The Field Properties sheet appears, as shown on the right in Figure B-14.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-14 Field properties of a calculated field

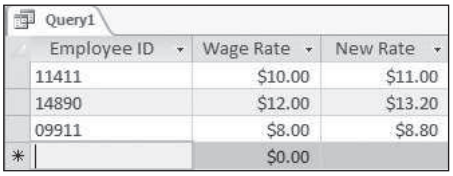
Click Format and choose Currency, as shown in Figure B-15. Then click the *X* in the upper-right corner of the window to close it.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-15 Currency format of a calculated field

When you run the query, the output should resemble the one in Figure B-16.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-16 Query output with formatted calculated field

Next, you examine how to avoid errors when making calculated fields.

Avoiding Errors when Making Calculated Fields

Follow these guidelines to avoid making errors in calculated fields:

- Do not enter the expression in the *Criteria* cell as if the field definition were a filter. You are making a field, so enter the expression in the *Field* cell.

- Spell, capitalize, and space a field's name *exactly* as you did in the table definition. If the table definition differs from what you type, Access thinks you are defining a new field by that name. Access then prompts you to enter values for the new field, which it calls a Parameter Query field. This problem is easy to debug because of the tag *Parameter Query*. If Access asks you to enter values for a parameter, you almost certainly misspelled a field name in an expression in a calculated field or criterion.
  For example, here are some errors you might make for Wage Rate:

  Misspelling: (Wag Rate)

  Case change: (wage Rate / WAGE RATE)

  Spacing change: (WageRate / Wage Rate)

- Do not use parentheses or curly braces instead of the square brackets. Also, do not put parentheses inside square brackets. You *can*, however, use parentheses outside the square brackets in the normal algebraic manner.
  For example, suppose that you want to multiply Hours by Wage Rate to get a field called Wages Owed. This is the correct expression:

  Wages Owed: [Wage Rate] * [Hours]

  The following expression also would be correct:

  Wages Owed: ([Wage Rate] * [Hours])

  But it would *not* be correct to omit the inside brackets, which is a common error:

  Wages Owed: [Wage Rate * Hours]

"Relating" Two or More Tables by the Join Operation

Often, the data you need for a query is in more than one table. To complete the query, you must **join** the tables by linking the common fields. One rule of thumb is that joins are made on fields that have common *values*, and those fields often can be key fields. The names of the join fields are irrelevant; also, the names of the tables or fields to be joined may be the same, but it is not required for an effective join.

Make a join by bringing in (adding) the tables needed. Next, decide which fields you will join. Then click one field name and hold down the left mouse button while you drag the cursor over to the other field's name in its window. Release the button. Access inserts a line to signify the join. (If a relationship between two tables has been formed elsewhere, Access inserts the line automatically, and you do not have to perform the click-and-drag operation. Access often inserts join lines without the user forming relationships.)

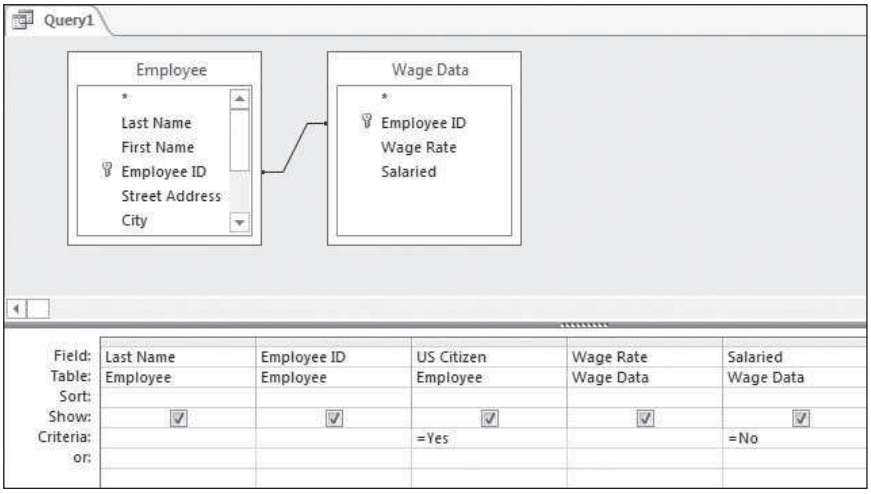
You can join more than two tables. The common fields *need not* be the same in all tables; that is, you can daisy-chain them together.

A common join error is to add a table to the query and then fail to link it to another table. In that case, you will have a table floating in the top part of the QBE (query by example) screen. When you run the query, your output will show the same records over and over. The error is unmistakable because there is *so much* redundant output. The two rules are to add only the tables you need and to link all tables.

Next, you will work through an example of a query that needs a join.

AT THE KEYBOARD

Suppose you want to see the last names, employee IDs, wage rates, salary status, and citizenship only for U.S. citizens and hourly workers. Because the data is spread across two tables, Employee and Wage Data, you should add both tables and pull down the five fields you need. Then you should add the Criteria expressions. Set up your work to resemble the one in Figure B-17. Make sure the tables are joined on the common field, Employee ID.

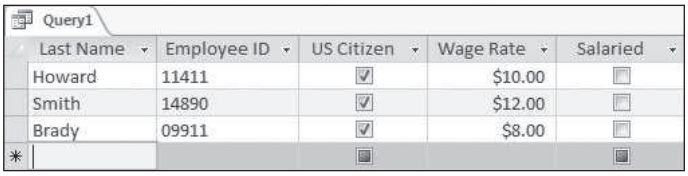


Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-17 A query based on two joined tables

You should quickly review the criteria you will need to set up this join: If you want data for employees who are U.S. citizens *and* who are hourly workers, the Criteria expressions go in the *same* Criteria row. If you want data for employees who are U.S. citizens *or* who are hourly workers, one of the expressions goes in the second Criteria row (the one with the or: notation).

Now run the query. The output should resemble the one in Figure B-18, with the exception of the name "Brady."



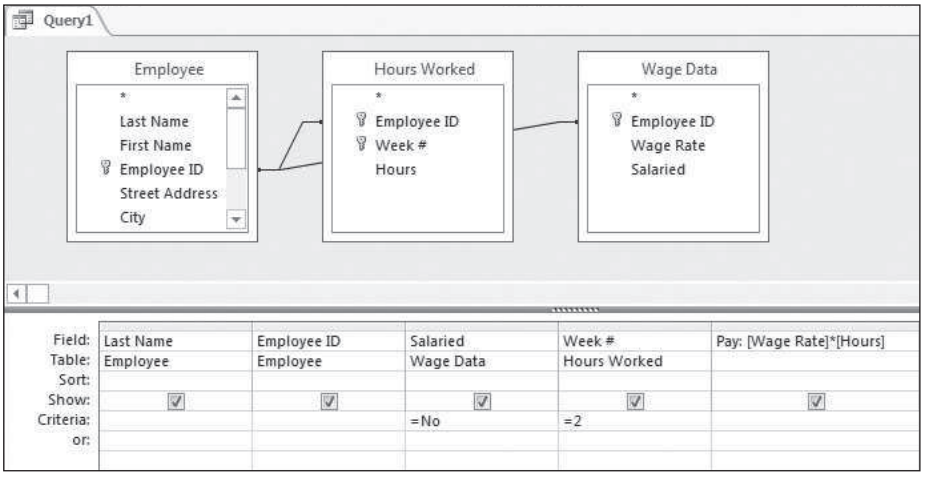
Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-18 Output of a query based on two joined tables

You do not need to print or save the query output, so return to Design view and close the query. Another practice query follows.

AT THE KEYBOARD

Suppose you want to see the wages owed to hourly employees for Week 2. You should show the last name, the employee ID, the salaried status, the week #, and the wages owed. Wages will have to be a calculated field ([Wage Rate] * [Hours]). The criteria are No for Salaried and 2 for the Week #. (This means that another "And" query is required.) Your query should be set up like the one in Figure B-19.



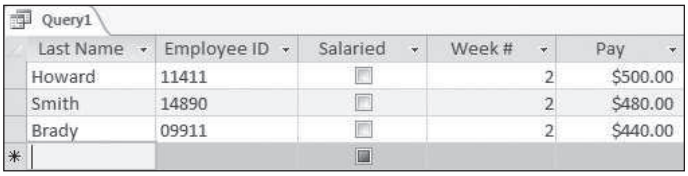
Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-19 Query setup for wages owed to hourly employees for Week 2

NOTE

In the query in Figure B-19, the calculated field column was widened so you could see the whole expression. To widen a column, click the column boundary line and drag to the right.

Run the query. The output should be similar to that in Figure B-20, if you formatted your calculated field to Currency.



Query1

Last Name	Employee ID	Salaried	Week #	Pay
Howard	11411	☐	2	$500.00
Smith	14890	☐	2	$480.00
Brady	09911	☐	2	$440.00
		☐		

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-20 Query output for wages owed to hourly employees for Week 2

Notice that it was not necessary to pull down the Wage Rate and Hours fields to make the query work. You do not need to save or print the query output, so return to Design view and close the query.

Summarizing Data from Multiple Records (Totals Queries)

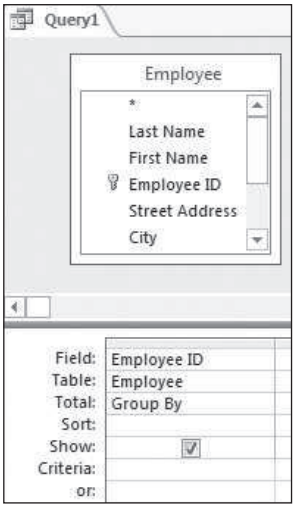
You may want data that summarizes values from a field for several records (or possibly all records) in a table. For example, you might want to know the average hours that all employees worked in a week or the total (sum) of all of the hours worked. Furthermore, you might want data grouped or stratified in some way. For example, you might want to know the average hours worked, grouped by all U.S. citizens versus all non-U.S. citizens. Access calls such a query a **Totals query**. These queries include the following operations:

Sum	The total of a given field's values
Count	A count of the number of instances in a field—that is, the number of records. In the current example, you would count the number of employee IDs to get the number of employees.
Average	The average of a given field's values
Min	The minimum of a given field's values
Var	The variance of a given field's values
StDev	The standard deviation of a given field's values
Where	The field has criteria for the query output

AT THE KEYBOARD

Suppose you want to know how many employees are represented in the example database. First, bring the Employee table into the QBE screen. Because you will need to count the number of employee IDs, which is a Totals query operation, you must bring down the Employee ID field.

To tell Access that you want a Totals query, click the Design tab and then click the Totals button in the Show/Hide group. A new row called the Total row opens in the lower part of the QBE screen. At this point, the screen resembles the one in Figure B-21.

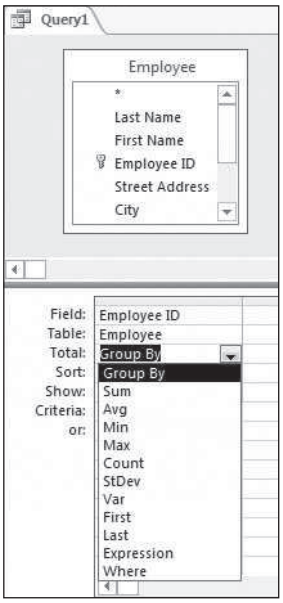


Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-21 Totals query setup

Note that the Total cell contains the words *Group By*. Until you specify a statistical operation, Access assumes that a field will be used for grouping (stratifying) data.

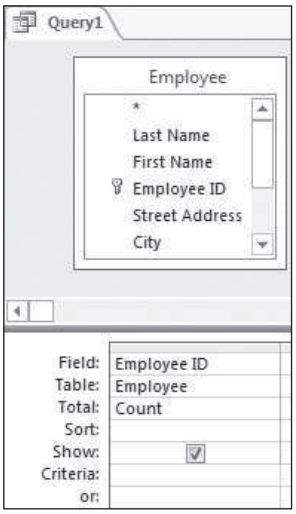
To count the number of employee IDs, click next to Group By to display an arrow. Click the arrow to reveal a drop-down menu, as shown in Figure B-22.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-22 Choices for statistical operation in a Totals query

Select the Count operator. (You might need to scroll down the menu to see the operator you want.) Your screen should resemble the one shown in Figure B-23.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

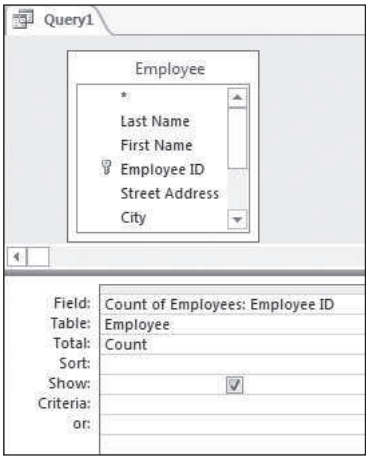
FIGURE B-23 Count in a Totals query

Run the query. Your output should resemble the one in Figure B-24.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-24 Output of Count in a Totals query

Notice that Access created a pseudo-heading, "CountOfEmployee ID," by splicing together the statistical operation (Count), the word Of, and the name of the field (Employee ID). If you wanted a phrase such as "Count of Employees" as a heading, you would go to Design view and change the query to resemble the one shown in Figure B-25.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-25 Heading change in a Totals query

When you run the query, the output should resemble the one in Figure B-26.

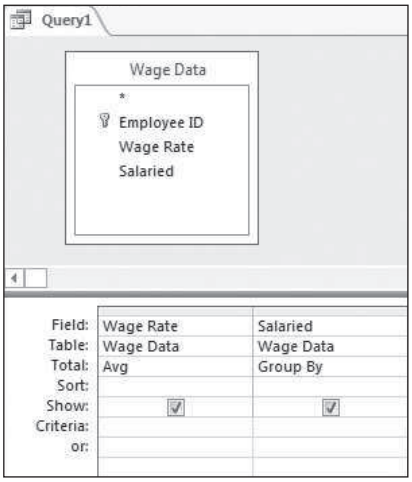
Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-26 Output of heading change in a Totals query

You do not need to print or save the query output, so return to Design view and close the query.

AT THE KEYBOARD

As another example of a Totals query, suppose you want to know the average wage rate of employees, grouped by whether the employees are salaried. Figure B-27 shows how to set up your query.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

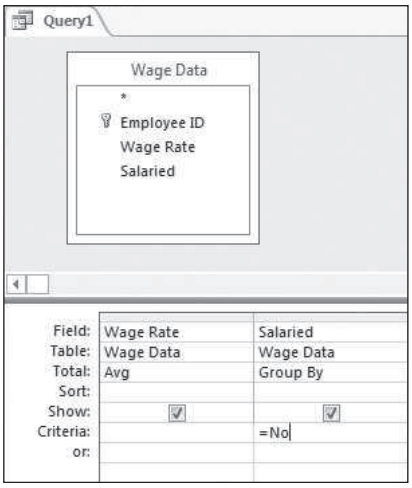
FIGURE B-27 Query setup for average wage rate of employees

When you run the query, your output should resemble the one in Figure B-28.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-28 Output of query for average wage rate of employees

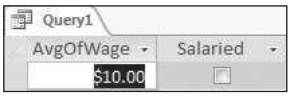
Recall the convention that salaried workers are assigned zero dollars an hour. Suppose you want to eliminate the output line for zero dollars an hour because only hourly-rate workers matter for the query. The query setup is shown in Figure B-29.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-29 Query setup for nonsalaried workers only

When you run the query, you will get output for nonsalaried employees only, as shown in Figure B-30.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

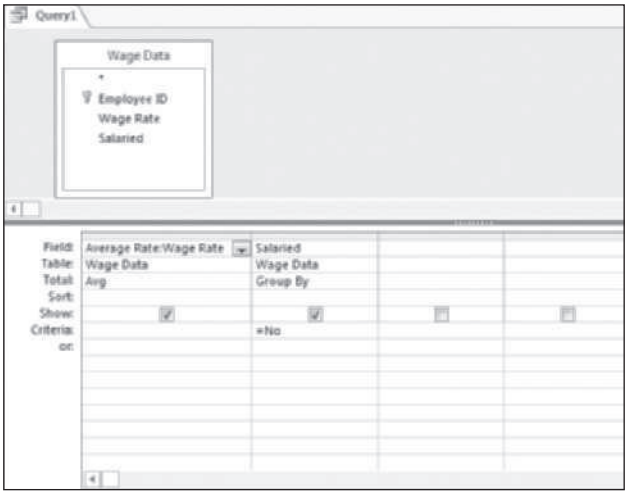
FIGURE B-30 Query output for nonsalaried workers only

Thus, it is possible to use Criteria in a Totals query, just as you would with a "regular" query. You do not need to print or save the query output, so return to Design view and close the query.

AT THE KEYBOARD

Assume that you want to see two pieces of information for hourly workers: (1) the average wage rate, which you will call Average Rate in the output, and (2) 110 percent of the average rate, which you will call the Increased Rate. To get this information, you can make a calculated field in a new query from a Totals query. In other words, you use one query as a basis for another query.

Create the first query; you already know how to perform certain tasks for this query. The revised heading for the average rate will be Average Rate, so type *Average Rate: Wage Rate* in the Field cell. Note that you want the average of this field. Also, the grouping will be by the Salaried field. (To get hourly workers only, enter *Criteria: No*.) Confirm that your query resembles the one in Figure B-31, then save the query and close it.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-31 A totals query with average

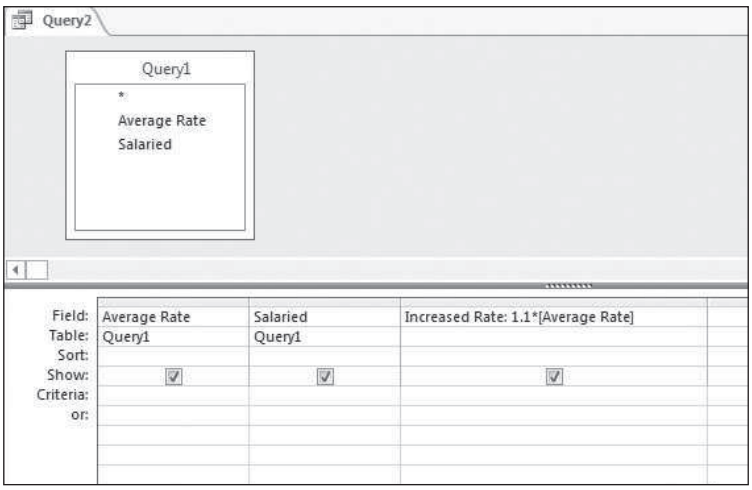
Now begin a new query. However, instead of bringing in a table to the query design, select a query. To start a new query, click the Create tab and then click the Query Design button in the Queries group. The Show Table window appears. Click the Queries tab instead of using the default Tables tab, and select the query you just saved as a basis for the new query. The most difficult part of this query is to construct the expression for the calculated field. Conceptually, it is as follows:

Increased Rate: 1.1 * [The current average]

You use the new field name in the new query as the current average, and you treat the new name like a new field:

Increased Rate: 1.1 * [Average Rate]

The query within a query is shown in Figure B-32.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-32 A query within a query

Figure B-33 shows the output of the new query. Note that the calculated field is formatted.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-33 Output of an Expression in a Totals query

You do not need to print or save the query output, so return to Design view and close the query.

Using the Date() Function in Queries

Access has two important date function features:

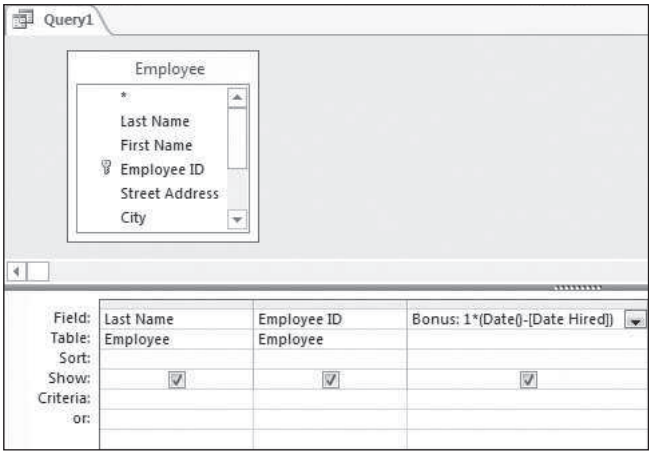
- The built-in Date() function gives you today's date. You can use the function in query criteria or in a calculated field. The function "returns" the day on which the query is run; in other words, it inserts the value where the Date() function appears in an expression.
- Date arithmetic lets you subtract one date from another to obtain the difference—in number of days—between two calendar dates. For example, suppose you create the following expression:
  10/9/2012 – 10/4/2012
  Access would evaluate the expression as the integer 5 (9 minus 4 is 5).

As another example of how date arithmetic works, suppose you want to give each employee a one-dollar bonus for each day the employee has worked. You would need to calculate the number of days between the employee's date of hire and the day the query is run, and then multiply that number by $1.

You would find the number of elapsed days by using the following equation:

Date() – [Date Hired]

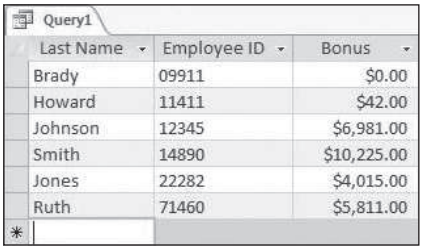
Also suppose that for each employee, you want to see the last name, employee ID, and bonus amount. You would set up the query as shown in Figure B-34.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-34 Date arithmetic in a query

Assume that you set the format of the Bonus field to Currency. The output will be similar to that in Figure B-35, although your Bonus data will be different because you used a different date.



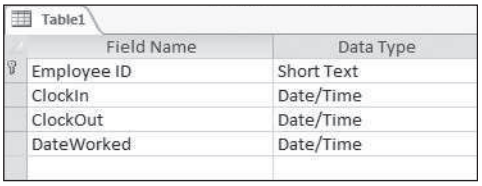
Last Name	Employee ID	Bonus
Brady	09911	$0.00
Howard	11411	$42.00
Johnson	12345	$6,981.00
Smith	14890	$10,225.00
Jones	22282	$4,015.00
Ruth	71460	$5,811.00

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-35 Output of query with date arithmetic

Using Time Arithmetic in Queries

Access also allows you to subtract the values of time fields to get an elapsed time. Assume that your database has a Job Assignments table showing the times that nonsalaried employees were at work during a day. The definition is shown in Figure B-36.

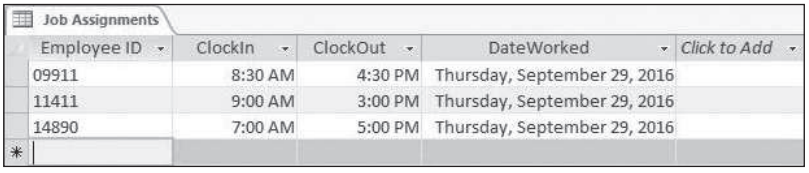


Field Name	Data Type
Employee ID	Short Text
ClockIn	Date/Time
ClockOut	Date/Time
DateWorked	Date/Time

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-36 Date/Time data definition in the Job Assignments table

Assume that the DateWorked field is formatted for Long Date and that the ClockIn and ClockOut fields are formatted for Medium Time. Also assume that for a particular day, nonsalaried workers were scheduled as shown in Figure B-37.

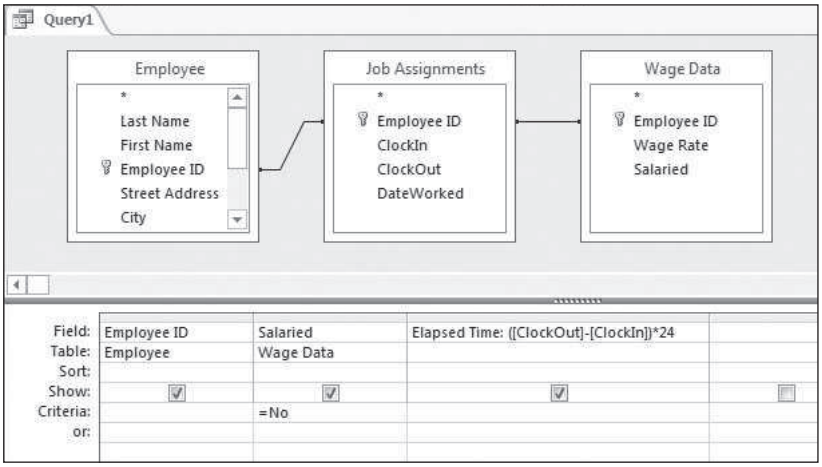


Employee ID	ClockIn	ClockOut	DateWorked	Click to Add
09911	8:30 AM	4:30 PM	Thursday, September 29, 2016	
11411	9:00 AM	3:00 PM	Thursday, September 29, 2016	
14890	7:00 AM	5:00 PM	Thursday, September 29, 2016	

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-37 Display of date and time in a table

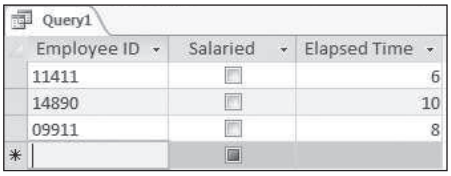
You want a query showing the elapsed time that your employees were on the premises for the day. When you add the tables, your screen may show the links differently. Click and drag the Job Assignments, Employee, and Wage Data table icons to look like those in Figure B-38.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-38 Query setup for time arithmetic

Figure B-39 shows the output, which looks correct. For example, employee 09911 was at work from 8:30 a.m. to 4:30 p.m., which is eight hours. But how does the odd expression that follows yield the correct answers?



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-39 Query output for time arithmetic

([ClockOut] – [ClockIn]) * 24

Why wouldn't the following expression work?

[ClockOut] – [ClockIn]

Here is the answer: In Access, subtracting one time from the other yields the *decimal* portion of a 24-hour day. Returning to the example, you can see that employee 09911 worked eight hours, which is one-third of a day, so the time arithmetic function yields .3333. That is why you must multiply by 24—to convert from decimals to an hourly basis. Hence, for employee 09911, the expression performs the following calculation: $1/3 \times 24 = 8$.

Note that parentheses are needed to force Access to do the subtraction *first*, before the multiplication. Without parentheses, multiplication takes precedence over subtraction. For example, consider the following expression:

[ClockOut] – [ClockIn] * 24

In this example, ClockIn would be multiplied by 24, the resulting value would be subtracted from ClockOut, and the output would be a nonsensical decimal number.

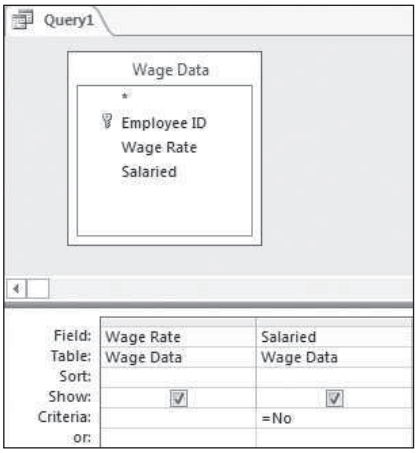
Deleting and Updating Queries

The queries presented in this tutorial so far have been Select queries. They select certain data from specific tables based on a given criterion. You also can create queries to update the original data in a database. Businesses use such queries often, and in real time. For example, when you order an item from a Web site, the company's database is updated to reflect your purchase through the deletion of that item from the company's inventory.

Consider an example. Suppose you want to give all nonsalaried workers a $0.50 per hour pay raise. Because you have only three nonsalaried workers, it would be easy to change the Wage Rate data in the table. However, if you had 3,000 nonsalaried employees, it would be much faster and more accurate to change the Wage Rate data by using an Update query that adds $0.50 to each nonsalaried employee's wage rate.

AT THE KEYBOARD

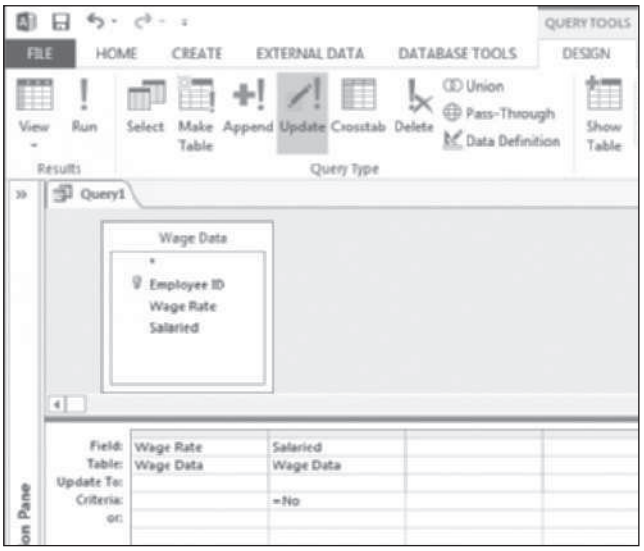
Now you will change each of the nonsalaried employees' pay via an Update query. Figure B-40 shows how to set up the query.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-40 Query setup for an Update query

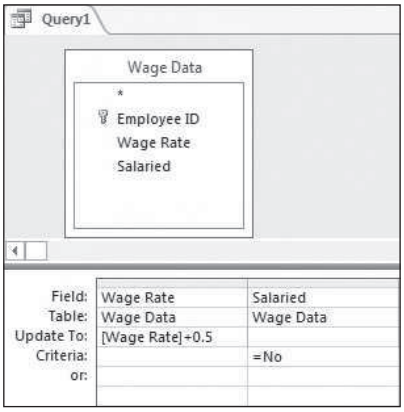
So far, this query is just a Select query. Click the Update button in the Query Type group, as shown in Figure B-41.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-41 Selecting a query type

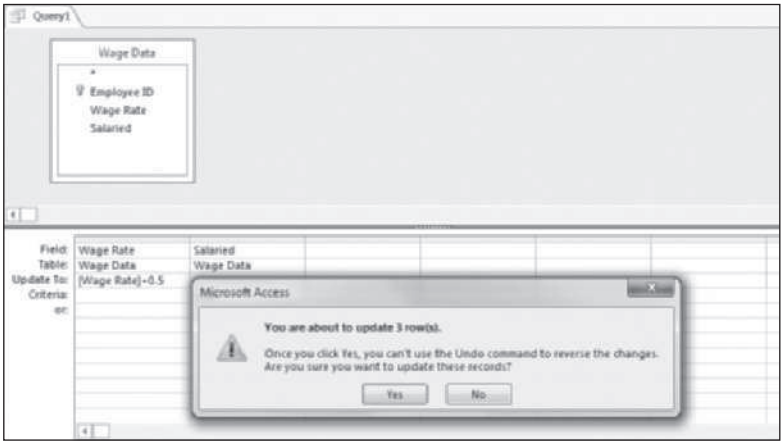
Notice that you now have another line on the QBE grid called Update To:, which is where you specify the change or update the data. Notice that you will update only the nonsalaried workers by using a filter under the Salaried field. Update the Wage Rate data to Wage Rate plus $0.50, as shown in Figure B-42. Note that the update involves the use of brackets [ ], as in a calculated field.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-42 Updating the wage rate for nonsalaried workers

Now run the query by clicking the Run button in the Results group. If you cannot run the query because it is blocked by Disabled Mode, click the Enable Content button on the Security Warning message bar. When you successfully run the query, the warning message in Figure B-43 appears.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-43 Update query warning

When you click Yes, the records are updated. Check the updated records by viewing the Wage Data table. Each nonsalaried wage rate should be increased by $0.50. You could add or subtract data from another table as well. If you do, remember to put the field name in square brackets.

Another type of query is the Delete query, which works like Update queries. For example, assume that your company has been purchased by the state of Delaware, which has a policy of employing only state residents. Thus, you must delete (or fire) all employees who are not exclusively Delaware residents.

To do that, you would create a Select query. Using the Employee table, you would click the Delete button in the Query Type group, then bring down the State field and filter only those records that were not in Delaware (DE). Do not perform the operation, but note that if you did, the setup would look like the one in Figure B-44.

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

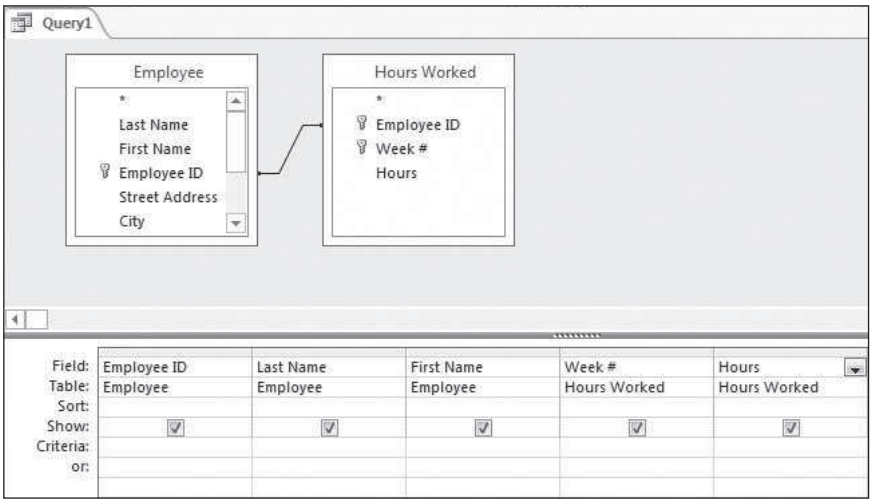
FIGURE B-44 Deleting all employees who are not Delaware residents

Using Parameter Queries

A **Parameter query** is actually a type of Select query. For example, suppose your company has 5,000 employees and you want to query the database to find the same kind of information repeatedly, but about different employees each time. For example, you might want to know how many hours a particular employee has worked. You could run a query that you created and stored previously, but run it only for a particular employee.

AT THE KEYBOARD

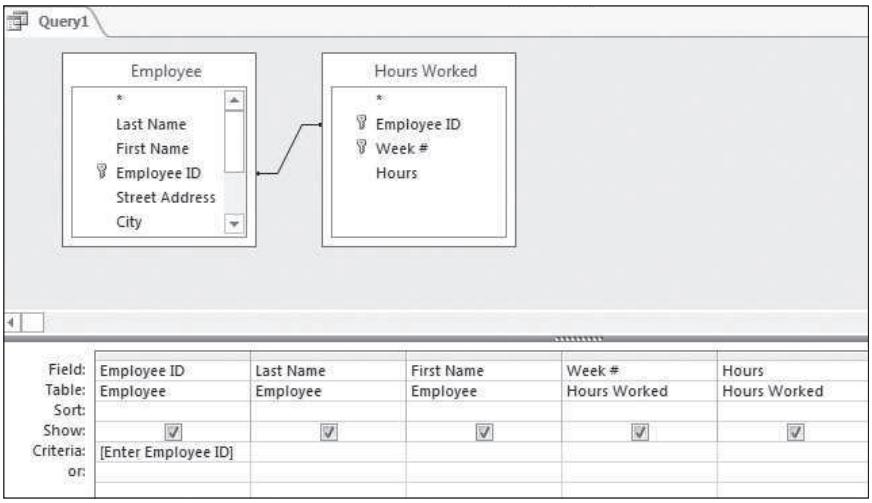
Create a Select query with the format shown in Figure B-45.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-45 Design of a Parameter query beginning as a Select query

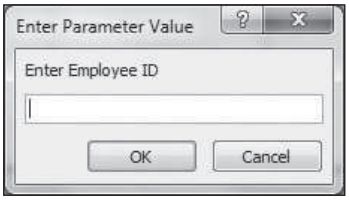
In the Criteria line of the QBE grid for the Employee ID field, type what is shown in Figure B-46.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-46 Design of a Parameter query, continued

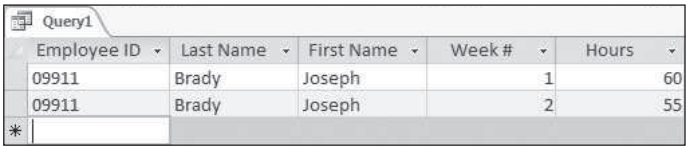
Note that the Criteria line uses square brackets, as you would expect to see in a calculated field. Now run the query. You will be prompted for the employee's ID number, as shown in Figure B-47.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-47 Enter Parameter Value window

Enter your own employee ID. Your query output should resemble the one in Figure B-48.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-48 Output of a Parameter query

MAKING SEVEN PRACTICE QUERIES

This portion of the tutorial gives you additional practice in creating queries. Before making these queries, you must create the specified tables and enter the records shown in the "Creating Tables" section of this tutorial. The output shown for the practice queries is based on those inputs.

AT THE KEYBOARD

For each query that follows, you are given a problem statement and a "scratch area." You also are shown what the query output should look like. Set up each query in Access and then run the query. When you are satisfied with the results, save the query and continue with the next one. Note that you will work with the Employee, Hours Worked, and Wage Data tables.

1. Create a query that shows the employee ID, last name, state, and date hired for employees who live in Delaware *and* were hired after 12/31/99. Perform an ascending sort by employee ID. First

click the Sort cell of the field, and then choose Ascending or Descending. Before creating your query, use the table shown in Figure B-49 to work out your QBE grid on paper.

Field					
Table					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-49 QBE grid template

Your output should resemble the one in Figure B-50.

Query 1

Employee ID	Last Name	State	Date Hired
11411	Howard	DE	6/1/2016
22282	Jones	DE	7/15/2004

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-50 Number 1 query output

2. Create a query that shows the last name, first name, date hired, and state for employees who live in Delaware *or* were hired after 12/31/99. The primary sort (ascending) is on last name, and the secondary sort (ascending) is on first name. The Primary Sort field must be to the left of the Secondary Sort field in the query setup. Before creating your query, use the table shown in Figure B-51 to work out your QBE grid on paper.

Field					
Table					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-51 QBE grid template

If your name was Joseph Brady, your output would look like the one in Figure B-52.

Query 2

Last Name	First Name	Date Hired	State
Brady	Joseph	9/15/2016	MD
Howard	Jane	6/1/2016	DE
Johnson	John	6/1/1996	DE
Jones	Sue	7/15/2004	DE
Smith	Albert	7/15/1987	DE

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-52 Number 2 query output

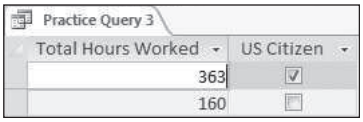
3. Create a query that sums the number of hours worked by U.S. citizens and the number of hours worked by non-U.S. citizens. In other words, create two sums, grouped on citizenship. The heading for total hours worked should be Total Hours Worked. Before creating your query, use the table shown in Figure B-53 to work out your QBE grid on paper.

Field					
Table					
Total					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-53 QBE grid template

Your output should resemble the one in Figure B-54.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-54 Number 3 query output

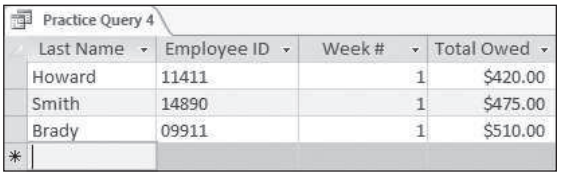
4. Create a query that shows the wages owed to hourly workers for Week 1. The heading for the wages owed should be Total Owed. The output headings should be Last Name, Employee ID, Week #, and Total Owed. Before creating your query, use the table shown in Figure B-55 to work out your QBE grid on paper.

Field					
Table					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-55 QBE grid template

If your name was Joseph Brady, your output would look like the one in Figure B-56.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-56 Number 4 query output

5. Create a query that shows the last name, employee ID, hours worked, and overtime amount owed for hourly employees who earned overtime during Week 2. Overtime is paid at 1.5 times the normal hourly rate for all hours worked over 40. Note that the amount shown in the query should be just the overtime portion of the wages paid. Also, this is not a Totals query—amounts should be shown for individual workers. Before creating your query, use the table shown in Figure B-57 to work out your QBE grid on paper.

Field					
Table					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-57 QBE grid template

If your name was Joseph Brady, your output would look like the one in Figure B-58.

Practice Query 5

Last Name	Employee ID	Hours	OT Pay
Howard	11411	50	$157.50
Brady	09911	55	$191.25

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-58 Number 5 query output

6. Create a Parameter query that shows the hours employees have worked. Have the Parameter query prompt for the week number. The output headings should be Last Name, First Name, Week #, and Hours. This query is for nonsalaried workers only. Before creating your query, use the table shown in Figure B-59 to work out your QBE grid on paper.

Field					
Table					
Sort					
Show					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-59 QBE grid template

Run the query and enter 2 when prompted for the week number. Your output should look like the one in Figure B-60.

Practice Query 6

Last Name	First Name	Week #	Hours
Howard	Jane	2	50
Smith	Albert	2	40
Brady	Joseph	2	55

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-60 Number 6 query output

7. Create an Update query that gives certain workers a merit raise. First, you must create an additional table, as shown in Figure B-61.

Merit Raises

Employee ID	Merit Raise	Click to Add
11411	$0.25	
14890	$0.15	
	$0.00	

Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-61 Merit Raises table

Create a query that adds the Merit Raise to the current Wage Rate for employees who will receive a raise. When you run the query, you should be prompted with *You are about to update two rows.* Check the original Wage Data table to confirm the update. Before creating your query, use the table shown in Figure B-62 to work out your QBE grid on paper.

Field					
Table					
Update to					
Criteria					
Or:					

Source: © 2016 Cengage Learning®

FIGURE B-62 QBE grid template

CREATING REPORTS

Database packages let you make attractive management reports from a table's records or from a query's output. If you are making a report from a table, the Access report generator looks up the data in the table and puts it into report format. If you are making a report from a query's output, Access runs the query in the background (you do not control it or see it happen) and then puts the output in report format.

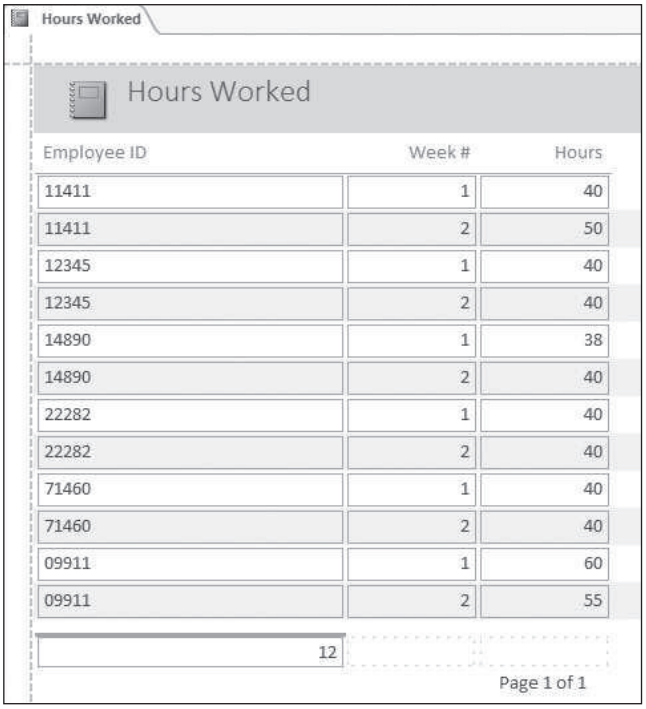
There are different ways to make a report. One method is to create one from scratch in Design view, but this tedious process is not explained in this tutorial. A simpler way is to select the query or table on which the report is based and then click Report on the Create tab. This streamlined method of creating reports is explained in this tutorial.

Creating a Grouped Report

This tutorial assumes that you already know how to create a basic ungrouped report, so this section teaches you how to make a grouped report. If you do not know how to create an ungrouped report, you can learn by following the first example in the upcoming section.

AT THE KEYBOARD

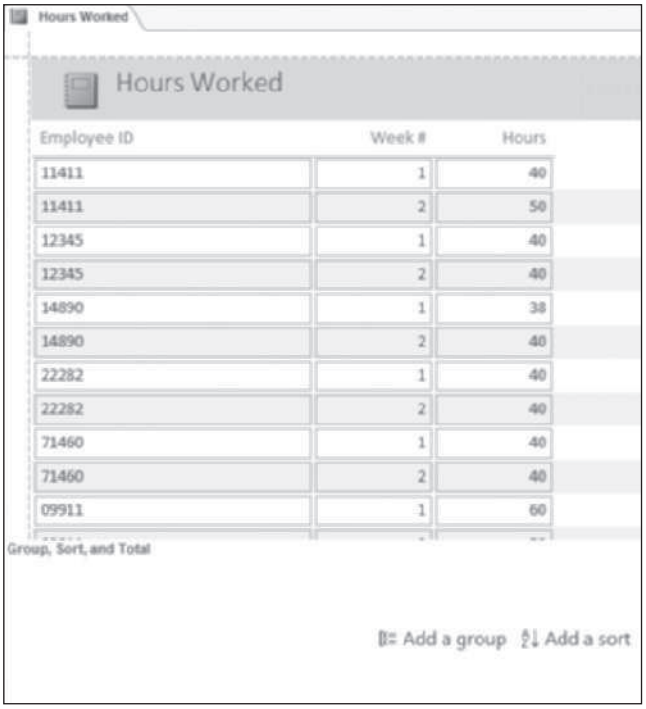
Suppose you want to create a report from the Hours Worked table. Select the table by clicking it once. Click the Create tab, then click Report in the Reports group. A report appears, as shown in Figure B-63.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-63 Initial report based on a table

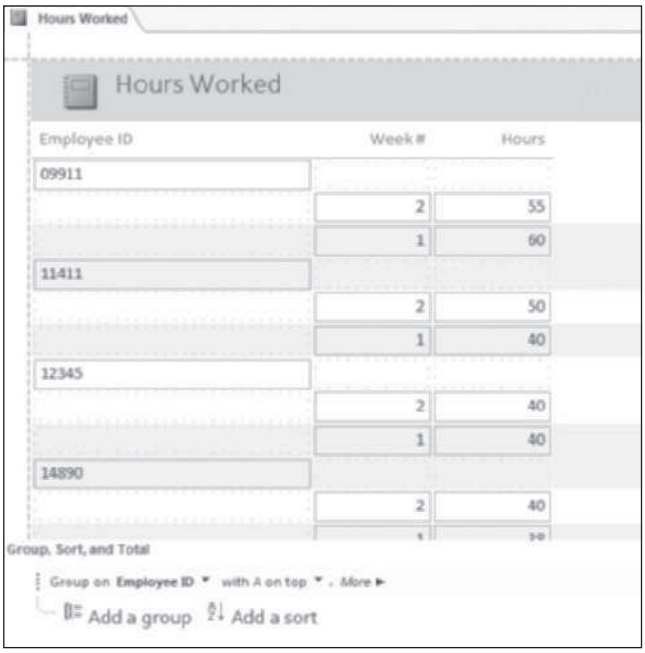
On the Design tab, select the Group & Sort button in the Grouping & Totals group. Your report will have an additional selection at the bottom, as shown in Figure B-64.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-64 Report with grouping and sorting options

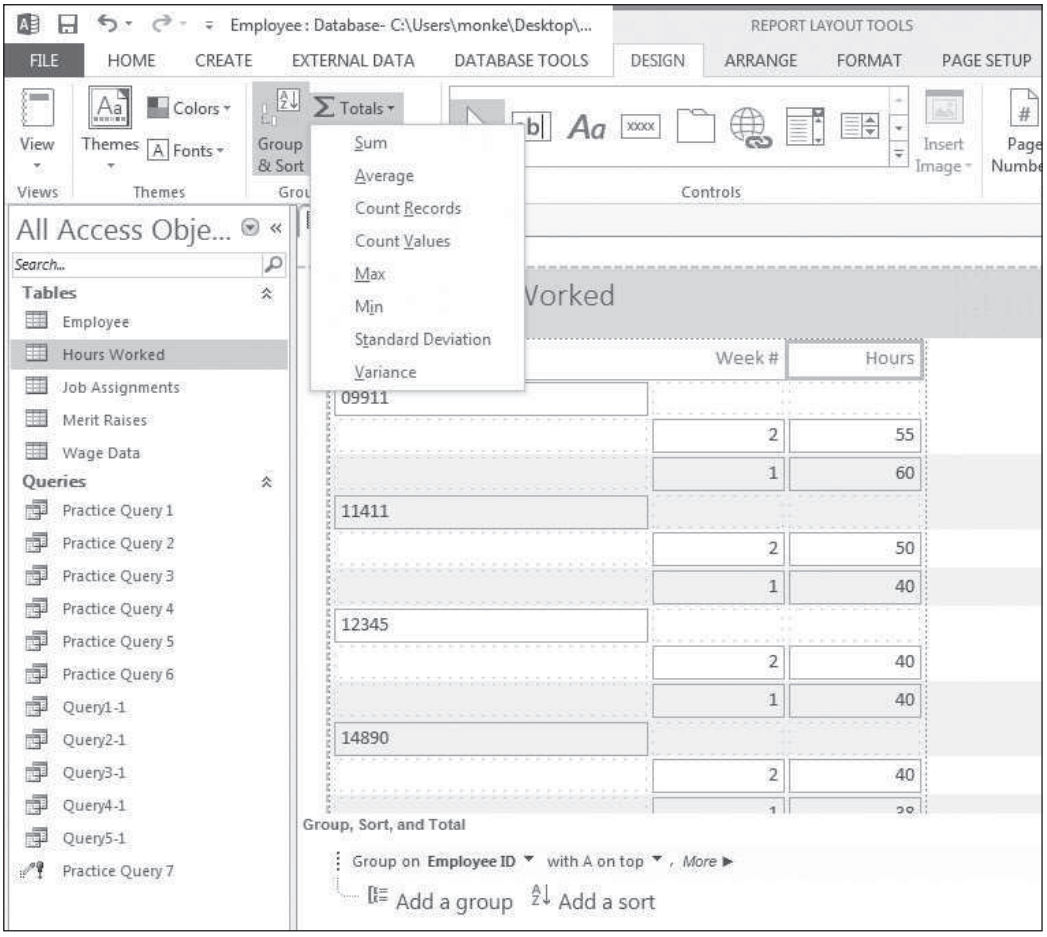
Click the Add a group button at the bottom of the report, and then select Employee ID. Your report will be grouped as shown in Figure B-65.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-65 Grouped report

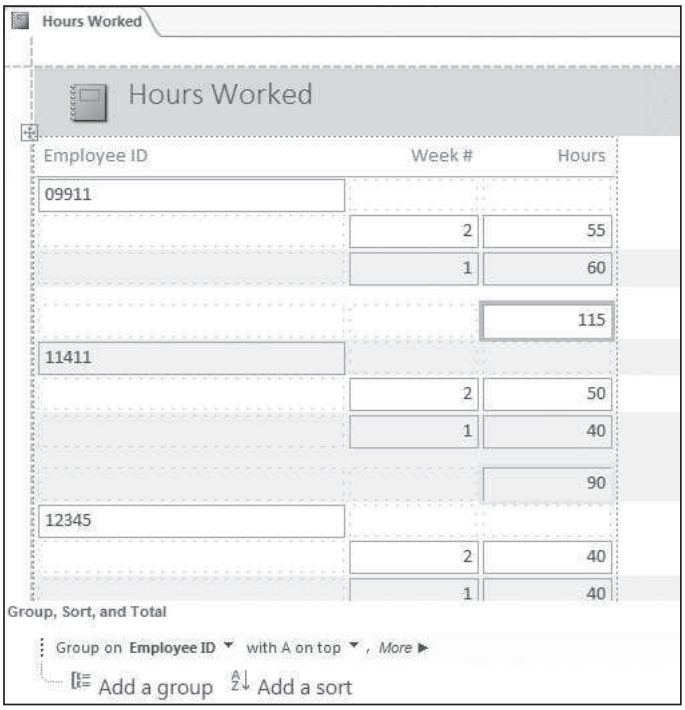
To complete this report, you need to total the hours for each employee by selecting the Hours column heading. Your report will show that the entire column is selected. On the Design tab, click the Totals button in the Grouping & Totals group, and then choose Sum from the menu, as shown in Figure B-66.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-66 Totaling the hours

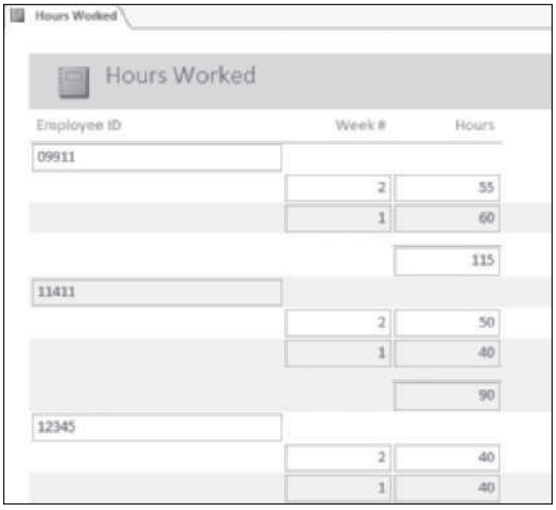
Your report will look like the one in Figure B-67.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-67 Completed report

Your report is currently in Layout view. To see how the final report looks when printed, click the Design tab and select Report View from the Views group. Your report looks like the one in Figure B-68, although only a portion is shown in the figure.



Source: Microsoft product screenshots used with permission from Microsoft Corporation.

FIGURE B-68 Report in Report view

NOTE

To change the picture or logo in the upper-left corner of the report when in Layout view, click the notebook symbol and press the Delete key. You can insert a logo in place of the notebook by clicking the Design tab and then clicking the Insert Image button in the Controls group.