INSTRUCTOR'S MANUAL

TO ACCOMPANY

DAVID M. KROENKE AND DAVID J. AUER

Database Processing

Fundamentals, Design, and Implementation

13th Edition

CHAPTER TWO

INTRODUCTION TO STRUCTURE QUERY LANGUAGE

Prepared By

David J. Auer

Western Washington University

CHAPTER OBJECTIVES

- To understand the use of extracted data sets
- To understand the use of ad-hoc queries
- To understand the history and significance of Structured Query Language (SQL)
- To understand the basic SQL SELECT/FROM/WHERE framework as the basis for database queries
- To be able to write queries in SQL to retrieve data from a single table
- To be able to write queries in SQL to use the SQL SELECT, FROM, WHERE, ORDER BY, GROUP BY, and HAVING clauses
- To be able to write queries in SQL to use SQL DISTINCT, AND, OR, NOT, BETWEEN, LIKE, and IN keywords
- To be able to use the SQL built-in functions of SUM, COUNT, MIN, MAX, and AVG with and without the use of a GROUP BY clause
- To be able to write queries in SQL to retrieve data from a single table but restricting the data based upon data in another table (subquery)
- To create SQL queries that retrieve data from multiple tables using the SQL join and JOIN ON operations
- To create SQL queries that retrieve data from multiple tables using the SQL OUTER JOIN operation

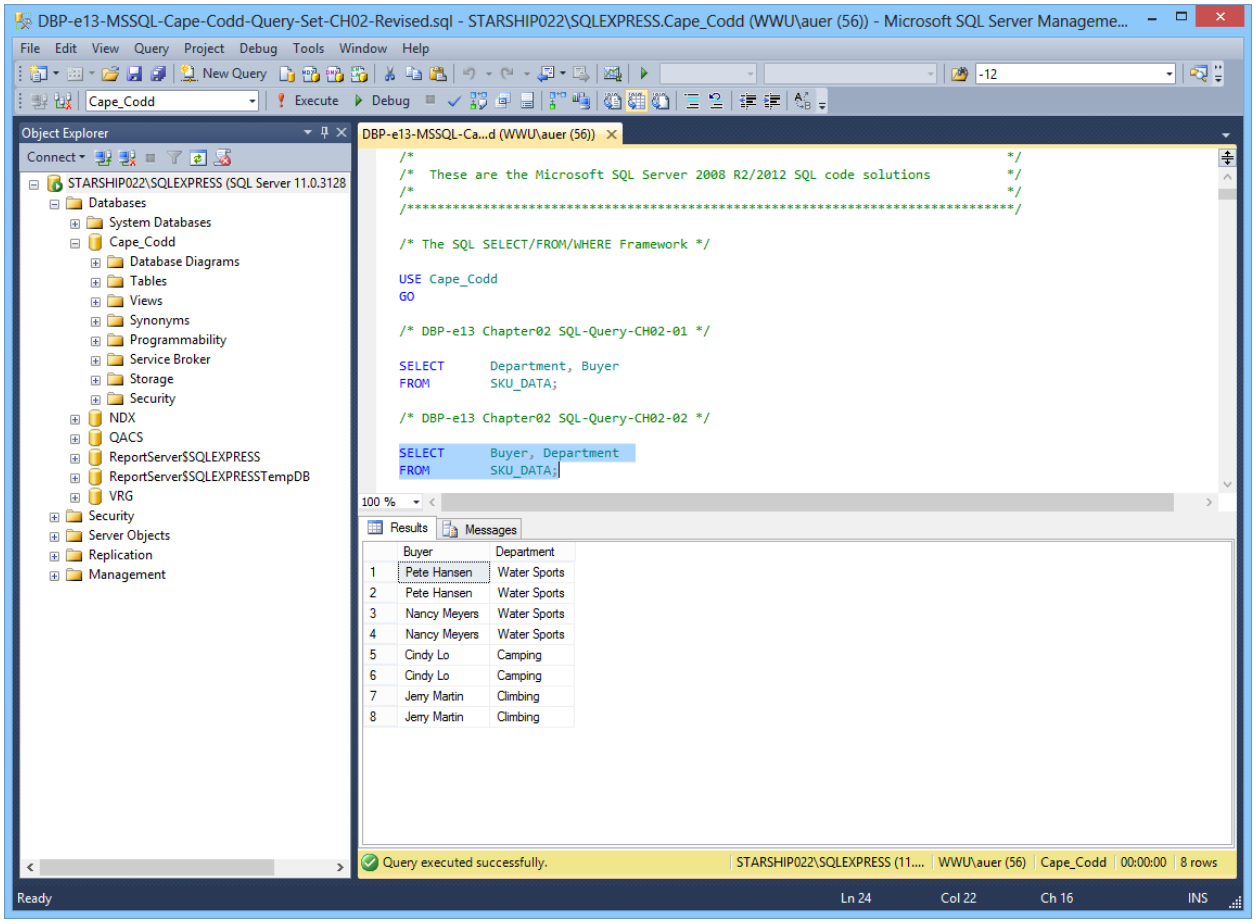
ERRATA

There are no known errors at this time. Any errors that are discovered in the future will be reported and corrected in the Online DBP e13 Errata document, which will be available at http://www.pearsonhighered.com/kroenke.

TEACHING SUGGESTIONS

- Database files to illustrate the examples in the chapter and solution database files for your use are available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).
- The best way for students to understand SQL is by using it. Have your students work through the Review Questions, Project Questions and the Marcia's Dry Cleaning and Morgan Importing Project Questions in an actual database. Students can create databases in Microsoft Access with basic tables, relationships and data from the material in the book. SQL scripts for Microsoft SQL Server, Oracle Database and MySQL versions of Cape Codd, WPC, MDC and MI are available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

- Microsoft Access database files for Cape Codd and the NASDAQ data (NDX.accdb), together with SQL scripts for Microsoft SQL Server, Oracle Database and MySQL versions of Cape Codd, MDC and MI are available for student use in the Student Resources on the text's Web site (www.pearsonhighered.com/kroenke).
- The SQL processors in the various DBMSs are very fussy about character sets used for SQL statements. They want to see plain ASCII text, not fancy fonts. This is particularly true of the single quotation (') used to designate character strings, but I've also had problems with the minus sign. If your students are having problems getting a "properly structured SQL statement" to run, look closely for this type of problem.
- There is a useful teaching technique which will allow you to demonstrate the SQL queries in the text using Microsoft SQL Server if you have it available.
    - Open the Microsoft SQL Server Management Studio, and create a new SQL Server database named Cape-Codd.
    - In the Microsoft SQL Server Management Studio, use the SQL statements in the *.sql text file *DBP-e13-MSSQL-Cape-Codd-Create-Tables.sql* to create the RETAIL_ORDER, ORDER_ITEM and SKU_DATA tables [the WAREHOUSE and INVENTORY tables, used in the Review Questions, are also created].
    - In the Microsoft SQL Server Management Studio, use the SQL statements *.sql text file *DBP-e13-MSSQL-Cape-Dodd-Insert-Data.sql* to populate the RETAIL_ORDER, ORDER_ITEM and SKU_DATA tables [the WAREHOUSE and INVENTORY tables, used in the Review Questions, are also populated].
    - In the Microsoft SQL Server Management Studio, open the *.sql text file *DBP-e13-MSSQL-Cape-Codd-Query-Set-CH02.sql*. This file contains all the queries shown in the Chapter 2 text.
    - Highlight the query you want to run and click the Execute Query button to display the results of the query. An example of this is shown in the following screenshot.
    - All of the *.sql text files needed to do this are available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).



- Microsoft Access 2013 does not support all SQL-92 (and newer) constructs. While this chapter still considers Microsoft Access as the DBMS most likely to be used by students at this point in the course, there are some Review Questions and Project Questions that use the ORDER BY clause with aliased computed columns that will not run in Access (see Review Questions 2.42 – 2.44 and Project Questions 2.63.e – 2.63.g). The correct solutions for these questions were obtained using Microsoft SQL Server 2012. The Microsoft Access results without the ORDER BY clause are also shown, so you can assign these problems without the ORDER BY part of the questions.
- Microsoft Access 2013 does not support SQL wildcard characters (see Review Questions 2.36 – 2.38), although it does have equivalent wildcard characters as described in the chapter. The correct solutions for these questions were obtained using Microsoft SQL Server 2012.
- For those students who are used to procedural languages, they may have some initial difficulty with a language that does set processing like SQL. These students are accustomed to processing rows (records) rather than sets. It is time well spent to make sure they understand that SQL processes tables at a time, not rows at a time.
- Students may have some trouble understanding the GROUP BY clause. If you can explain it in terms of traditional control break logic (sort rows on a key then process the rows until the value of the key changes), they will have less trouble.

This also explains why the GROUP BY clause will present the rows sorted even though you do not use an ORDER BY clause.

- At this point, students familiar with Microsoft Access will wonder why they are learning SQL. They have made queries in Microsoft Access using Microsoft Access's version of Query-By-Example (QBE), and therefore never had to understand the SQL. In many cases, they will not know that Microsoft Access generates SQL code when you create a query in design view. It is worth letting them know this is done and even showing them the SQL created for and underlying a Microsoft Access query.
- It is also important for students to understand that, in many cases, the Query-By-Example forms such as Microsoft Access' design view can be very inefficient. Also, the QBE forms are not available from within an application program such as Java or C, and so SQL must be written.
- It has been our experience that a review of a Cartesian Product from an algebra class is time well spent. Show students what will happen if a WHERE statement is left off of a join. The following example will work. Assume you create four tables with five columns each and 100 rows each. How many columns and rows will be displayed by the statement:

  ```
  SELECT * FROM TABLE1, TABLE2, TABLE3, TABLE4;
  ```

  The result is 20 columns (not bad) but 100,000,000 rows (100 * 100 = 10,000, 10,000 * 100 = 1,000,000, 1,000,000 * 100 = 100,000,000). This happens because the JOIN is not qualified. If they understand Cartesian products then they will understand how to fix a JOIN where the results are much too large.
- Note that in the Marcia's Dry Cleaning project, where in some previous editions we have used tables named ORDER and ORDER_ITEM, we have changed these table names to INVOICE and INVOICE_ITEM. We did this because ORDER is an SQL reserved word (part of ORDER BY). Therefore, when the table name ORDER is used as part of a query, it may need to be ("must be" in Access 2013) enclosed in delimiters as [ORDER] if the query is going to run correctly. The topic of reserved words and delimiters is discussed in more detail in Chapters 6 and 7. However, now is a good time to introduce it to your students.
- Note that Microsoft Access SQL requires the INNER JOIN syntax instead of the standard SQL syntax JOIN used by Microsoft SQL Server, Oracle Database and MySQL

ANSWERS TO REVIEW QUESTIONS

2.1 What is a business intelligence (BI) system?

A business intelligence (BI) system, is a system used to support management decisions by producing information for assessment, analysis, planning and control.

2.2 What is an ad-hoc query?

An ad-hoc query is a query created by the user as needed, rather than a query programmed into an application.

2.3 What does SQL stand for, and what is SQL?

SQL stands for *Structured Query Language*. SQL is the universal query language for relational DBMS products.

2.4 What does SKU stand for, and what is an SKU?

SKU stands for stock keeping unit. An SKU is a an identifier used to label and distinguish each item sold by a business.

2.5 Summarize how data were altered and filtered in creating the Cape Codd data extraction.

Data from the Cape Codd operational retail sales database were used to create a retail sales extraction database with three tables: RETAIL_ORDER, ORDER_ITEM and SKU_DATA.

The **RETAIL_ORDER** table uses only a few of the columns in the operational database. The structure of the table is:

RETAIL_ORDER (OrderNumber, StoreNumber, StoreZip, OrderMonth, OrderYear, OrderTotal)

For this table, the original column OrderDate (in the data format MM/DD/YYYY [04/26/2013]) was converted into the columns OrderMonth (in a Character(12) format so that each month is spelled out [April]) and OrderYear (in an Integer format with each year appearing as a four-digit year [2013]).

We also note that the OrderTotal column includes tax, shipping and other charges that do not appear in the data extract. Thus, it does not equal the sum of the related ExtendedPrice column in the ORDER_ITEM table discussed below.

The **ORDER_ITEM** table uses an extract of the items purchased for each order. The structure of the table is:

ORDER_ITEM (OrderNumber, SKU, Quantity, Price, ExtendedPrice)

For this table, there is one row for each SKU associated with a given OrderNumber, representing one row for each type of item purchased in a specific order.

The **SKU_DATA** table uses an extract of the item identifying and describing data in the complete operational table. The structure of the table is:

SKU_DATA (SKU, SKU_Description, Department, Buyer)

For this table, there is one row to describe each SKU, representing one particular item that is sold by Cape Codd.

2.6 Explain, in general terms, the relationships of the RETAIL_ORDER, ORDER_ITEM, and SKU_DATA tables.

In general, each sale in RETAIL_ORDER relates to one or more rows in ORDER_ITEM that detail the items sold in the specific order. Each row in ORDER_ITEM is associated with a specific SKU in the SKU_DATA table. Thus one SKU may be associated once with each specific order number, but may also be associated with many different order numbers (as long as it appears only once in each order).

Using the Microsoft Access Relationship window, the relationships (including the additional relationships with the INVENTORY and WAREHOUSE tables described after Review Question 2.15) are shown in Figure 2-24 and look like this:

Figure 2-24
The Cape Codd Database with the WAREHOUSE and INVENTORY Tables

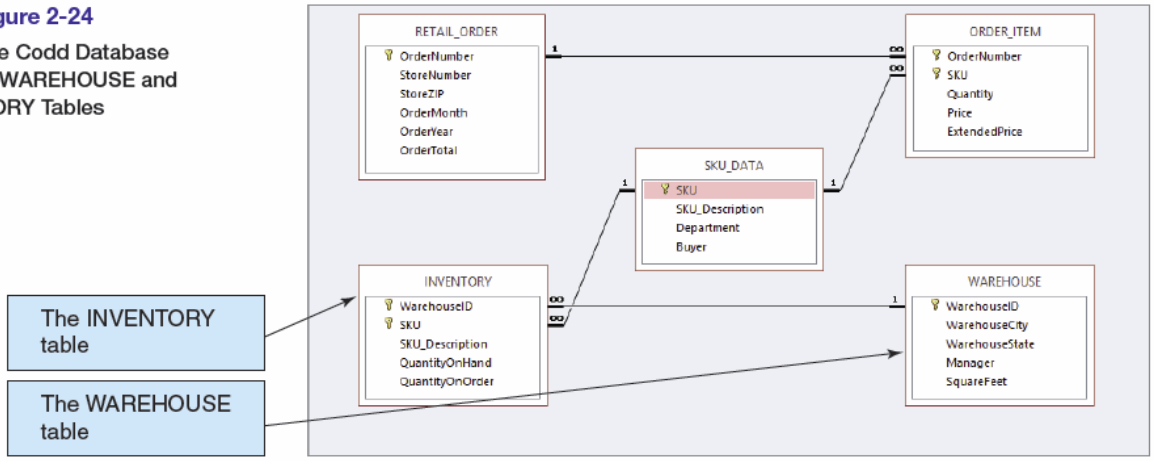


Figure 2-23 – The Cape Codd Database with the WAREHOUSE and INVENTORY tables

In traditional database terms (which will be discussed in Chapter 6) OrderNumber and SKU in ORDER_ITEM are foreign keys that provide the links to the RETAIL_ORDER and SKU_DATA tables respectively. Using an underline to show primary keys and italics to show foreign keys, the tables and their relationships are shown as:

RETAIL_ORDER (OrderNumber, StoreNumber, StoreZip, OrderMonth, OrderYear, OrderTotal)

ORDER_ITEM (*OrderNumber*, *SKU*, Quantity, Price, ExtendedPrice)

SKU_DATA (SKU, SKU_Description, Department, Buyer)

2.7 Summarize the background of SQL.

SQL was developed by IBM in the late 1970s, and in 1992 it was endorsed as a national standard by the American National Standards Institute (ANSI). That version is called SQL-92. There is a later version called SQL3 that has some object-oriented concepts, but SQL3 has not received much commercial attention.

2.8 What is SQL-92? How does it relate to the SQL statements in this chapter?

SQL-92 is the version of SQL endorsed as a national standard by the American National Standards Institute (ANSI) in 1992. It is the version of SQL supported by most commonly used database management systems. The SQL statements in the chapter are based on SQL-92 and the SQL standards that followed and modified it.

2.9 What features have been added to SQL in versions subsequent to the SQL-92?

Versions of SQL subsequent to SQL-92 have extended features or added new features to SQL, the most important of which, for our purposes, is support for Extensible Markup Language (XML).

2.10 Why is SQL described as a data sublanguage?

A data sublanguage consists only of language statements for defining and processing a database. To obtain a full programming language, SQL statements must be embedded in scripting languages such as VBScript or in programming languages such as Java or C#.

2.11 What does DML stand for? What are DML statements?

DML stands for *data manipulation language*. DML statements are used for querying and modifying data.

2.12 What does DDL stand for? What are DDL statements?

DDL stands for *data definition language*. DDL statements are used for creating tables, relationships and other database querying and modifying data.

2.13 What is the SQL SELECT/FROM/WHERE framework?

The SQL SELECT/FROM/WHERE framework is the basis for queries in SQL. In this framework:

- The SQL SELECT clause specifies which columns are to be listed in the query results.
- The SQL FROM clause specifies which tables are to be used in the query.
- The SQL WHERE clause specifies which rows are to be listed in the query results.

2.14 Explain how Microsoft Access uses SQL.

Microsoft Access uses SQL, but generally hides the SQL from the user. For example, Microsoft Access automatically generates SQL and sends it to the Microsoft Access's internal Access Database Engine (ADE, which is a variant of the Microsoft Jet engine) every time you run a query, process a form or create a report. To go beyond elementary database processing, you need to know how to use SQL in Microsoft Access.

2.15 Explain how enterprise-class DBMS products use SQL.

Enterprise-class DBMS products, which include Microsoft SQL Server, Oracle Corporation's Oracle Database and MySQL, and IBM's DB2, require you to know and use SQL. All data manipulation is expressed in SQL in these products.

The Cape Codd Outdoor Sports sale extraction database has been modified to include two additional tables, the INVENTORY table and the WAREHOUSE table. The table schemas for these tables, together with the SKU table, are as follows:

RETAIL_ORDER (OrderNumber, StoreNumber, StoreZip, OrderMonth, OrderYear, OrderTotal)

ORDER_ITEM (*OrderNumber*, *SKU*, Quantity, Price, ExtendedPrice)

SKU_DATA (SKU, SKU_Description, Department, Buyer)

WAREHOUSE (WarehouseID, WarehouseCity, WarehouseState, Manager, Squarefeet)

INVENTORY (*WarehouseID*, *SKU*, SKU_Description, QuantityOnHand, QuantityOnOrder)

The five tables in the revised Cape Codd database schema are shown in Figure 2-24. The column characteristics for the WAREHOUSE table are shown in Figure 2-25, and the column characteristics for the INVENTORY table are shown in Figure 2-26. The data for the WAREHOUSE table are shown in Figure 2-27, and the data for the INVENTORY table are shown in Figure 2-28.

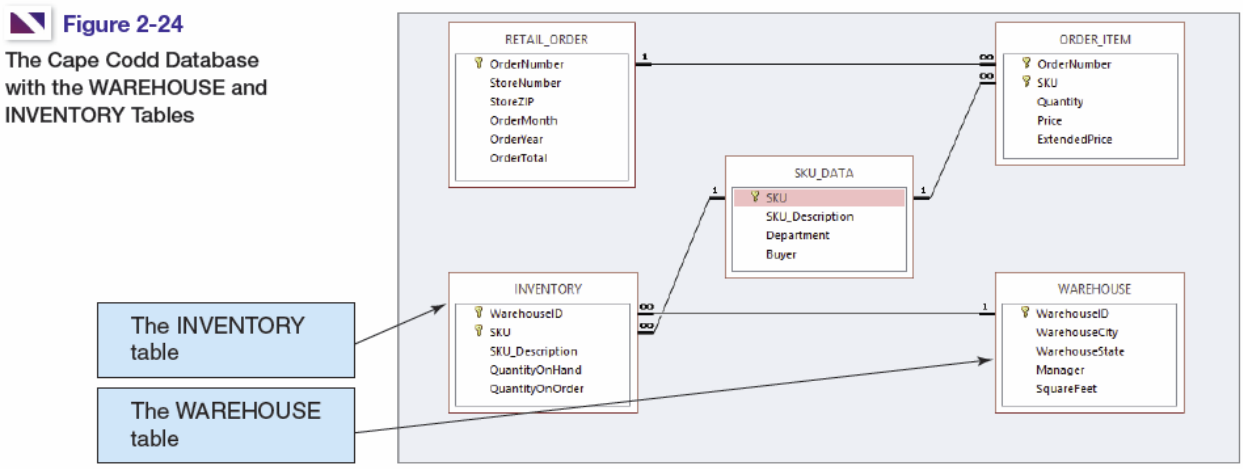


Figure 2-24 – The Cape Codd Database with the WAREHOUSE and INVENTORY tables

WAREHOUSE

Column Name	Type	Key	Required	Remarks
WarehouseID	Integer	Primary Key	Yes	Surrogate Key
WarehouseCity	Text (30)		Yes	
WarehouseState	Text (2)		Yes	
Manager	Text (35)	No	No	
SquareFeet	Integer	No	No	

Figure 2-25 - Column Characteristics for the WAREHOUSE Table

INVENTORY

Column Name	Type	Key	Required	Remarks
WarehouseID	Integer	Primary Key, Foreign Key	Yes	Surrogate Key
SKU	Integer	Primary Key, Foreign Key	Yes	Surrogate Key
SKU_Description	Text (35)	No	Yes	
QuantityOnHand	Integer	No	No	
QuantityOnOrder	Integer	No	No	

Figure 2-26 - Column Characteristics for the INVENTORY Table

WarehouseID	WarehouseCity	WarehouseState	Manager	SquareFeet
100	Atlanta	GA	Dave Jones	125,000
200	Chicago	IL	Lucille Smith	100,000
300	Bangor	MA	Bart Evans	150,000
400	Seattle	WA	Dale Rogers	130,000

Figure 2-27 - Cape Codd Outdoor Sports WAREHOUSE Data

WarehouseID	SKU	SKU_Description	QuantityOnHand	QuantityOnOrder
100	100100	Std. Scuba Tank, Yellow	250	0
200	100100	Std. Scuba Tank, Yellow	100	50
300	100100	Std. Scuba Tank, Yellow	100	0
400	100100	Std. Scuba Tank, Yellow	200	0
100	100200	Std. Scuba Tank, Magenta	200	30
200	100200	Std. Scuba Tank, Magenta	75	75
300	100200	Std. Scuba Tank, Magenta	100	100
400	100200	Std. Scuba Tank, Magenta	250	0
100	101100	Dive Mask, Small Clear	0	500
200	101100	Dive Mask, Small Clear	0	500
300	101100	Dive Mask, Small Clear	300	200
400	101100	Dive Mask, Small Clear	450	0
100	101200	Dive Mask, Med Clear	100	500
200	101200	Dive Mask, Med Clear	50	500
300	101200	Dive Mask, Med Clear	475	0
400	101200	Dive Mask, Med Clear	250	250
100	201000	Half-Dome Tent	2	100
200	201000	Half-Dome Tent	10	250
300	201000	Half-Dome Tent	250	0
400	201000	Half-Dome Tent	0	250
100	202000	Half-Dome Tent Vestibule	10	250
200	202000	Half-Dome Tent Vestibule	1	250
300	202000	Half-Dome Tent Vestibule	100	0
400	202000	Half-Dome Tent Vestibule	0	200
100	301000	Light Fly Climbing Harness	300	250
200	301000	Light Fly Climbing Harness	250	250
300	301000	Light Fly Climbing Harness	0	250
400	301000	Light Fly Climbing Harness	0	250
100	302000	Locking Carabiner, Oval	1000	0
200	302000	Locking Carabiner, Oval	1250	0
300	302000	Locking Carabiner, Oval	500	500
400	302000	Locking Carabiner, Oval	0	1000

Figure 2-28 - Cape Codd Outdoor Sports INVENTORY Data

If at all possible, you should run your SQL solutions to the following questions against an actual database. A Microsoft Access database named Cape-Codd.accdb is available on our Web site (www.pearsonhighered.com/kroenke) that contains all the tables and data for the Cape Codd Outdoor Sports sales data extract database. Also available on our Web site are SQL scripts for creating and populating the tables for the Cape Codd database in Microsoft SQL Server, Oracle Database, and MySQL.

NOTE: All answers below show the correct SQL statement, as well as SQL statements modified for Microsoft Access 2013 when needed. Whenever possible, all results were obtained by running the SQL statements in Microsoft Access 2013, and the corresponding screen shots of the results are shown below. As explained in the text, some queries cannot be run in Microsoft Access 2013, and for those queries the correct result was obtained using Microsoft SQL Server 2012. The SQL statements shown should run with little, if any, modification needed for Oracle Database 11*g* Release 2 and MySQL 5.6.

Solutions to Project Questions 2.17 – 2.55 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

If your students are using a DBMS other than Microsoft Access, the SQL code to create and populate the Cape Codd database is available in the *.sql script files for SQL Server 2012, Oracle Database 11*g*, and MySQL 5.5 in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

2.16 *There is an intentional flaw in the design of the INVENTORY table used in these exercises. This flaw was purposely included in the INVENTORY tables so that you can answer some of the following questions using only that table. Compare the SKU and INVENTORY tables, and determine what design flaw is included in INVENTORY. Specifically, why did we include it?*

The flaw is the inclusion of the SKU_Description attribute in the INVENTORY table. This attribute duplicates the SKU_Description attribute and data in the SKU_DATA table, where the attribute rightfully belongs. By duplicating SKU_Description in the INVENTORY table, we can ask you to list the SKU and its associated description in a single table query against the INVENTORY table. Otherwise, a two table query would be required. If these tables were in a production database, we would eliminate the INVENTORY.SKU_Description column.

Use only the INVENTORY table to answer Review Questions 2.17 through 2.39:

2.17 Write an SQL statement to display SKU and SKU_Description.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description
FROM      INVENTORY;
```

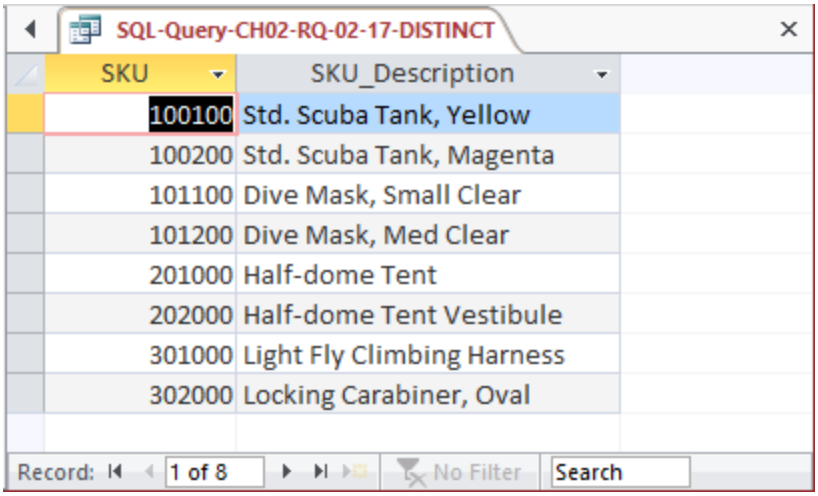
SQL-Query-CH02-RQ-02-17

SKU	SKU_Description
100100	Std. Scuba Tank, Yellow
100200	Std. Scuba Tank, Magenta
101100	Dive Mask, Small Clear
101200	Dive Mask, Med Clear
201000	Half-dome Tent
202000	Half-dome Tent Vestibule
301000	Light Fly Climbing Harness
302000	Locking Carabiner, Oval
100100	Std. Scuba Tank, Yellow
100200	Std. Scuba Tank, Magenta
101100	Dive Mask, Small Clear
101200	Dive Mask, Med Clear
201000	Half-dome Tent
202000	Half-dome Tent Vestibule
301000	Light Fly Climbing Harness
302000	Locking Carabiner, Oval
100100	Std. Scuba Tank, Yellow
100200	Std. Scuba Tank, Magenta
101100	Dive Mask, Small Clear
101200	Dive Mask, Med Clear
201000	Half-dome Tent
202000	Half-dome Tent Vestibule
301000	Light Fly Climbing Harness
302000	Locking Carabiner, Oval
100100	Std. Scuba Tank, Yellow
100200	Std. Scuba Tank, Magenta
101100	Dive Mask, Small Clear
101200	Dive Mask, Med Clear
201000	Half-dome Tent
202000	Half-dome Tent Vestibule
301000	Light Fly Climbing Harness
302000	Locking Carabiner, Oval

Record: 1 of 32 No Filter Search

The question does not ask for unique SKU and SKU_Description data, but could be obtained by using:

```
SELECT    UNIQUE SKU, SKU_Description
FROM      INVENTORY;
```



2.18 Write an SQL statement to display SKU_Description and SKU.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT   SKU_Description, SKU
FROM     INVENTORY;
```

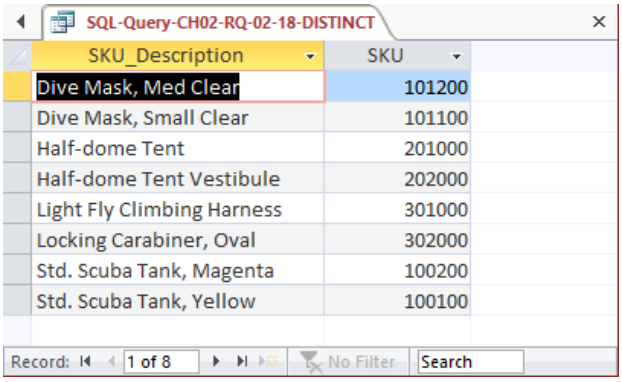
SQL-Query-CH02-RQ-02-18

SKU_Description	SKU
Std. Scuba Tank, Yellow	100100
Std. Scuba Tank, Magenta	100200
Dive Mask, Small Clear	101100
Dive Mask, Med Clear	101200
Half-dome Tent	201000
Half-dome Tent Vestibule	202000
Light Fly Climbing Harness	301000
Locking Carabiner, Oval	302000
Std. Scuba Tank, Yellow	100100
Std. Scuba Tank, Magenta	100200
Dive Mask, Small Clear	101100
Dive Mask, Med Clear	101200
Half-dome Tent	201000
Half-dome Tent Vestibule	202000
Light Fly Climbing Harness	301000
Locking Carabiner, Oval	302000
Std. Scuba Tank, Yellow	100100
Std. Scuba Tank, Magenta	100200
Dive Mask, Small Clear	101100
Dive Mask, Med Clear	101200
Half-dome Tent	201000
Half-dome Tent Vestibule	202000
Light Fly Climbing Harness	301000
Locking Carabiner, Oval	302000
Std. Scuba Tank, Yellow	100100
Std. Scuba Tank, Magenta	100200
Dive Mask, Small Clear	101100
Dive Mask, Med Clear	101200
Half-dome Tent	201000
Half-dome Tent Vestibule	202000
Light Fly Climbing Harness	301000
Locking Carabiner, Oval	302000

Record: 1 of 32 No Filter Search

The question does not ask for unique SKU and SKU_Description data, but could be obtained by using:

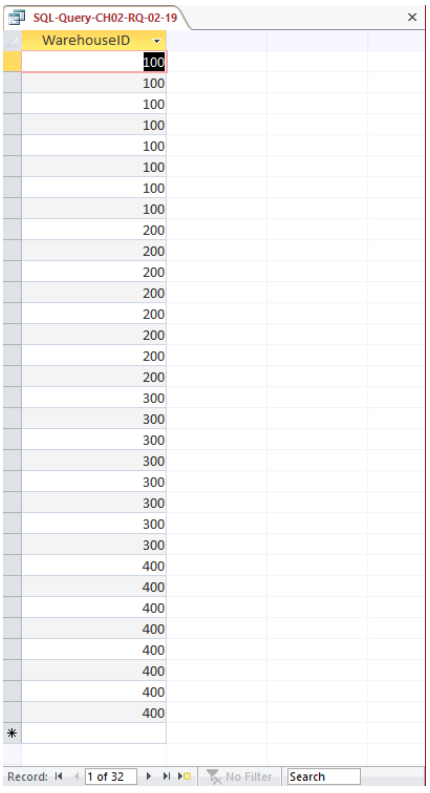
```
SELECT   UNIQUE SKU_Description, SKU
FROM     INVENTORY;
```



2.19 Write an SQL statement to display WarehouseID.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

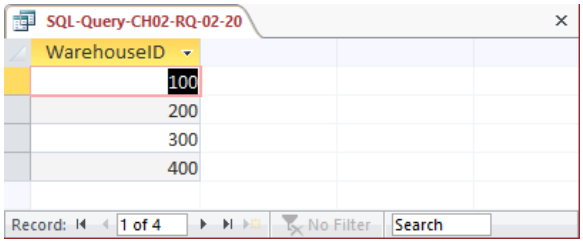
```
SELECT   WarehouseID
FROM     INVENTORY;
```



2.20 Write an SQL statement to display unique WarehouseIDs.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    DISTINCT WarehouseID
FROM      INVENTORY;
```



2.21 Write an SQL statement to display all of the columns without using the SQL asterisk (*) wildcard character.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    WarehouseID, SKU, SKU_Description,
          QuantityOnHand, QuantityOnOrder
FROM      INVENTORY;
```

SQL-Query-CH02-RQ-02-21

WarehouseID	SKU	SKU_Description	QuantityOnHand	QuantityOnOrder
100	100100	Std. Scuba Tank, Yellow	250	0
100	100200	Std. Scuba Tank, Magenta	200	30
100	101100	Dive Mask, Small Clear	0	500
100	101200	Dive Mask, Med Clear	100	500
100	201000	Half-dome Tent	2	100
100	202000	Half-dome Tent Vestibule	10	250
100	301000	Light Fly Climbing Harness	300	250
100	302000	Locking Carabiner, Oval	1000	0
200	100100	Std. Scuba Tank, Yellow	100	50
200	100200	Std. Scuba Tank, Magenta	75	75
200	101100	Dive Mask, Small Clear	0	500
200	101200	Dive Mask, Med Clear	50	500
200	201000	Half-dome Tent	10	250
200	202000	Half-dome Tent Vestibule	1	250
200	301000	Light Fly Climbing Harness	250	250
200	302000	Locking Carabiner, Oval	1250	0
300	100100	Std. Scuba Tank, Yellow	100	0
300	100200	Std. Scuba Tank, Magenta	100	100
300	101100	Dive Mask, Small Clear	300	200
300	101200	Dive Mask, Med Clear	475	0
300	201000	Half-dome Tent	250	0
300	202000	Half-dome Tent Vestibule	100	0
300	301000	Light Fly Climbing Harness	0	250
300	302000	Locking Carabiner, Oval	500	500
400	100100	Std. Scuba Tank, Yellow	200	0
400	100200	Std. Scuba Tank, Magenta	250	0
400	101100	Dive Mask, Small Clear	450	0
400	101200	Dive Mask, Med Clear	250	250
400	201000	Half-dome Tent	0	250
400	202000	Half-dome Tent Vestibule	0	200
400	301000	Light Fly Climbing Harness	0	250
400	302000	Locking Carabiner, Oval	0	1000

Record: 1 of 32 No Filter Search

2.22 Write an SQL statement to display all of the columns using the SQL asterisk (*) wildcard character.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    *
FROM      INVENTORY;
```

SQL-Query-CH02-RQ-02-22

WarehouseID	SKU	SKU_Description	QuantityOnHand	QuantityOnOrder
100	100100	Std. Scuba Tank, Yellow	250	0
100	100200	Std. Scuba Tank, Magenta	200	30
100	101100	Dive Mask, Small Clear	0	500
100	101200	Dive Mask, Med Clear	100	500
100	201000	Half-dome Tent	2	100
100	202000	Half-dome Tent Vestibule	10	250
100	301000	Light Fly Climbing Harness	300	250
100	302000	Locking Carabiner, Oval	1000	0
200	100100	Std. Scuba Tank, Yellow	100	50
200	100200	Std. Scuba Tank, Magenta	75	75
200	101100	Dive Mask, Small Clear	0	500
200	101200	Dive Mask, Med Clear	50	500
200	201000	Half-dome Tent	10	250
200	202000	Half-dome Tent Vestibule	1	250
200	301000	Light Fly Climbing Harness	250	250
200	302000	Locking Carabiner, Oval	1250	0
300	100100	Std. Scuba Tank, Yellow	100	0
300	100200	Std. Scuba Tank, Magenta	100	100
300	101100	Dive Mask, Small Clear	300	200
300	101200	Dive Mask, Med Clear	475	0
300	201000	Half-dome Tent	250	0
300	202000	Half-dome Tent Vestibule	100	0
300	301000	Light Fly Climbing Harness	0	250
300	302000	Locking Carabiner, Oval	500	500
400	100100	Std. Scuba Tank, Yellow	200	0
400	100200	Std. Scuba Tank, Magenta	250	0
400	101100	Dive Mask, Small Clear	450	0
400	101200	Dive Mask, Med Clear	250	250
400	201000	Half-dome Tent	0	250
400	202000	Half-dome Tent Vestibule	0	200
400	301000	Light Fly Climbing Harness	0	250
400	302000	Locking Carabiner, Oval	0	1000

Record: 1 of 32 No Filter Search

2.23 Write an SQL statement to display all data on products having a QuantityOnHand greater than 0.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    *
FROM      INVENTORY
WHERE     QuantityOnHand >0;
```

SQL-Query-CH02-RQ-02-23

WarehouseID	SKU	SKU_Description	QuantityOnHand	QuantityOnOrder
100	100100	Std. Scuba Tank, Yellow	250	0
200	100100	Std. Scuba Tank, Yellow	100	50
300	100100	Std. Scuba Tank, Yellow	100	0
400	100100	Std. Scuba Tank, Yellow	200	0
100	100200	Std. Scuba Tank, Magenta	200	30
200	100200	Std. Scuba Tank, Magenta	75	75
300	100200	Std. Scuba Tank, Magenta	100	100
400	100200	Std. Scuba Tank, Magenta	250	0
300	101100	Dive Mask, Small Clear	300	200
400	101100	Dive Mask, Small Clear	450	0
100	101200	Dive Mask, Med Clear	100	500
200	101200	Dive Mask, Med Clear	50	500
300	101200	Dive Mask, Med Clear	475	0
400	101200	Dive Mask, Med Clear	250	250
100	201000	Half-dome Tent	2	100
200	201000	Half-dome Tent	10	250
300	201000	Half-dome Tent	250	0
100	202000	Half-dome Tent Vestibule	10	250
200	202000	Half-dome Tent Vestibule	1	250
300	202000	Half-dome Tent Vestibule	100	0
100	301000	Light Fly Climbing Harness	300	250
200	301000	Light Fly Climbing Harness	250	250
100	302000	Locking Carabiner, Oval	1000	0
200	302000	Locking Carabiner, Oval	1250	0
300	302000	Locking Carabiner, Oval	500	500

Record: 1 of 25 No Filter Search

2.24 Write an SQL statement to display the SKU and SKU_Description for products having QuantityOnHand equal to 0.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description
FROM      INVENTORY
WHERE     QuantityOnHand =0;
```

SQL-Query-CH02-RQ-02-24

SKU	SKU_Description
101100	Dive Mask, Small Clear
101100	Dive Mask, Small Clear
201000	Half-dome Tent
202000	Half-dome Tent Vestibule
301000	Light Fly Climbing Harness
301000	Light Fly Climbing Harness
302000	Locking Carabiner, Oval

Record: 1 of 7 No Filter Search

2.25 Write an SQL statement to display the SKU, SKU_Description, and Warehouse for products having QuantityOnHand equal to 0. Sort the results in ascending order by WarehouseID.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description, WarehouseID
FROM      INVENTORY
WHERE     QuantityOnHand =0
ORDER BY WarehouseID;
```

SQL-Query-CH02-RQ-02-25

SKU	SKU_Description	WarehouseID
101100	Dive Mask, Small Clear	100
101100	Dive Mask, Small Clear	200
301000	Light Fly Climbing Harness	300
302000	Locking Carabiner, Oval	400
301000	Light Fly Climbing Harness	400
202000	Half-dome Tent Vestibule	400
201000	Half-dome Tent	400

Record: 1 of 7 No Filter Search

2.26 Write an SQL statement to display the SKU, SKU_Description, and WarehouseID for products having QuantityOnHand greater than 0. Sort the results in descending order by WarehouseID and ascending order by SKU.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      SKU, SKU_Description, WarehouseID
FROM        INVENTORY
WHERE       QuantityOnHand > 0
ORDER BY    WarehouseID DESC, SKU;
```

SQL-Query-CH02-RQ-02-26

SKU	SKU_Description	WarehouseID
100100	Std. Scuba Tank, Yellow	400
100200	Std. Scuba Tank, Magenta	400
101100	Dive Mask, Small Clear	400
101200	Dive Mask, Med Clear	400
100100	Std. Scuba Tank, Yellow	300
100200	Std. Scuba Tank, Magenta	300
101100	Dive Mask, Small Clear	300
101200	Dive Mask, Med Clear	300
201000	Half-dome Tent	300
202000	Half-dome Tent Vestibule	300
302000	Locking Carabiner, Oval	300
100100	Std. Scuba Tank, Yellow	200
100200	Std. Scuba Tank, Magenta	200
101200	Dive Mask, Med Clear	200
201000	Half-dome Tent	200
202000	Half-dome Tent Vestibule	200
301000	Light Fly Climbing Harness	200
302000	Locking Carabiner, Oval	200
100100	Std. Scuba Tank, Yellow	100
100200	Std. Scuba Tank, Magenta	100
101200	Dive Mask, Med Clear	100
201000	Half-dome Tent	100
202000	Half-dome Tent Vestibule	100
301000	Light Fly Climbing Harness	100
302000	Locking Carabiner, Oval	100

Record: 1 of 25 No Filter Search

2.27 Write an SQL statement to display SKU, SKU_Description, and WarehouseID for all products that have a QuantityOnHand equal to 0 and a QuantityOnOrder greater than 0. Sort the results in descending order by WarehouseID and in ascending order by SKU.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      SKU, SKU_Description, WarehouseID
FROM        INVENTORY
WHERE       QuantityOnHand = 0
   AND      QuantityOnOrder > 0
ORDER BY    WarehouseID DESC, SKU;
```

SQL-Query-CH02-RQ-02-27

SKU	SKU_Description	WarehouseID
201000	Half-dome Tent	400
202000	Half-dome Tent Vestibule	400
301000	Light Fly Climbing Harness	400
302000	Locking Carabiner, Oval	400
301000	Light Fly Climbing Harness	300
101100	Dive Mask, Small Clear	200
101100	Dive Mask, Small Clear	100

Record: 1 of 7 No Filter Search

2.28 Write an SQL statement to display SKU, SKU_Description, and WarehouseID for all products that have a QuantityOnHand equal to 0 or a QuantityOnOrder equal to 0. Sort the results in descending order by WarehouseID and in ascending order by SKU.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      SKU, SKU_Description, WarehouseID
FROM        INVENTORY
WHERE       QuantityOnHand = 0
   OR       QuantityOnOrder = 0
ORDER BY    WarehouseID DESC, SKU;
```

SQL-Query-CH02-RQ-02-28

SKU	SKU_Description	WarehouseID
100100	Std. Scuba Tank, Yellow	400
100200	Std. Scuba Tank, Magenta	400
101100	Dive Mask, Small Clear	400
201000	Half-dome Tent	400
202000	Half-dome Tent Vestibule	400
301000	Light Fly Climbing Harness	400
302000	Locking Carabiner, Oval	400
100100	Std. Scuba Tank, Yellow	300
101200	Dive Mask, Med Clear	300
201000	Half-dome Tent	300
202000	Half-dome Tent Vestibule	300
301000	Light Fly Climbing Harness	300
101100	Dive Mask, Small Clear	200
302000	Locking Carabiner, Oval	200
100100	Std. Scuba Tank, Yellow	100
101100	Dive Mask, Small Clear	100
302000	Locking Carabiner, Oval	100

Record: 1 of 17 No Filter Search

2.29 Write an SQL statement to display the SKU, SKU_Description, WarehouseID, and QuantityOnHand for all products having a QuantityOnHand greater than 1 and less than 10. Do not use the BETWEEN keyword.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      SKU, SKU_Description, WarehouseID, QuantityOnHand
FROM        INVENTORY
WHERE       QuantityOnHand > 1
   AND      QuantityOnhand < 10;
```

SQL-Query-CH02-RQ-02-29

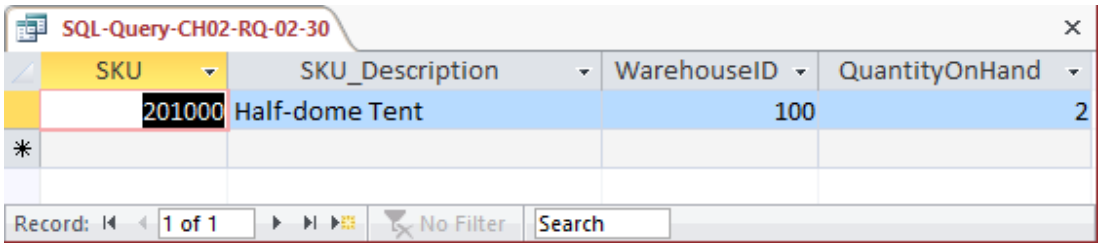
SKU	SKU_Description	WarehouseID	QuantityOnHand
201000	Half-dome Tent	100	2

Record: 1 of 1 No Filter Search

2.30 Write an SQL statement to display the SKU, SKU_Description, WarehouseID, and QuantityOnHand for all products having a QuantityOnHand greater than 1 and less than 10. Use the BETWEEN keyword.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      SKU, SKU_Description, WarehouseID, QuantityOnHand
FROM        INVENTORY
WHERE       QuantityOnHand BETWEEN 2 AND 9;
```



2.31 Write an SQL statement to show a unique SKU and SKU_Description for all products having an SKU description starting with 'Half-dome'.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

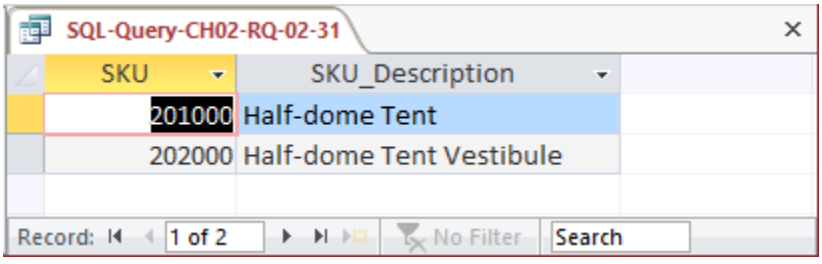
Note that, as discussed in Chapter 2, Microsoft Access 2013 uses wildcard characters that differ from the SQL standard.

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE 'Half-dome%';
```

For Microsoft Access:

```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE 'Half-dome*';
```



2.32 Write an SQL statement to show a unique SKU and SKU_Description for all products having a description that includes the word 'Climb'.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

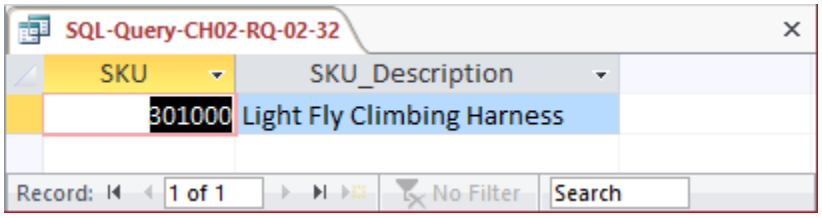
Note that, as discussed in Chapter 2, Microsoft Access 2013 uses wildcard characters that differ from the SQL standard.

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE '%Climb%';
```

For Microsoft Access:

```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE '*Climb*';
```



2.33 Write an SQL statement to show a unique SKU and SKU_Description for all products having a 'd' in the third position from the left in SKU_Description.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

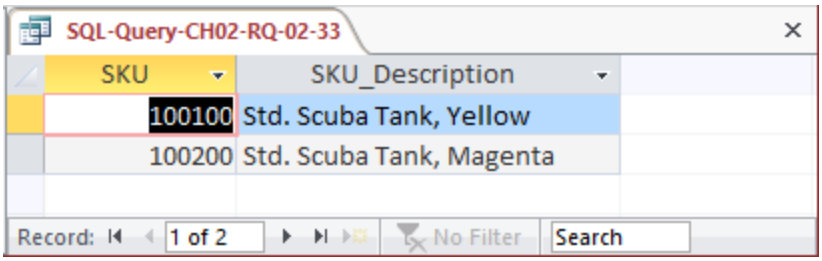
Note that, as discussed in Chapter 2, Microsoft Access 2013 uses wildcard characters that differ from the SQL standard.

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE '__d%';
```

For Microsoft Access:

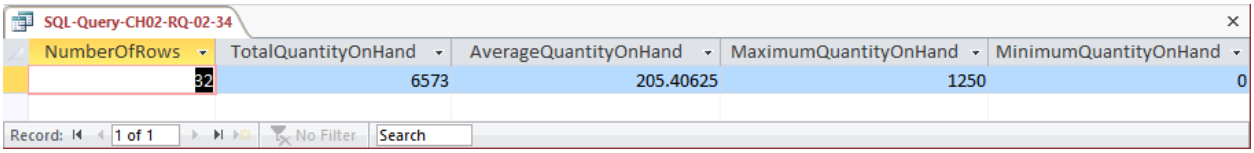
```
SELECT    DISTINCT SKU, SKU_Description
FROM      INVENTORY
WHERE     SKU_Description LIKE '??d*';
```



2.34 Write an SQL statement that uses all of the SQL built-in functions on the QuantityOnHand column. Include meaningful column names in the result.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    COUNT(QuantityOnHand) AS NumberOfRows,
          SUM(QuantityOnHand) AS TotalQuantityOnHand,
          AVG(QuantityOnHand) AS AverageQuantityOnHand,
          MAX(QuantityOnHand) AS MaximumQuantityOnHand,
          MIN(QuantityOnHand) AS MinimumQuantityOnHand
FROM      INVENTORY;
```



2.35 Explain the difference between the SQL built-in functions COUNT and SUM.

COUNT counts the number of rows or records in a table, while SUM adds up the data values in the specified column.

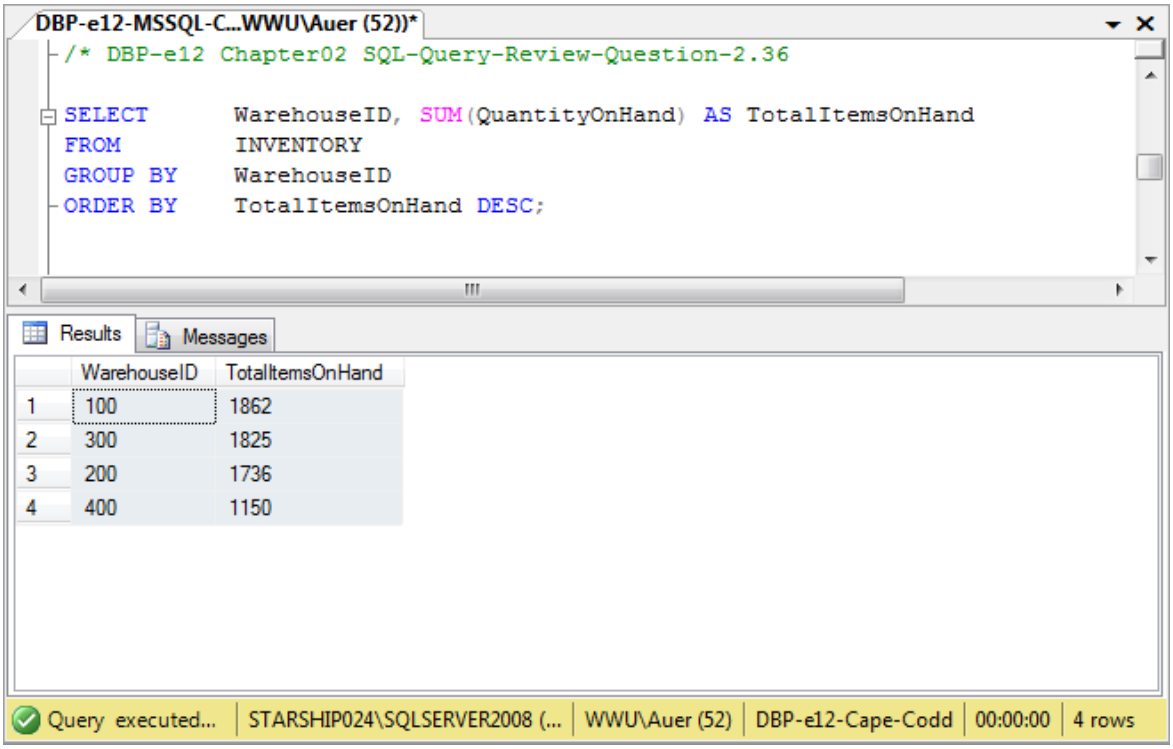
2.36 Write an SQL statement to display the WarehouseID and the sum of QuantityOnHand, grouped by WarehouseID. Name the sum TotalItemsOnHand and display the results in descending order of TotalItemsOnHand.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHand
FROM        INVENTORY
GROUP BY    WarehouseID
ORDER BY    TotalItemsOnHand DESC;
```

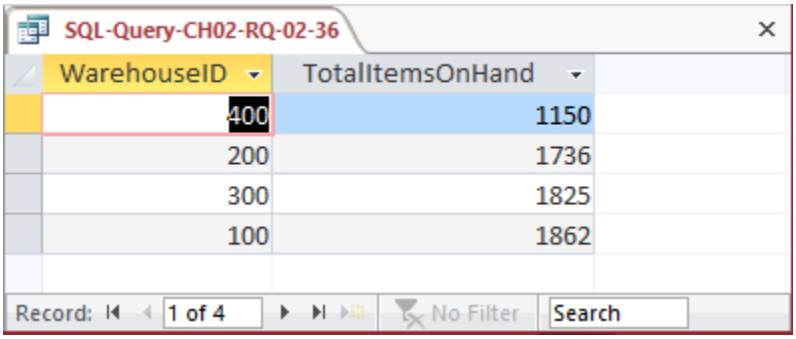
The correct results, obtained from SQL Server 2008 R2 / 2012, are:



For Microsoft Access:

Unfortunately, Microsoft Access cannot process the ORDER BY clause because it contains an aliased computed result. To correct this, we use an SQL statement with the un-aliased computation:

```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHand
FROM        INVENTORY
GROUP BY    WarehouseID
ORDER BY    SUM(QuantityOnHand) DESC;
```

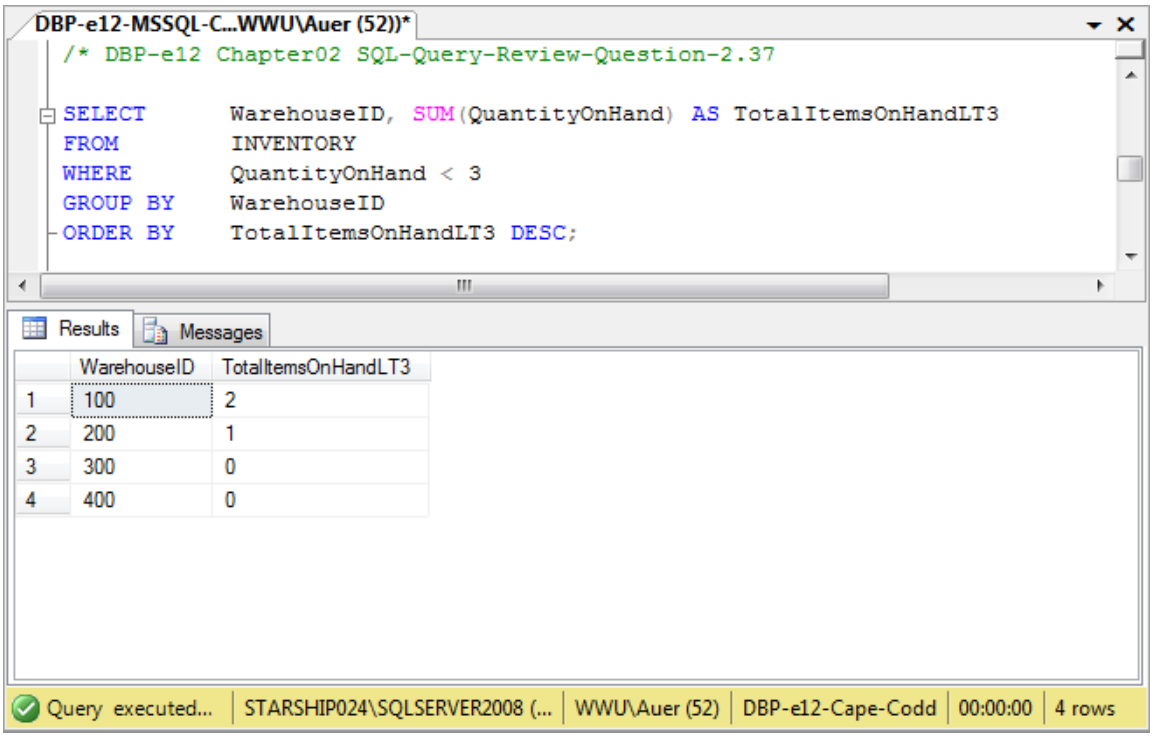


2.37 Write an SQL statement to display the WarehouseID and the sum of QuantityOnHand, grouped by WarehouseID. Omit all SKU items that have 3 or more items on hand from the sum, and name the sum TotalItemsOnHandLT3. Display the results in descending order of TotalItemsOnHandLT3.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

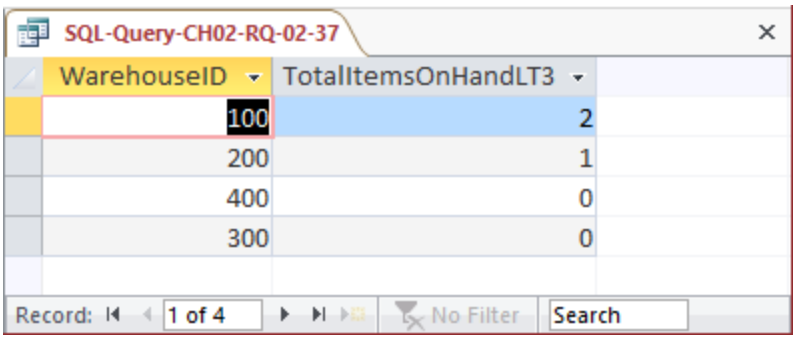
```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHandLT3
FROM        INVENTORY
WHERE       QuantityOnHand < 3
GROUP BY    WarehouseID
ORDER BY    TotalItemsOnHandLT3 DESC;
```



For Microsoft Access:

Unfortunately, Microsoft Access cannot process the ORDER BY clause because it contains an aliased computed result. To correct this, we use an SQL statement with the un-aliased computation:

```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHandLT3
FROM        INVENTORY
WHERE       QuantityOnHand < 3
GROUP BY    WarehouseID
ORDER BY    SUM(QuantityOnHand) DESC;
```

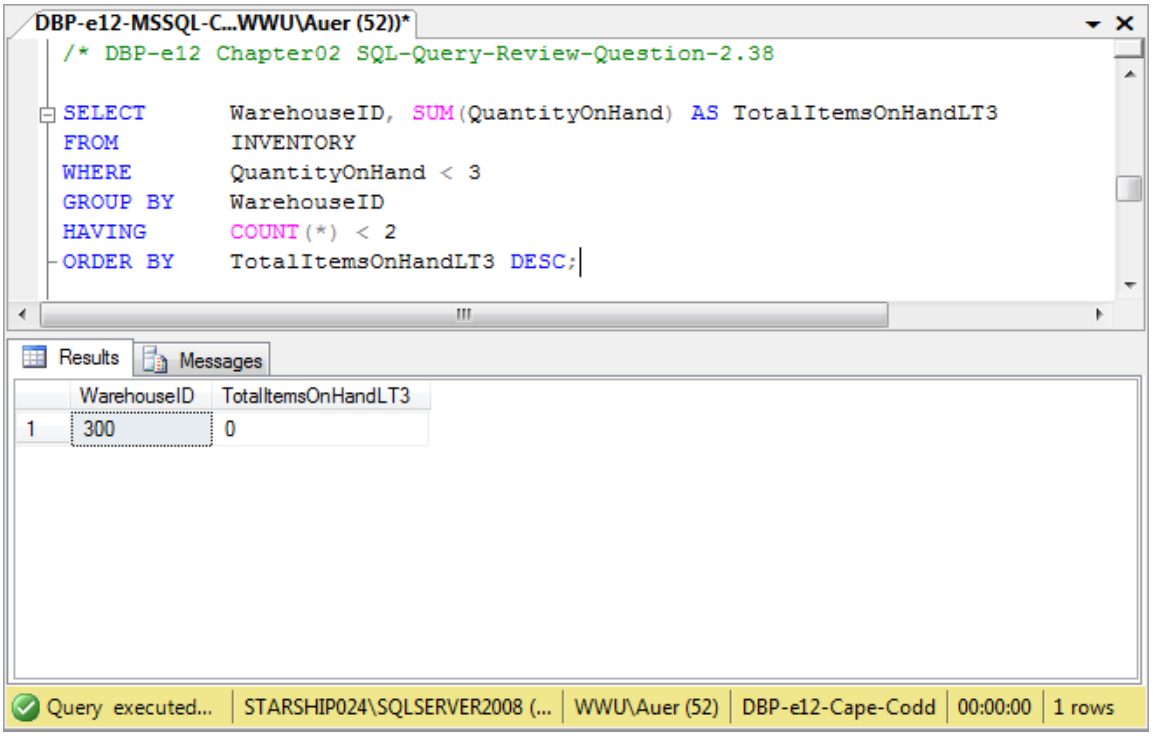


2.38 *Write an SQL statement to display the WarehouseID and the sum of QuantityOnHand grouped by WarehouseID. Omit all SKU items that have 3 or more items on hand from the sum, and name the sum TotalItemsOnHandLT3. Show Warehouse ID only for warehouses having fewer than 2 SKUs in their TotalItemsOnHandLT3. Display the results in descending order of TotalItemsOnHandLT3.*

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

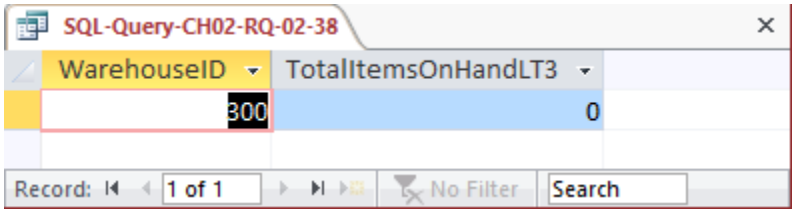
```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHandLT3
FROM        INVENTORY
WHERE       QuantityOnHand < 3
GROUP BY    WarehouseID
HAVING      COUNT(*) < 2
ORDER BY    TotalItemsOnHandLT3 DESC;
```



For Microsoft Access:

Unfortunately, Microsoft Access cannot process the ORDER BY clause because it contains an aliased computed result. To correct this, we use an SQL statement with the un-aliased computation:

```
SELECT      WarehouseID, SUM(QuantityOnHand) AS TotalItemsOnHandLT3
FROM        INVENTORY
WHERE       QuantityOnHand < 3
GROUP BY    WarehouseID
HAVING      COUNT(*) < 2
ORDER BY    SUM(QuantityOnHand) DESC;
```



2.39 In your answer to Review Question 2.38, was the WHERE or HAVING applied first? Why?

The WHERE clause is always applied before the HAVING clause. Otherwise there would be ambiguity in the SQL statement and the results would differ according to which clause was applied first.

Use both the INVENTORY and WAREHOUSE tables to answer Review Questions 2.40 through 2.55:

2.40 Write an SQL statement to display the SKU, SKU_Description, and WarehouseID, WarehouseCity, and WarehouseState for all items stored in the Atlanta, Bangor, or Chicago warehouse. Do not use the IN keyword.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description,
          WAREHOUSE.WarehouseID, WarehouseCity, WarehouseState
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
   AND    (WarehouseCity = 'Atlanta'
           OR   WarehouseCity = 'Bangor'
           OR   WarehouseCity = 'Chicago')
```

SQL-Query-CH02-RQ-02-40

SKU	SKU_Description	WarehouseID	WarehouseCity	WarehouseState
100100	Std. Scuba Tank, Yellow	100	Atlanta	GA
100200	Std. Scuba Tank, Magenta	100	Atlanta	GA
101100	Dive Mask, Small Clear	100	Atlanta	GA
101200	Dive Mask, Med Clear	100	Atlanta	GA
201000	Half-dome Tent	100	Atlanta	GA
202000	Half-dome Tent Vestibule	100	Atlanta	GA
301000	Light Fly Climbing Harness	100	Atlanta	GA
302000	Locking Carabiner, Oval	100	Atlanta	GA
100100	Std. Scuba Tank, Yellow	200	Chicago	IL
100200	Std. Scuba Tank, Magenta	200	Chicago	IL
101100	Dive Mask, Small Clear	200	Chicago	IL
101200	Dive Mask, Med Clear	200	Chicago	IL
201000	Half-dome Tent	200	Chicago	IL
202000	Half-dome Tent Vestibule	200	Chicago	IL
301000	Light Fly Climbing Harness	200	Chicago	IL
302000	Locking Carabiner, Oval	200	Chicago	IL
100100	Std. Scuba Tank, Yellow	300	Bangor	ME
100200	Std. Scuba Tank, Magenta	300	Bangor	ME
101100	Dive Mask, Small Clear	300	Bangor	ME
101200	Dive Mask, Med Clear	300	Bangor	ME
201000	Half-dome Tent	300	Bangor	ME
202000	Half-dome Tent Vestibule	300	Bangor	ME
301000	Light Fly Climbing Harness	300	Bangor	ME
302000	Locking Carabiner, Oval	300	Bangor	ME

Record: 1 of 24 No Filter Search

2.41 Write an SQL statement to display the SKU, SKU_Description, and WarehouseID, WarehouseCity, and WarehouseState for all items stored in the Atlanta, Bangor, or Chicago warehouse. Use the IN keyword.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description,
          WAREHOUSE.WarehouseID, WarehouseCity, WarehouseState
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
   AND    WarehouseCity IN ('Atlanta', 'Bangor' ,'Chicago');
```

SQL-Query-CH02-RQ-02-41

SKU	SKU_Description	WarehouseID	WarehouseCity	WarehouseState
100100	Std. Scuba Tank, Yellow	100	Atlanta	GA
100200	Std. Scuba Tank, Magenta	100	Atlanta	GA
101100	Dive Mask, Small Clear	100	Atlanta	GA
101200	Dive Mask, Med Clear	100	Atlanta	GA
201000	Half-dome Tent	100	Atlanta	GA
202000	Half-dome Tent Vestibule	100	Atlanta	GA
301000	Light Fly Climbing Harness	100	Atlanta	GA
302000	Locking Carabiner, Oval	100	Atlanta	GA
100100	Std. Scuba Tank, Yellow	200	Chicago	IL
100200	Std. Scuba Tank, Magenta	200	Chicago	IL
101100	Dive Mask, Small Clear	200	Chicago	IL
101200	Dive Mask, Med Clear	200	Chicago	IL
201000	Half-dome Tent	200	Chicago	IL
202000	Half-dome Tent Vestibule	200	Chicago	IL
301000	Light Fly Climbing Harness	200	Chicago	IL
302000	Locking Carabiner, Oval	200	Chicago	IL
100100	Std. Scuba Tank, Yellow	300	Bangor	ME
100200	Std. Scuba Tank, Magenta	300	Bangor	ME
101100	Dive Mask, Small Clear	300	Bangor	ME
101200	Dive Mask, Med Clear	300	Bangor	ME
201000	Half-dome Tent	300	Bangor	ME
202000	Half-dome Tent Vestibule	300	Bangor	ME
301000	Light Fly Climbing Harness	300	Bangor	ME
302000	Locking Carabiner, Oval	300	Bangor	ME

Record: 1 of 24 No Filter Search

2.42 Write an SQL statement to display the SKU, SKU_Description, WarehouseID, WarehouseCity, and WarehouseState of all items not stored in the Atlanta, Bangor, or Chicago warehouse. Do not use the NOT IN keyword.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

NOTE: The symbol for "not equal to" is < >. Since we want the query output for warehouses that are not Atlanta or Bangor or Chicago as a set, we must ask for warehouses that are not in the group (Atlanta **and** Bangor **and** Chicago). This means we use AND in the WHERE clause – if we used OR in the WHERE clause, we would end up with ALL warehouses being in the query output. This happens because each OR eliminates only one warehouse, but that warehouse still qualifies for inclusion in the other OR statements. To demonstrate this, substitute OR for each AND in the SQL statement below.

```
SELECT    SKU, SKU_Description,
          WAREHOUSE.WarehouseID, WarehouseCity, WarehouseState
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
   AND    WarehouseCity <> 'Atlanta'
   AND    WarehouseCity <> 'Bangor'
   AND    WarehouseCity <> 'Chicago';
```

SQL-Query-CH02-RQ-02-42

SKU	SKU_Description	WarehouseID	WarehouseCity	WarehouseState
100100	Std. Scuba Tank, Yellow	400	Seattle	WA
100200	Std. Scuba Tank, Magenta	400	Seattle	WA
101100	Dive Mask, Small Clear	400	Seattle	WA
101200	Dive Mask, Med Clear	400	Seattle	WA
201000	Half-dome Tent	400	Seattle	WA
202000	Half-dome Tent Vestibule	400	Seattle	WA
301000	Light Fly Climbing Harness	400	Seattle	WA
302000	Locking Carabiner, Oval	400	Seattle	WA

Record: 1 of 8 No Filter Search

2.43 *Write an SQL statement to display the SKU, SKU_Description, WarehouseID, WarehouseCity, and WarehouseState of all items not stored in the Atlanta, Bangor, or Chicago warehouse. Use the NOT IN keyword.*

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description,
          WAREHOUSE.WarehouseID, WarehouseCity, WarehouseState
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
   AND    WarehouseCity NOT IN ('Atlanta', 'Bangor' ,'Chicago');
```

SQL-Query-CH02-RQ-02-43

SKU	SKU_Description	WarehouseID	WarehouseCity	WarehouseState
100100	Std. Scuba Tank, Yellow	400	Seattle	WA
100200	Std. Scuba Tank, Magenta	400	Seattle	WA
101100	Dive Mask, Small Clear	400	Seattle	WA
101200	Dive Mask, Med Clear	400	Seattle	WA
201000	Half-dome Tent	400	Seattle	WA
202000	Half-dome Tent Vestibule	400	Seattle	WA
301000	Light Fly Climbing Harness	400	Seattle	WA
302000	Locking Carabiner, Oval	400	Seattle	WA

Record: 1 of 8 No Filter Search

2.44 *Write an SQL statement to produce a single column called ItemLocation that combines the SKU_Description, the phrase "is in a warehouse in", and WarehouseCity. Do not be concerned with removing leading or trailing blanks.*

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

Note that the SQL syntax will vary depending upon the DBMS—see the discussion in Chapter 2.

```
SELECT    SKU_Description+' is in a warehouse in '
          +WarehouseCity AS ITEM_Location
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID;
```

SQL-Query-CH02-RQ-02-44

ITEM_Location
Std. Scuba Tank, Yellow is in a warehouse in Atlanta
Std. Scuba Tank, Magenta is in a warehouse in Atlanta
Dive Mask, Small Clear is in a warehouse in Atlanta
Dive Mask, Med Clear is in a warehouse in Atlanta
Half-dome Tent is in a warehouse in Atlanta
Half-dome Tent Vestibule is in a warehouse in Atlanta
Light Fly Climbing Harness is in a warehouse in Atlanta
Locking Carabiner, Oval is in a warehouse in Atlanta
Std. Scuba Tank, Yellow is in a warehouse in Chicago
Std. Scuba Tank, Magenta is in a warehouse in Chicago
Dive Mask, Small Clear is in a warehouse in Chicago
Dive Mask, Med Clear is in a warehouse in Chicago
Half-dome Tent is in a warehouse in Chicago
Half-dome Tent Vestibule is in a warehouse in Chicago
Light Fly Climbing Harness is in a warehouse in Chicago
Locking Carabiner, Oval is in a warehouse in Chicago
Std. Scuba Tank, Yellow is in a warehouse in Bangor
Std. Scuba Tank, Magenta is in a warehouse in Bangor
Dive Mask, Small Clear is in a warehouse in Bangor
Dive Mask, Med Clear is in a warehouse in Bangor
Half-dome Tent is in a warehouse in Bangor
Half-dome Tent Vestibule is in a warehouse in Bangor
Light Fly Climbing Harness is in a warehouse in Bangor
Locking Carabiner, Oval is in a warehouse in Bangor
Std. Scuba Tank, Yellow is in a warehouse in Seattle
Std. Scuba Tank, Magenta is in a warehouse in Seattle
Dive Mask, Small Clear is in a warehouse in Seattle
Dive Mask, Med Clear is in a warehouse in Seattle
Half-dome Tent is in a warehouse in Seattle
Half-dome Tent Vestibule is in a warehouse in Seattle
Light Fly Climbing Harness is in a warehouse in Seattle
Locking Carabiner, Oval is in a warehouse in Seattle

Record: 1 of 32 No Filter Search

2.45 Write an SQL statement to show the SKU, SKU_Description, WarehouseID for all items stored in a warehouse managed by 'Lucille Smith'. Use a subquery.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    SKU, SKU_Description, WarehouseID
FROM      INVENTORY
WHERE     WarehouseID IN
          (SELECT      WarehouseID
           FROM     WAREHOUSE
           WHERE       Manager = 'Lucille Smith');
```

SQL-Query-CH02-RQ-02-45

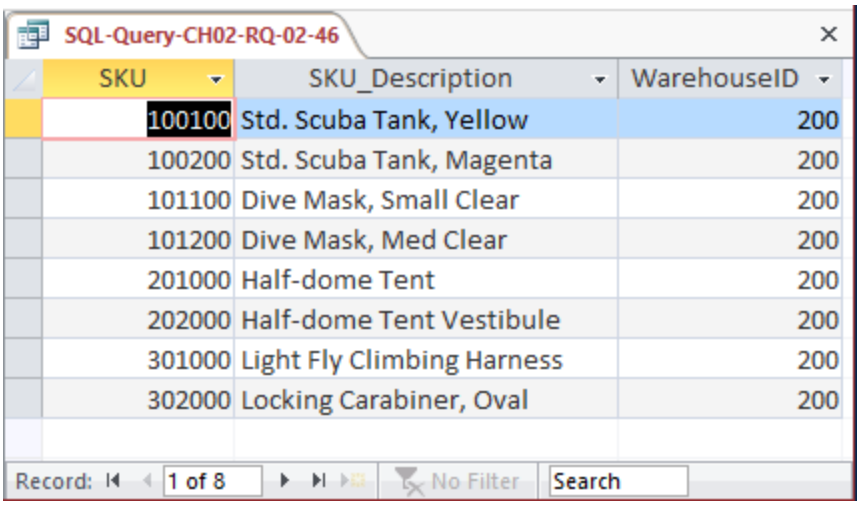
SKU	SKU_Description	WarehouseID
100100	Std. Scuba Tank, Yellow	200
100200	Std. Scuba Tank, Magenta	200
101100	Dive Mask, Small Clear	200
101200	Dive Mask, Med Clear	200
201000	Half-dome Tent	200
202000	Half-dome Tent Vestibule	200
301000	Light Fly Climbing Harness	200
302000	Locking Carabiner, Oval	200

Record: 1 of 8 No Filter Search

2.46 Write an SQL statement to show the SKU, SKU_Description, WarehouseID for all items stored in a warehouse managed by 'Lucille Smith'. Use a join, but do not use JOIN ON syntax.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT       SKU, SKU_Description, WAREHOUSE.WarehouseID
FROM      INVENTORY, WAREHOUSE
WHERE     INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
      AND       Manager = 'Lucille Smith';
```


SQL-Query-CH02-RQ-02-46

SKU	SKU_Description	WarehouseID
100100	Std. Scuba Tank, Yellow	200
100200	Std. Scuba Tank, Magenta	200
101100	Dive Mask, Small Clear	200
101200	Dive Mask, Med Clear	200
201000	Half-dome Tent	200
202000	Half-dome Tent Vestibule	200
301000	Light Fly Climbing Harness	200
302000	Locking Carabiner, Oval	200

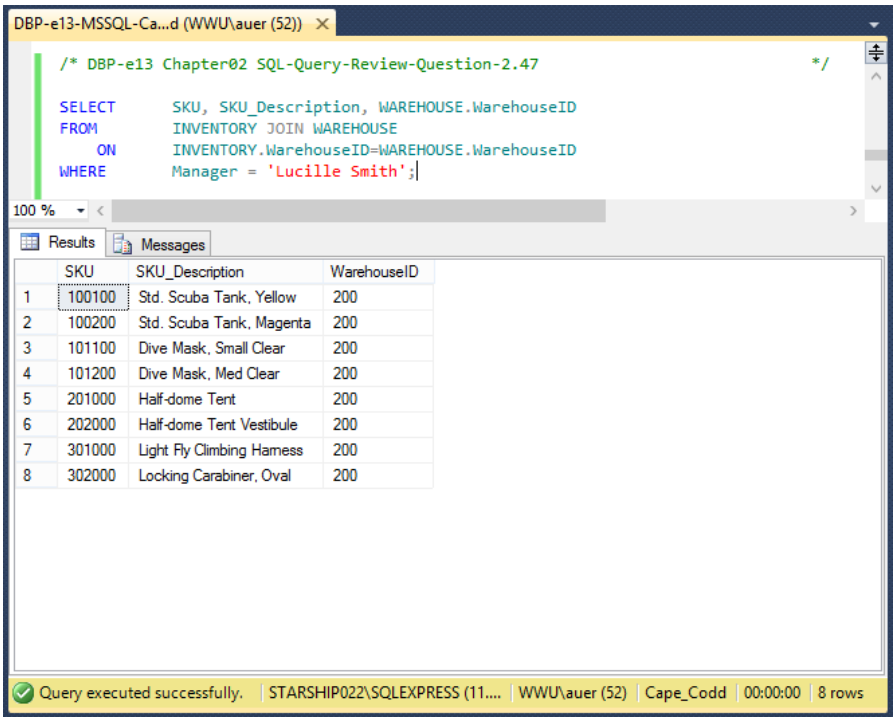
Record: 1 of 8 No Filter Search

2.47 Write an SQL statement to show the SKU, SKU_Description, WarehouseID for all items stored in a warehouse managed by 'Lucille Smith'. Use a join using JOIN ON syntax.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT      SKU, SKU_Description, WAREHOUSE.WarehouseID
FROM        INVENTORY JOIN WAREHOUSE
      ON    INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
WHERE       Manager = 'Lucille Smith';
```


DBP-e13-MSSQL-Ca...d (WWU\auer (52))

```
/* DBP-e13 Chapter02 SQL-Query-Review-Question-2.47                    */

SELECT      SKU, SKU_Description, WAREHOUSE.WarehouseID
FROM        INVENTORY JOIN WAREHOUSE
    ON      INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
WHERE       Manager = 'Lucille Smith';
```

100 %

Results Messages

	SKU	SKU_Description	WarehouseID
1	100100	Std. Scuba Tank, Yellow	200
2	100200	Std. Scuba Tank, Magenta	200
3	101100	Dive Mask, Small Clear	200
4	101200	Dive Mask, Med Clear	200
5	201000	Half-dome Tent	200
6	202000	Half-dome Tent Vestibule	200
7	301000	Light Fly Climbing Harness	200
8	302000	Locking Carabiner, Oval	200

Query executed successfully. | STARSHIP022\SQLEXPRESS (11.... | WWU\auer (52) | Cape_Codd | 00:00:00 | 8 rows

For Microsoft Access:

Microsoft Access requires the SQL JOIN ON syntax INNER JOIN instead of just JOIN:

For Microsoft SQL Server, Oracle Database and MySQL:

```
SELECT      SKU, SKU_Description, WAREHOUSE.WarehouseID
FROM        INVENTORY INNER JOIN WAREHOUSE
      ON    INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
WHERE       Manager = 'Lucille Smith';
```

SQL-Query-CH02-RQ-02-47

SKU	SKU_Description	WarehouseID
100100	Std. Scuba Tank, Yellow	200
100200	Std. Scuba Tank, Magenta	200
101100	Dive Mask, Small Clear	200
101200	Dive Mask, Med Clear	200
201000	Half-dome Tent	200
202000	Half-dome Tent Vestibule	200
301000	Light Fly Climbing Harness	200
302000	Locking Carabiner, Oval	200

Record: 1 of 8 No Filter Search

2.48 Write an SQL statement to show the WarehouseID and average QuantityOnHand of all items stored in a warehouse managed by 'Lucille Smith'. Use a subquery.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      WarehouseID,
            AVG(QuantityOnHand) AS AverageQuantityOnHand
FROM        INVENTORY
WHERE       WarehouseID IN
            (SELECT     WarehouseID
             FROM       WAREHOUSE
             WHERE      Manager = 'Lucille Smith')
GROUP BY    WarehouseID;
```

SQL-Query-CH02-RQ-02-48

WarehouseID	AverageQuantityOnHand
200	217

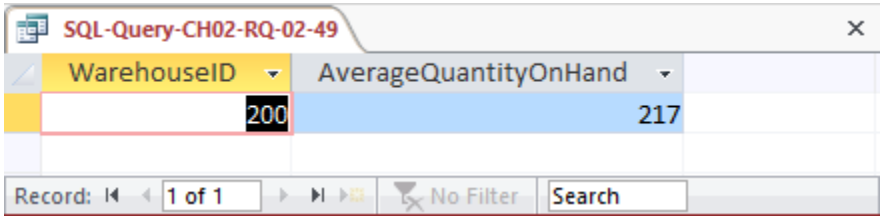
Record: 1 of 1 No Filter Search

2.49 Write an SQL statement to show the WarehouseID and average QuantityOnHand of all items stored in a warehouse managed by 'Lucille Smith'. Use a join, but do not use JOIN ON syntax.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      INVENTORY.WarehouseID,
            AVG(QuantityOnHand) AS AverageQuantityOnHand
FROM        INVENTORY, WAREHOUSE
WHERE       INVENTORY.WarehouseID = WAREHOUSE.WarehouseID
   AND      Manager = 'Lucille Smith'
GROUP BY    INVENTORY.Warehouse.ID;
```

Note the use of the complete references to **INVENTORY.Warehouse**—the query will NOT work without them.

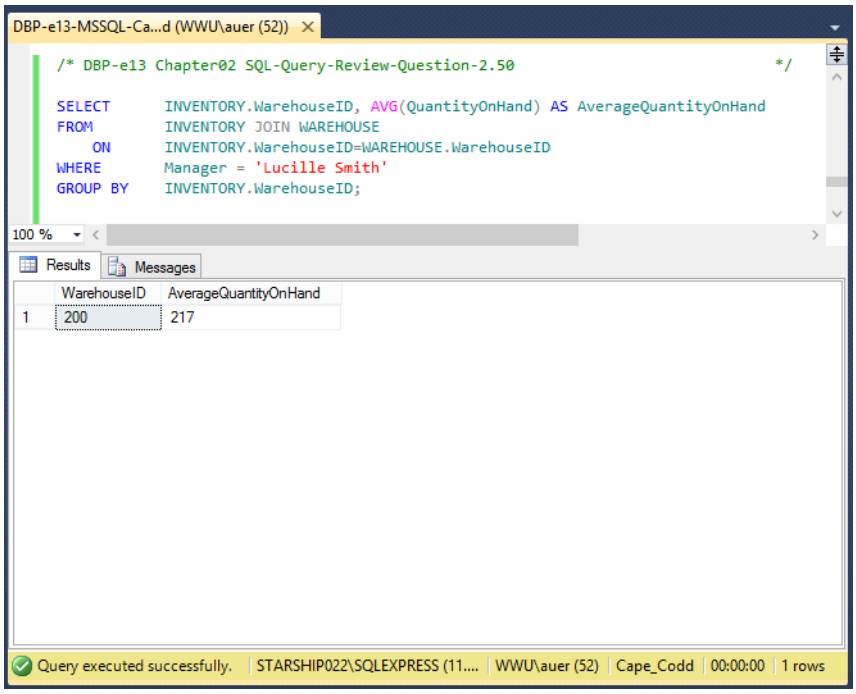


2.50 Write an SQL statement to show the WarehouseID and average QuantityOnHand of all items stored in a warehouse managed by 'Lucille Smith'. Use a join using JOIN ON syntax.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

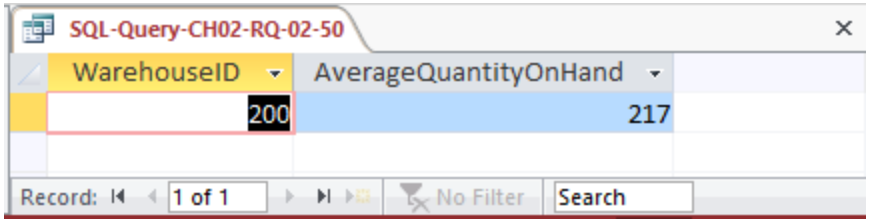
```
SELECT      INVENTORY.WarehouseID,
            AVG(QuantityOnHand) AS AverageQuantityOnHand
FROM        INVENTORY JOIN WAREHOUSE
      ON    INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
WHERE       Manager = 'Lucille Smith'
GROUP BY    INVENTORY.WarehouseID;
```



For Microsoft Access:

Microsoft Access requires the SQL JOIN ON syntax INNER JOIN instead of just JOIN:

```
SELECT      INVENTORY.WarehouseID,
            AVG(QuantityOnHand) AS AverageQuantityOnHand
FROM        INVENTORY INNER JOIN WAREHOUSE
      ON    INVENTORY.WarehouseID=WAREHOUSE.WarehouseID
WHERE       Manager = 'Lucille Smith'
GROUP BY    INVENTORY.WarehouseID;
```



2.51 Write an SQL statement to display the WarehouseID, the sum of QuantityOnOrder and sum of QuantityOnHand, grouped by WarehouseID and QuantityOnOrder. Name the sum of QuantityOnOrder as TotalItemsOnOrder and the sum of QuantityOnHand as TotalItemsOnHand.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT      WarehouseID,
            SUM(QuantityOnOrder) AS TotalItemsOnOrder,
            SUM(QuantityOnHand) AS TotalItemsOnHand
FROM        INVENTORY
GROUP BY    WarehouseID, QuantityOnHand;
```

SQL-Query-CH02-RQ-02-51

WarehouseID	TotalItemsOnOrder	TotalItemsOnHand
100	500	0
100	100	2
100	250	10
100	500	100
100	30	200
100	0	250
100	250	300
100	0	1000
200	500	0
200	250	1
200	250	10
200	500	50
200	75	75
200	50	100
200	250	250
200	0	1250
300	250	0
300	100	300
300	0	250
300	200	300
300	0	475
300	500	500
400	1700	0
400	0	200
400	250	500
400	0	450

Record: 1 of 26 No Filter Search

2.52 Write an SQL statement to show the WarehouseID, WarehouseCity, WarehouseState, Manager, SKU, SKU_Description, and QuantityOnHand of all items with a Manager of 'Lucille Smith'. Use a join.

SQL Solutions to Project Questions 2.17 – 2.52 are contained in the Microsoft Access database *DBP-e13-IM-CH02-Cape-Codd-RQ.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
SELECT    W.WarehouseID, WarehouseCity,
          WarehouseState, Manager,
          SKU, SKU_Description, QuantityOnHand
FROM      INVENTORY AS I, WAREHOUSE AS W
WHERE     I.WarehouseID=W.WarehouseID
   AND    Manager = 'Lucille Smith';
```

SQL-Query-CH02-RQ-02-52

WarehouseID	WarehouseCity	WarehouseState	Manager	SKU	SKU_Description	QuantityOnHand
200	Chicago	IL	Lucille Smith	100100	Std. Scuba Tank, Yellow	100
200	Chicago	IL	Lucille Smith	100200	Std. Scuba Tank, Magenta	75
200	Chicago	IL	Lucille Smith	101100	Dive Mask, Small Clear	0
200	Chicago	IL	Lucille Smith	101200	Dive Mask, Med Clear	50
200	Chicago	IL	Lucille Smith	201000	Half-dome Tent	10
200	Chicago	IL	Lucille Smith	202000	Half-dome Tent Vestibule	1
200	Chicago	IL	Lucille Smith	301000	Light Fly Climbing Harness	250
200	Chicago	IL	Lucille Smith	302000	Locking Carabiner, Oval	1250

Record: 1 of 8 No Filter Search

Note the use of the complete references to **INVENTORY.WarehouseID (aliased as I.Warehouse) and WAREHOUSE.WarehouseID (aliased as W.WarehouseID)**—the query will NOT work without them.

2.53 Explain why you cannot use a subquery in your answer to question 2.50.

In a query that contains a subquery, only data from fields in the table used in the top-level query can be included in the SELECT statement. If data from fields from other tables are also needed, a join must be used. In question 2.50 we needed to display WAREHOUSE.Manager but INVENTORY would have been the table in the top-level query. Therefore, we had to use a join.

2.54 Explain how subqueries and joins differ.

(1) In a query that contains a subquery, only data from fields in the table used in the top-level query can be included in the SELECT statement. If data from fields from other tables are also needed, a join must be used. See the answer to question 2.53.

(2) The subqueries in this chapter are **non-correlated subqueries**, which have an equivalent join structure. In Chapter 8, **correlated subqueries** will be discussed, and correlated subqueries do not have an equivalent join structure—you must use subqueries.

2.55 Write an SQL statement to show the WAREHOUSE and INVENTORY and include all rows of WAREHOUSE in your answer, regardless of whether they have any INVENTORY. Run this statement.

```
SELECT   W.WarehouseID, WarehouseCity, WarehouseState, Manager,
         SKU, SKU_Description, QuantityOnHand, QuantityOnOrder
FROM     WAREHOUSE AS W LEFT JOIN INVENTORY AS I
         ON W.WarehouseID = I.WarehouseID;
```

SQL-Query-CH02-RQ-02-55

WarehouseID	WarehouseCity	WarehouseState	Manager	SKU	SKU_Description	QuantityOnHand	QuantityOnOrder
100	Atlanta	GA	Dave Jones	100100	Std. Scuba Tank, Yellow	250	0
100	Atlanta	GA	Dave Jones	100200	Std. Scuba Tank, Magenta	200	30
100	Atlanta	GA	Dave Jones	101100	Dive Mask, Small Clear	0	500
100	Atlanta	GA	Dave Jones	101200	Dive Mask, Med Clear	100	500
100	Atlanta	GA	Dave Jones	201000	Half-dome Tent	2	100
100	Atlanta	GA	Dave Jones	202000	Half-dome Tent Vestibule	10	250
100	Atlanta	GA	Dave Jones	301000	Light Fly Climbing Harness	300	250
100	Atlanta	GA	Dave Jones	302000	Locking Carabiner, Oval	1000	0
200	Chicago	IL	Lucille Smith	100100	Std. Scuba Tank, Yellow	100	50
200	Chicago	IL	Lucille Smith	100200	Std. Scuba Tank, Magenta	75	75
200	Chicago	IL	Lucille Smith	101100	Dive Mask, Small Clear	0	500
200	Chicago	IL	Lucille Smith	101200	Dive Mask, Med Clear	50	500
200	Chicago	IL	Lucille Smith	201000	Half-dome Tent	10	250
200	Chicago	IL	Lucille Smith	202000	Half-dome Tent Vestibule	1	250
200	Chicago	IL	Lucille Smith	301000	Light Fly Climbing Harness	250	250
200	Chicago	IL	Lucille Smith	302000	Locking Carabiner, Oval	1250	0
300	Bangor	ME	Bart Evans	100100	Std. Scuba Tank, Yellow	100	0
300	Bangor	ME	Bart Evans	100200	Std. Scuba Tank, Magenta	100	100
300	Bangor	ME	Bart Evans	101100	Dive Mask, Small Clear	300	200
300	Bangor	ME	Bart Evans	101200	Dive Mask, Med Clear	475	0
300	Bangor	ME	Bart Evans	201000	Half-dome Tent	250	0
300	Bangor	ME	Bart Evans	202000	Half-dome Tent Vestibule	100	0
300	Bangor	ME	Bart Evans	301000	Light Fly Climbing Harness	0	250
300	Bangor	ME	Bart Evans	302000	Locking Carabiner, Oval	500	500
400	Seattle	WA	Dale Rogers	100100	Std. Scuba Tank, Yellow	200	0
400	Seattle	WA	Dale Rogers	100200	Std. Scuba Tank, Magenta	250	0
400	Seattle	WA	Dale Rogers	101100	Dive Mask, Small Clear	450	0
400	Seattle	WA	Dale Rogers	101200	Dive Mask, Med Clear	250	250
400	Seattle	WA	Dale Rogers	201000	Half-dome Tent	0	250
400	Seattle	WA	Dale Rogers	202000	Half-dome Tent Vestibule	0	200
400	Seattle	WA	Dale Rogers	301000	Light Fly Climbing Harness	0	250
400	Seattle	WA	Dale Rogers	302000	Locking Carabiner, Oval	0	1000
500	San Francisco	CA	Grace Jefferson				

Record: 1 of 33 No Filter Search

ANSWERS TO PROJECT QUESTIONS

For this set of project questions, we will continue creating a Microsoft Access database for the Wedgewood Pacific Corporation (WPC) that we created in Chapter 1. Founded in 1957 in Seattle, Washington, WPC has grown into an internationally recognized organization. The company is located in two buildings. One building houses the Administration, Accounting, Finance, and Human Resources departments, and the second houses the Production, Marketing, and Information Systems departments. The company database contains data about company employees, departments, company projects, company assets such as computer equipment, and other aspects of company operations.

In the following project questions, we have already created the WPC.accdb database with the following two tables (see Chapter 1 Project Questions):

DEPARTMENT (DepartmentName, BudgetCode, OfficeNumber, Phone)

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

Now we will add in the following two tables:

PROJECT (ProjectID, Name, *Department*, MaxHours, StartDate, EndDate)

ASSIGNMENT (*ProjectID*, *EmployeeNumber*, HoursWorked)

The four tables in the revised WPC database schema are shown in Figure 2-28. The column characteristics for the PROJECT table are shown in Figure 2-29, and the column characteristics for the ASSIGNMENT table are shown in Figure 2-31. Data for the PROJECT table are shown in Figure 2-30, and the data for the ASSIGNMENT table are shown in Figure 2-32.

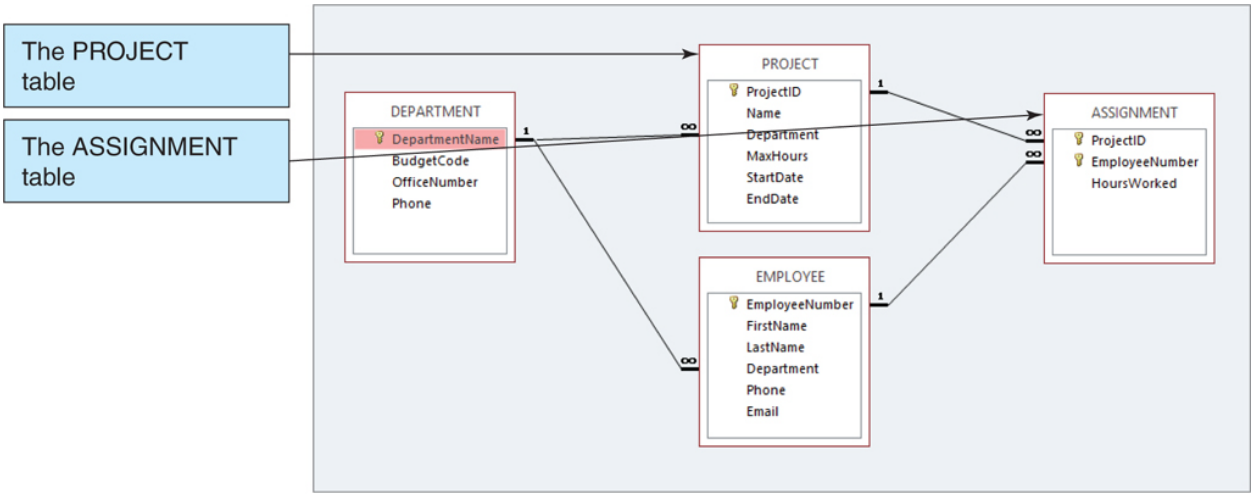


Figure 2-29 – The WPC Database with the PROJECT and ASSIGNMENT Tables

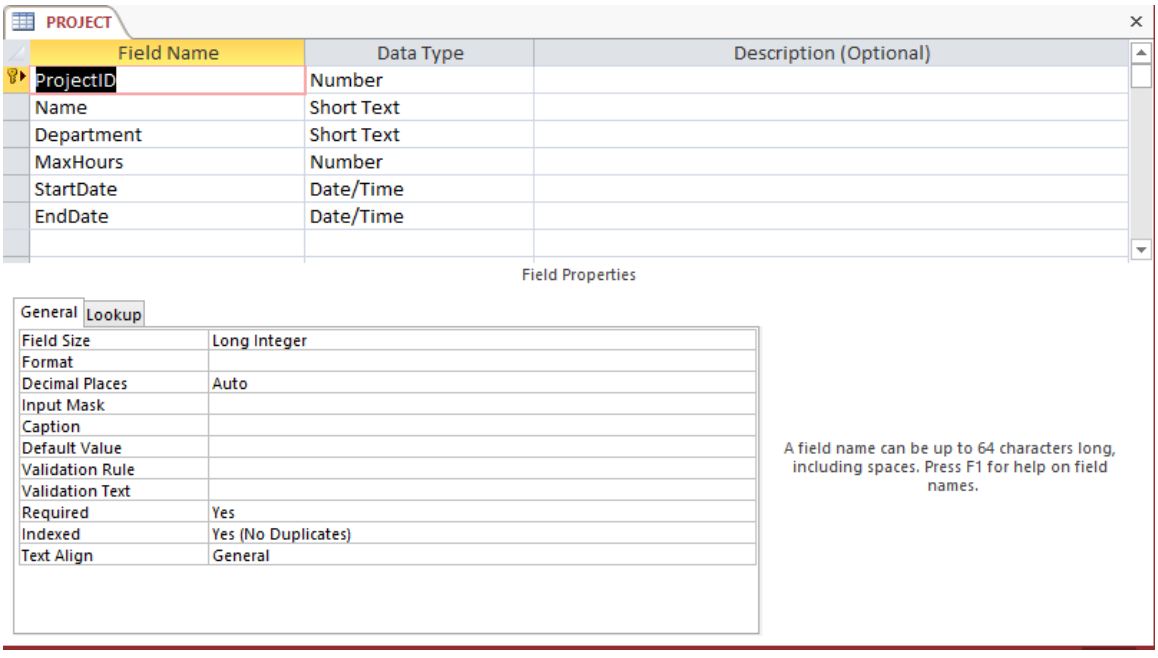
2.56 Figure 2-29 shows the column characteristics for the WPC PROJECT table. Using the column characteristics, create the PROJECT table in the WPC.accdb database.

SQL Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

PROJECT

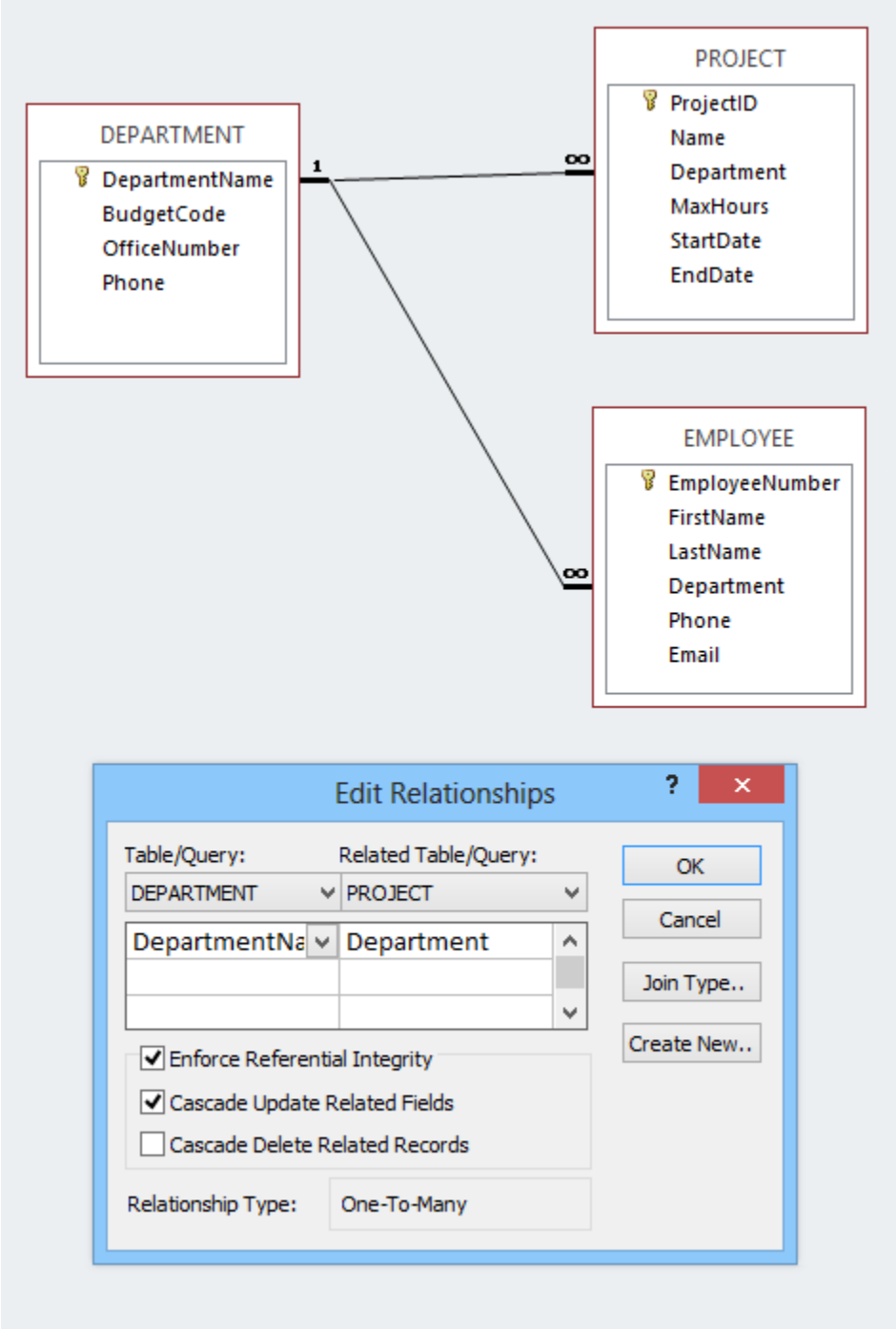
Column Name	Type	Key	Required	Remarks
ProjectID	Number	Primary Key	Yes	Long Integer
Name	Text (50)	No	Yes	
Department	Text (35)	Foreign Key	Yes	
MaxHours	Number	No	Yes	Double
StartDate	Date	No	No	
EndDate	Date	No	No	

Figure 2-30 - Column Characteristics for the PROJECT Table



2.57 *Create the relationship and referential integrity constraint between PROJECT and DEPARTMENT. Enable enforcing of referential integrity and cascading of data updates, but do not enable cascading of data from deleted records.*

SQL Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

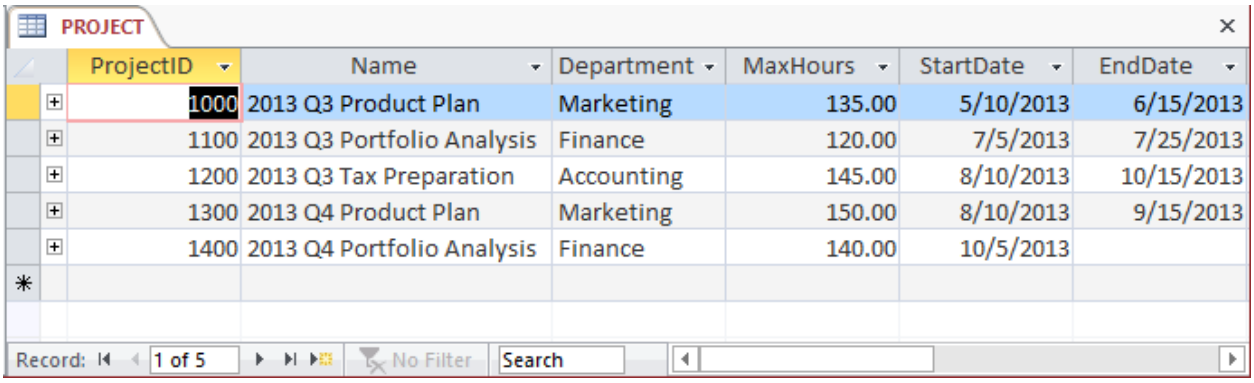


2.58 Figure 2-31 shows the data for the WPC PROJECT table. Using the Datasheet view, enter the data shown in Figure 2-27 into your PROJECT table.

Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

ProjectID	Name	Department	MaxHours	StartDate	EndDate
1000	2013 Q3 Product Plan	Marketing	135.00	10-MAY-13	15-JUN-13
1100	2013 Q3 Portfolio Analysis	Finance	120.00	07-JUL-13	25-JUL-13
1200	2013 Q3 Tax Preparation	Accounting	145.00	10-AUG-13	15-OCT-13
1300	2013 Q4 Product Plan	Marketing	150.00	10-AUG-13	15-SEP-13
1400	2013 Q4 Portfolio Analysis	Finance	140.00	05-OCT-13	

Figure 2-31 - Sample Data for the PROJECT Table



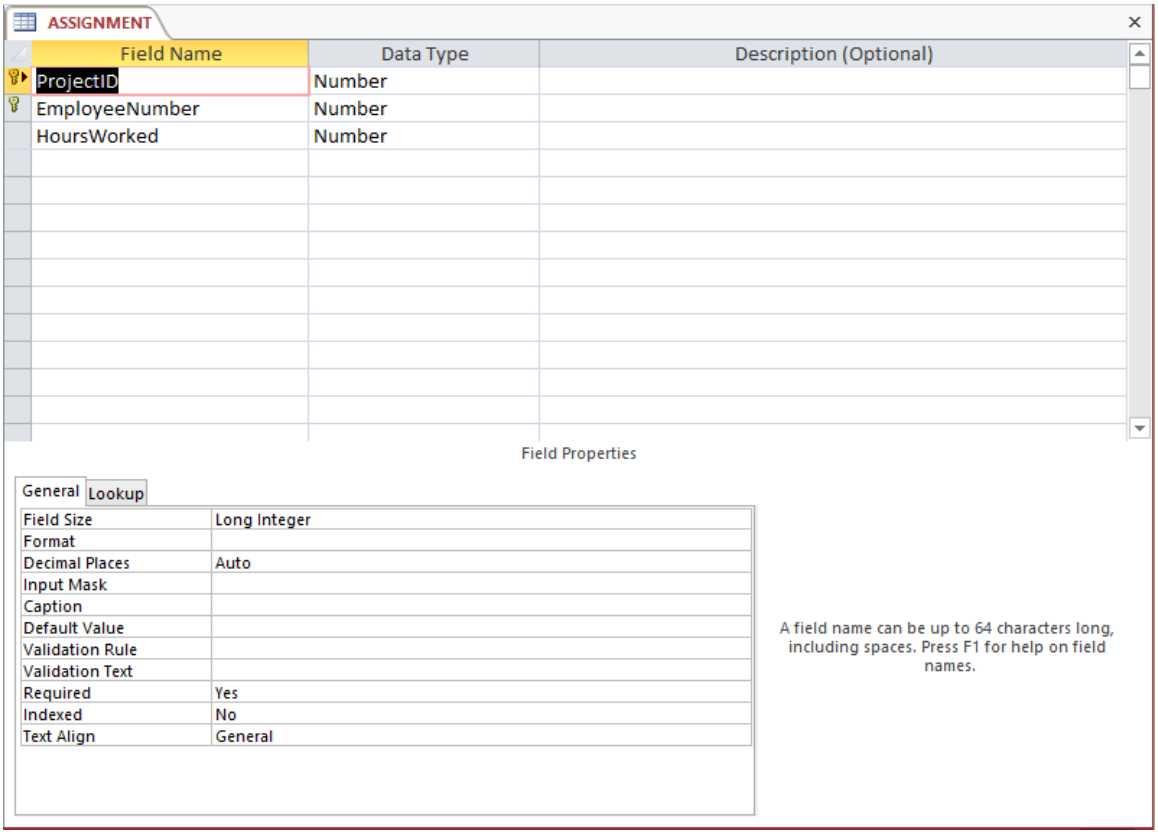
2.59 Figure 2-32 shows the column characteristics for the WPC ASSIGNMENT table. Using the column characteristics, create the ASSIGNMENT table in the WPC.accdb database.

Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

ASSIGNMENT

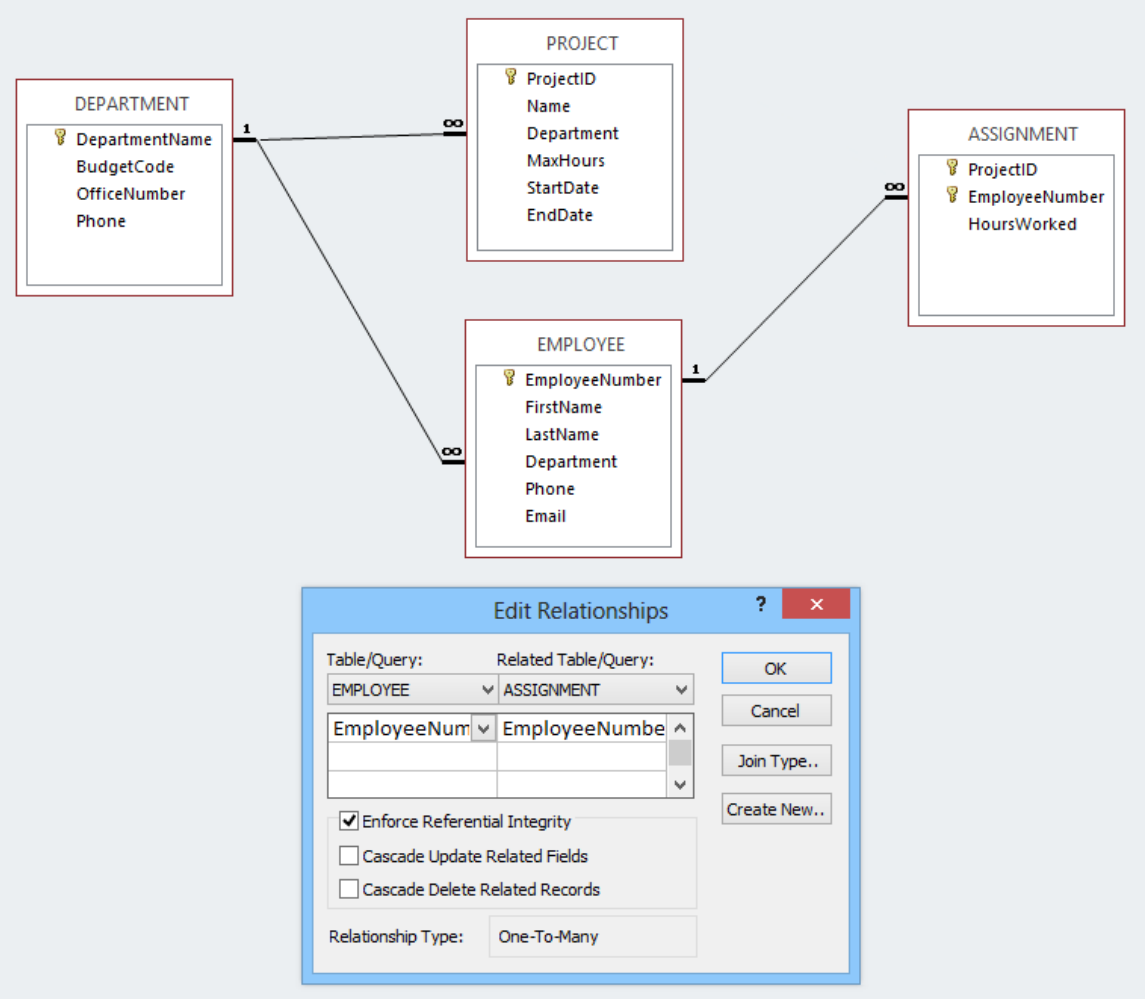
Column Name	Type	Key	Required	Remarks
ProjectID	Number	Primary Key, Foreign Key	Yes	Long Integer
EmployeeNumber	Number	Primary Key, Foreign Key	Yes	Long Integer
HoursWorked	Number	No	No	Double

Figure 2-32 - Column Characteristics for the ASSIGNMENT Table



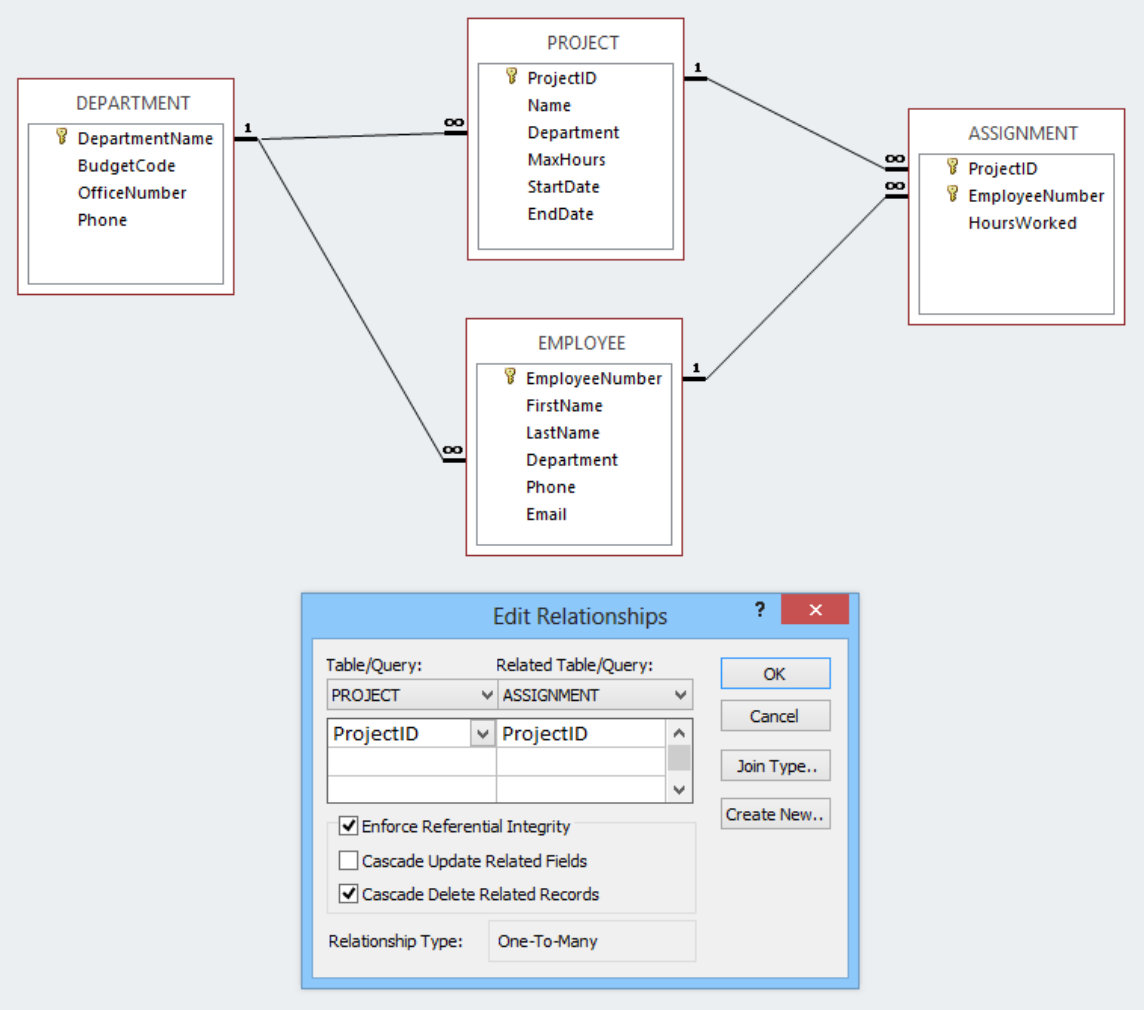
2.60 Create the relationship and referential integrity constraint between ASSIGNMENT and EMPLOYEE. Enable enforcing of referential integrity, but do not enable either cascading updates or the cascading of data from deleted records.

Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).



2.61 Create the relationship and referential integrity constraint between ASSIGNMENT and PROJECT. Enable enforcing of referential integrity and cascading of deletes, but do not enable cascading updates.

Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

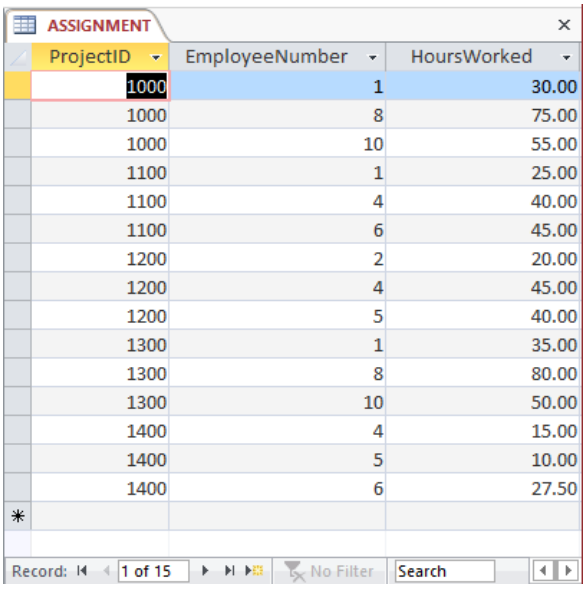


2.62 Figure 2-33 shows the data for the WPC ASSIGNMENT table. Using the Datasheet view, enter the data shown in Figure 2-33 into your ASSIGNMENT table.

Solutions to Project Questions 2.56 – 2.65 are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

ProjectID	EmployeeNumber	HoursWorked
1000	1	30.0
1000	8	75.0
1000	10	55.0
1100	4	40.0
1100	6	45.0
1100	1	25.0
1200	2	20.0
1200	4	45.0
1200	5	40.0
1300	1	35.0
1300	8	80.0
1300	10	50.0
1400	4	15.0
1400	5	10.0
1400	6	27.5

Figure 2-33 - Sample Data for the ASSIGNEMENT Table



2.63 In Project Question 2.58, the table data was entered after referential integrity constraints were created in Project Question 2.57. In Project Question 2.62, the table data was entered after referential integrity constraints were created in Project Questions 2.59 and 2.60. Why was the data entered after the referential integrity constraints were created instead of before the constraints were created?

Both the PROJECT and ASSIGNMENT tables have foreign keys. PROJECT.Department is the foreign key in PROJECT, and both ASSIGNMENT.ProjectID and ASSIGNMENT.EmployeeNumber are foreign keys in ASSIGNMENT, If data was entered into these columns before the referential integrity constraints were established, it would be possible to enter foreign key data that had no corresponding primary key data. Thus, we establish the referential integrity constraints so that the DBMS will not allow inconsistent data to be entered into the foreign key columns.

2.64 Using Access SQL, create and run queries to answer the following questions. Save each query using the query name format SQL-Query-02-##, where the ## sign is replaced by the letter designator of the question. For example, the first query will be saved as SQL-Query-02-A.Write SQL queries to produce the following results:

Solutions to Project Questions 2.64 A–N are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

A. What projects are in the PROJECT table? Show all information for each project.

```
/*****   Question A - SQL-Query-02-A   ************************/

SELECT * FROM PROJECT;
```

SQL-Query-02-A

ProjectID	Name	Department	MaxHours	StartDate	EndDate
1000	2013 Q3 Product Plan	Marketing	135.00	5/10/2013	6/15/2013
1100	2013 Q3 Portfolio Analysis	Finance	120.00	7/5/2013	7/25/2013
1200	2013 Q3 Tax Preparation	Accounting	145.00	8/10/2013	10/15/2013
1300	2013 Q4 Product Plan	Marketing	150.00	8/10/2013	9/15/2013
1400	2013 Q4 Portfolio Analysis	Finance	140.00	10/5/2013	

Record: 1 of 5 No Filter Search

B. What are the ProjectID, Name, StartDate, and EndDate values of projects in the PROJECT table?

```
/*****    Question B - SQL-Query-02-B    **********************/

SELECT     ProjectID, Name, StartDate, EndDate
FROM       PROJECT;
```

SQL-Query-02-B

ProjectID	Name	StartDate	EndDate
1000	2013 Q3 Product Plan	5/10/2013	6/15/2013
1100	2013 Q3 Portfolio Analysis	7/5/2013	7/25/2013
1200	2013 Q3 Tax Preparation	8/10/2013	10/15/2013
1300	2013 Q4 Product Plan	8/10/2013	9/15/2013
1400	2013 Q4 Portfolio Analysis	10/5/2013	

Record: 1 of 5 No Filter Search

C. What projects in the PROJECT table started before August 1, 2013? Show all the information for each project.

Note that the answer is an **empty set**—there are **no** PROJECTs that were started before **August 1, 2013**. This answer may surprise students, but it is the correct and intended answer. Point out in class that sometimes the results of a query will be an empty set. Then ask your class to rerun the query with the dates **August 1, 2012** and **August 1, 2014** and compare the results of the three queries.

```
/*****    Question C - SQL-Query-02-C    **********************/

SELECT     *
FROM       PROJECT
WHERE      StartDate < #01-AUG-13#;
```

SQL-Query-02-C

ProjectID	Name	Department	MaxHours	StartDate	EndDate
1000	2013 Q3 Product Plan	Marketing	135.00	5/10/2013	6/15/2013
1100	2013 Q3 Portfolio Analysis	Finance	120.00	7/5/2013	7/25/2013

Record: 1 of 2 No Filter Search

D. What projects in the PROJECT table have not been completed? Show all the information for each project.

```
/*****   Question D - SQL-Query-02-D   **********************/

SELECT     *
FROM       PROJECT
WHERE      EndDate IS NULL;
```

SQL-Query-02-D

ProjectID	Name	Department	MaxHours	StartDate	EndDate
1400	2013 Q4 Portfolio Analysis	Finance	140.00	10/5/2013	

Record: 1 of 1 No Filter Search

E. Who are the employees assigned to each project? Show ProjectID, Employee-Number, LastName, FirstName, and Phone.

```
/*****   Question E - SQL-Query-02-E   **********************/

SELECT     ProjectID, E.EmployeeNumber, LastName, FirstName, Phone
FROM       ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
           ON A.EmployeeNumber=E.EmployeeNumber;
```

SQL-Query-02-E

ProjectID	EmployeeNumber	LastName	FirstName	Phone
1000	1	Jacobs	Mary	360-285-8110
1100	1	Jacobs	Mary	360-285-8110
1300	1	Jacobs	Mary	360-285-8110
1200	2	Jackson	Rosalie	360-285-8120
1100	4	Caruthers	Tom	360-285-8310
1200	4	Caruthers	Tom	360-285-8310
1400	4	Caruthers	Tom	360-285-8310
1200	5	Jones	Heather	360-285-8320
1400	5	Jones	Heather	360-285-8320
1100	6	Abernathy	Mary	360-285-8410
1400	6	Abernathy	Mary	360-285-8410
1000	8	Jackson	Tom	360-287-8610
1300	8	Jackson	Tom	360-287-8610
1000	10	Numoto	Ken	360-287-8710
1300	10	Numoto	Ken	360-287-8710
	(New)			

Record: 1 of 15 No Filter Search

F. Who are the employees assigned to each project? Show the ProjectID, Name, and Department. Show EmployeeNumber, LastName, FirstName, and Phone.

Note the use of the aliases **ProjectName**, **ProjectDepartment**, **DepartmentPhone** and **EmployeePhone**)

```
/*****   Question F - SQL-Query-02-F   **********************/

SELECT     P.ProjectID, Name AS ProjectName,
           P.Department AS ProjectDepartment,
           E.EmployeeNumber, LastName, FirstName,
           Phone AS EmployeePhone
FROM       (ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
              ON A.EmployeeNumber=E.EmployeeNumber)
           INNER JOIN PROJECT AS P
              ON A.ProjectID=P.ProjectID;
```

SQL-Query-02-F

ProjectID	ProjectName	ProjectDepartment	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	1	Jacobs	Mary	360-285-8110
1000	2013 Q3 Product Plan	Marketing	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	10	Numoto	Ken	360-287-8710
1100	2013 Q3 Portfolio Analysis	Finance	4	Caruthers	Tom	360-285-8310
1100	2013 Q3 Portfolio Analysis	Finance	6	Abernathy	Mary	360-285-8410
1100	2013 Q3 Portfolio Analysis	Finance	1	Jacobs	Mary	360-285-8110
1200	2013 Q3 Tax Preparation	Accounting	2	Jackson	Rosalie	360-285-8120
1200	2013 Q3 Tax Preparation	Accounting	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	5	Jones	Heather	360-285-8320
1300	2013 Q4 Product Plan	Marketing	1	Jacobs	Mary	360-285-8110
1300	2013 Q4 Product Plan	Marketing	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	10	Numoto	Ken	360-287-8710
1400	2013 Q4 Portfolio Analysis	Finance	4	Caruthers	Tom	360-285-8310
1400	2013 Q4 Portfolio Analysis	Finance	5	Jones	Heather	360-285-8320
1400	2013 Q4 Portfolio Analysis	Finance	6	Abernathy	Mary	360-285-8410

Record: 1 of 15 No Filter Search

G. *Who are the employees assigned to each project? Show ProjectID, Name, Department, and Department Phone. Show EmployeeNumber, LastName, FirstName, and Employee Phone. Sort by ProjectID in ascending order.*

Note the use of the aliases **ProjectName**, **ProjectDepartment**, **DepartmentPhone** and **EmployeePhone**.

```
/*****   Question G - SQL-Query-02-G   ************************/

SELECT     P.ProjectID, Name AS ProjectName,
           D.DepartmentName AS ProjectDepartment,
           D.Phone AS DepartmentPhone,
           E.EmployeeNumber, LastName, FirstName,
           E.Phone AS EmployeePhone
FROM       ((ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
              ON A.EmployeeNumber=E.EmployeeNumber)
           INNER JOIN PROJECT AS P
              ON A.ProjectID=P.ProjectID)
           INNER JOIN DEPARTMENT AS D
              ON P.Department=D.DepartmentName
ORDER BY   P.ProjectID;
```

SQL-Query-02-G

ProjectID	ProjectName	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1000	2013 Q3 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	5	Jones	Heather	360-285-8320
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	2	Jackson	Rosalie	360-285-8120
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1300	2013 Q4 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	5	Jones	Heather	360-285-8320
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310

Record: 1 of 15 No Filter Search

H. *Who are the employees assigned to projects run by the marketing department? Show ProjectID, Name, Department, and Department Phone. Show EmployeeNumber, LastName, FirstName, and Employee Phone. Sort by ProjectID in ascending order.*

Note the use of the aliases **ProjectName**, **ProjectDepartment**, **DepartmentPhone** and **EmployeePhone**.

```
/*****   Question H - SQL-Query-02-H   **********************/

SELECT     P.ProjectID, Name AS ProjectName,
           D.DepartmentName AS ProjectDepartment,
           D.Phone AS DepartmentPhone,
           E.EmployeeNumber, LastName, FirstName,
           E.Phone AS EmployeePhone
FROM       ((ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
              ON A.EmployeeNumber=E.EmployeeNumber)
           INNER JOIN PROJECT AS P
              ON A.ProjectID=P.ProjectID)
           INNER JOIN DEPARTMENT AS D
              ON P.Department=D.DepartmentName
WHERE      DepartmentName='Marketing'
ORDER BY   P.ProjectID;
```

SQL-Query-02-H

ProjectID	ProjectName	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1000	2013 Q3 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1300	2013 Q4 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110

Record: 1 of 6 No Filter Search

I. *How many projects are being run by the marketing department? Be sure to assign an appropriate column name to the computed results.*

Note the use of the alias **NumberOfMarketingProjects**.

```
/*****   Question I - SQL-Query-02-I   **********************/

SELECT     COUNT(*) AS NumberOfMarketingProjects
FROM       PROJECT
WHERE      Department='Marketing';
```

SQL-Query-02-I

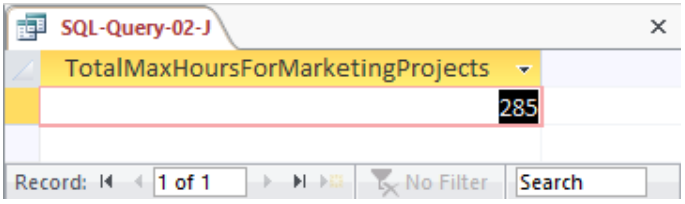
NumberOfMarketingProjects
2

Record: 1 of 1 No Filter Search

J. What is the total MaxHours of projects being run by the marketing department? Be sure to assign an appropriate column name to the computed results.

Note the use of the alias **TotalMaxHoursForMarketingProjects**.

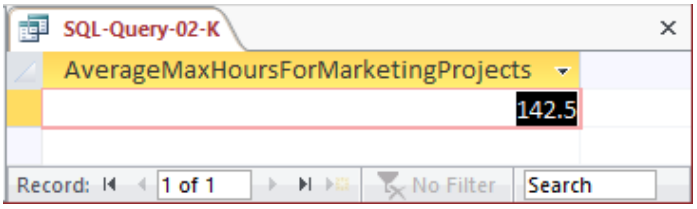
```
/*****   Question J - SQL-Query-02-J   ************************/

SELECT    SUM(MaxHours) AS TotalMaxHoursForMarketingProjects
FROM      PROJECT
WHERE     Department='Marketing';
```



K. What is the average MaxHours of projects being run by the marketing department? Be sure to assign an appropriate column name to the computed results.

Note the use of the alias **AverageMaxHoursForMarketingProjects**.

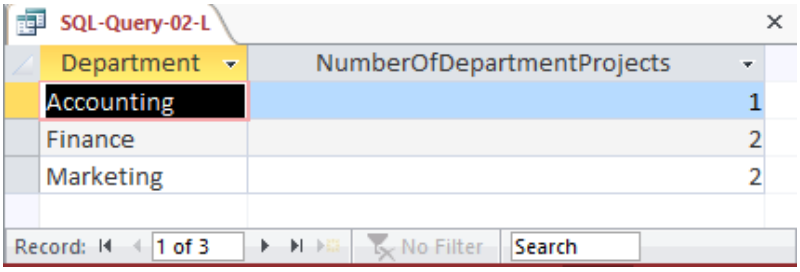
```
/*****   Question K - SQL-Query-02-K   ************************/

SELECT    AVG(MaxHours) AS AverageMaxHoursForMarketingProjects
FROM      PROJECT
WHERE     Department='Marketing';
```



L. How many projects are being run by each department? Be sure to display each DepartmentName and to assign an appropriate column name to the computed results.

Note the use of the alias **NumberOfDepartmentProjects**.

```
/*****   Question L - SQL-Query-02-L   ************************/

SELECT    Department, COUNT(*) AS NumberOfDepartmentProjects
FROM      PROJECT
GROUP BY  Department;
```



M. Write an SQL statement to join EMPLOYEE, ASSIGNMENT, and PROJECT using the JOIN ON syntax. Run this statement.

For Microsoft SQL Server, Oracle Database and MySQL:

```
/*****   Review Question  2.55 M  ***************************************/

SELECT    E.*, A.*, P.*
FROM   EMPLOYEE AS E JOIN ASSIGNMENT AS A
       ON E.EmployeeNumber = A.EmployeeNumber
          JOIN PROJECT AS P
             ON A.ProjectID = P.ProjectID;
```

For Microsoft Access:

Microsoft Access requires the SQL JOIN ON syntax INNER JOIN instead of just JOIN:

```
SELECT    E.*, A.*, P.*
FROM   (EMPLOYEE AS E INNER JOIN ASSIGNMENT AS A
       ON E.EmployeeNumber = A.EmployeeNumber)
          INNER JOIN PROJECT AS P
             ON A.ProjectID = P.ProjectID;
```

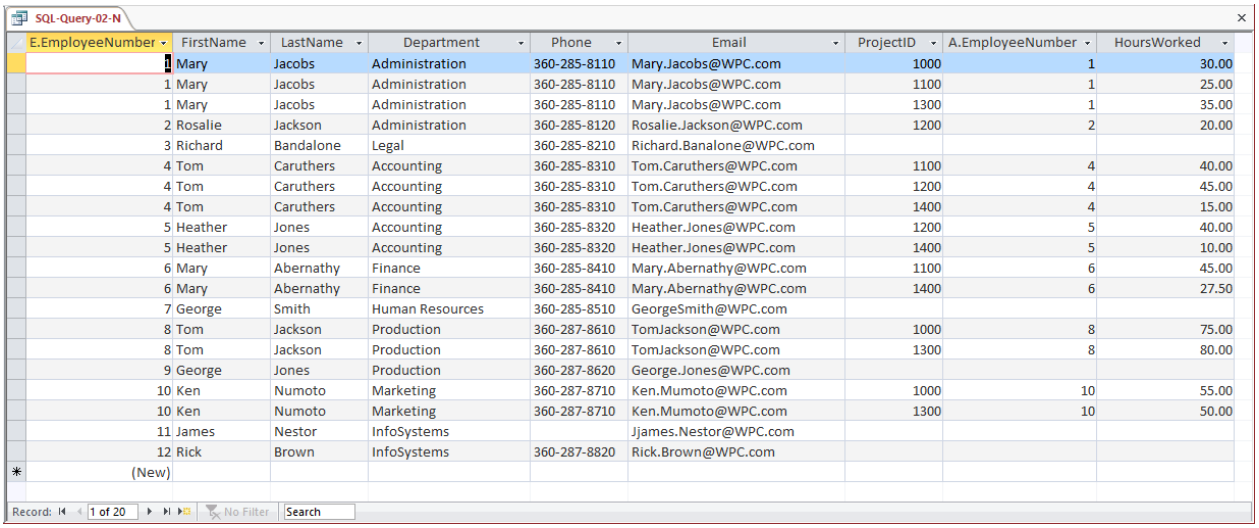
SQL-Query-02-M

E.EmployeeNum	FirstName	LastName	E.Department	Phone	Email	A.ProjectID	A.EmployeeNumber	HoursWorked	P.ProjectID	Name	P.Departme	MaxHours	StartDate	EndDate
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1000	1	30.00	1000	2013 Q3 Product Plan	Marketing	135.00	5/10/2013	6/15/2013
8	Tom	Jackson	Production	360-287-8610	TomJackson@WPC.com	1000	8	75.00	1000	2013 Q3 Product Plan	Marketing	135.00	5/10/2013	6/15/2013
10	Ken	Numoto	Marketing	360-287-8710	Ken.Mumoto@WPC.com	1000	10	55.00	1000	2013 Q3 Product Plan	Marketing	135.00	5/10/2013	6/15/2013
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1100	4	40.00	1100	2013 Q3 Portfolio Analysis	Finance	120.00	7/5/2013	7/25/2013
6	Mary	Abernathy	Finance	360-285-8410	Mary.Abernathy@WPC.com	1100	6	45.00	1100	2013 Q3 Portfolio Analysis	Finance	120.00	7/5/2013	7/25/2013
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1100	1	25.00	1100	2013 Q3 Portfolio Analysis	Finance	120.00	7/5/2013	7/25/2013
2	Rosalie	Jackson	Administration	360-285-8120	Rosalie.Jackson@WPC.com	1200	2	20.00	1200	2013 Q3 Tax Preparation	Accounting	145.00	8/10/2013	10/15/2013
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1200	4	45.00	1200	2013 Q3 Tax Preparation	Accounting	145.00	8/10/2013	10/15/2013
5	Heather	Jones	Accounting	360-285-8320	Heather.Jones@WPC.com	1200	5	40.00	1200	2013 Q3 Tax Preparation	Accounting	145.00	8/10/2013	10/15/2013
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1300	1	35.00	1300	2013 Q4 Product Plan	Marketing	150.00	8/10/2013	9/15/2013
8	Tom	Jackson	Production	360-287-8610	TomJackson@WPC.com	1300	8	80.00	1300	2013 Q4 Product Plan	Marketing	150.00	8/10/2013	9/15/2013
10	Ken	Numoto	Marketing	360-287-8710	Ken.Mumoto@WPC.com	1300	10	50.00	1300	2013 Q4 Product Plan	Marketing	150.00	8/10/2013	9/15/2013
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1400	4	15.00	1400	2013 Q4 Portfolio Analysis	Finance	140.00	10/5/2013	
5	Heather	Jones	Accounting	360-285-8320	Heather.Jones@WPC.com	1400	5	10.00	1400	2013 Q4 Portfolio Analysis	Finance	140.00	10/5/2013	
6	Mary	Abernathy	Finance	360-285-8410	Mary.Abernathy@WPC.com	1400	6	27.50	1400	2013 Q4 Portfolio Analysis	Finance	140.00	10/5/2013	
(New)														

Record: 1 of 15 No Filter Search

N. Write an SQL statement to join EMPLOYEE and ASSIGNMENT and include all rows of EMPLOYEE in your answer, regardless of whether they have an ASSIGNMENT. Run this statement.

```
/*****   Review Question  2.55 N
*****************************************/

SELECT   E.*, A.*
FROM  EMPLOYEE AS E LEFT JOIN ASSIGNMENT AS A
      ON E.EmployeeNumber = A.EmployeeNumber;
```



SQL-Query-02-N

E.EmployeeNumber	FirstName	LastName	Department	Phone	Email	ProjectID	A.EmployeeNumber	HoursWorked
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1000	1	30.00
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1100	1	25.00
1	Mary	Jacobs	Administration	360-285-8110	Mary.Jacobs@WPC.com	1300	1	35.00
2	Rosalie	Jackson	Administration	360-285-8120	Rosalie.Jackson@WPC.com	1200	2	20.00
3	Richard	Bandalone	Legal	360-285-8210	Richard.Banalone@WPC.com			
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1100	4	40.00
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1200	4	45.00
4	Tom	Caruthers	Accounting	360-285-8310	Tom.Caruthers@WPC.com	1400	4	15.00
5	Heather	Jones	Accounting	360-285-8320	Heather.Jones@WPC.com	1200	5	40.00
5	Heather	Jones	Accounting	360-285-8320	Heather.Jones@WPC.com	1400	5	10.00
6	Mary	Abernathy	Finance	360-285-8410	Mary.Abernathy@WPC.com	1100	6	45.00
6	Mary	Abernathy	Finance	360-285-8410	Mary.Abernathy@WPC.com	1400	6	27.50
7	George	Smith	Human Resources	360-285-8510	GeorgeSmith@WPC.com			
8	Tom	Jackson	Production	360-287-8610	TomJackson@WPC.com	1000	8	75.00
8	Tom	Jackson	Production	360-287-8610	TomJackson@WPC.com	1300	8	80.00
9	George	Jones	Production	360-287-8620	George.Jones@WPC.com			
10	Ken	Numoto	Marketing	360-287-8710	Ken.Mumoto@WPC.com	1000	10	55.00
10	Ken	Numoto	Marketing	360-287-8710	Ken.Mumoto@WPC.com	1300	10	50.00
11	James	Nestor	InfoSystems		Jjames.Nestor@WPC.com			
12	Rick	Brown	InfoSystems	360-287-8820	Rick.Brown@WPC.com			
(New)								

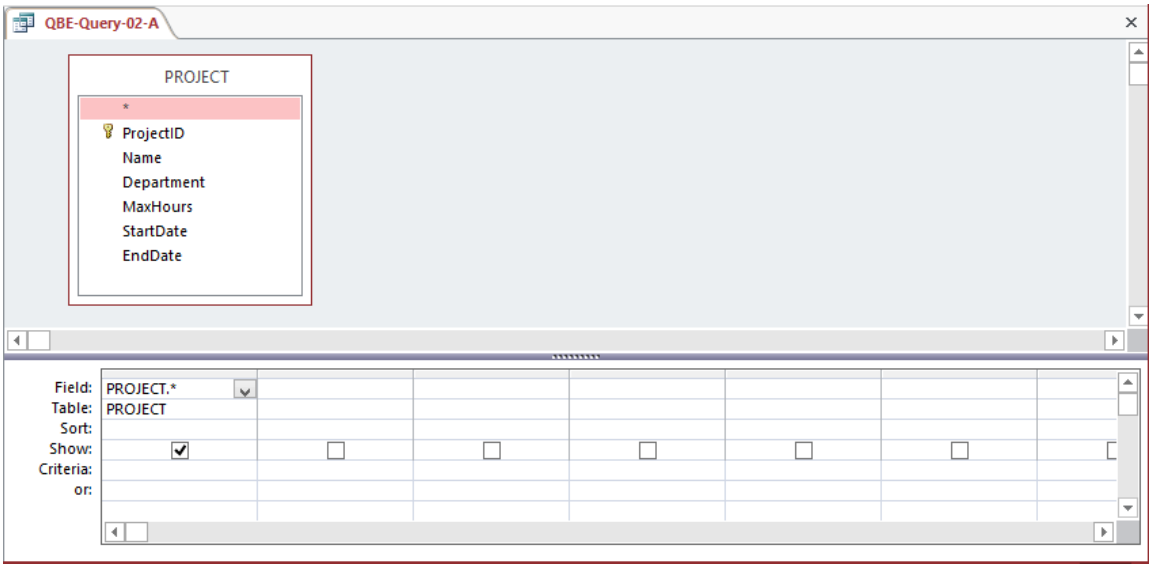
Record: 1 of 20 No Filter Search

2.65 Using Access QBE, create and run new queries to answer the questions in exercise 2.64. Save each query using the query name format QBE-Query-02-##, where the ## sign is replaced by the letter designator of the question. For example, the first query will be saved as QBE-Query-02-A.

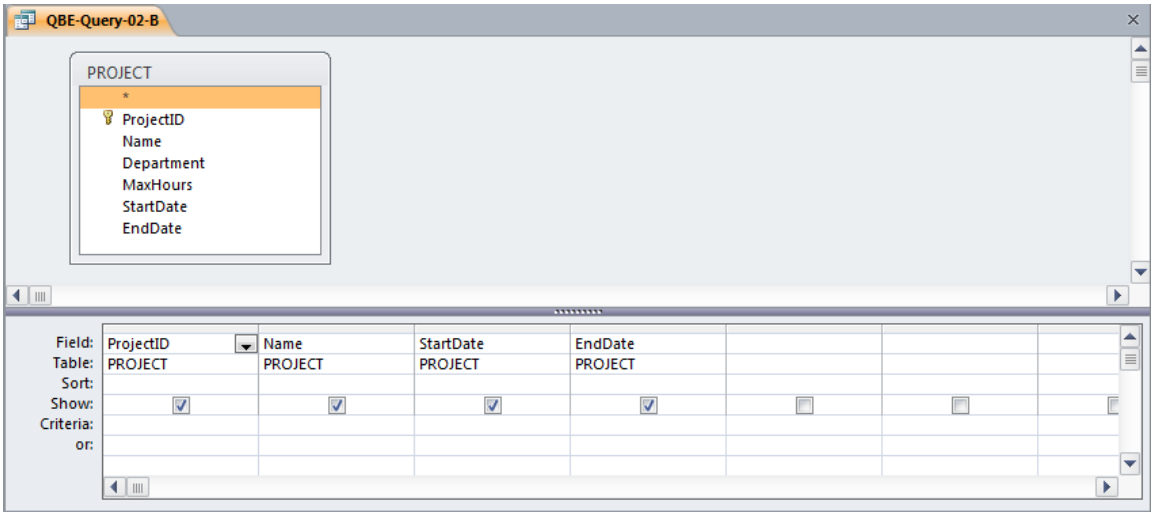
Solutions to Project Questions 2.65 A–N are contained in the Microsoft Access database *DBP-e13-IM-CH02-WPC.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

The results of each query will be identical to the corresponding SQL query in the previous Project Question. Here we will show the QBE design of the query.

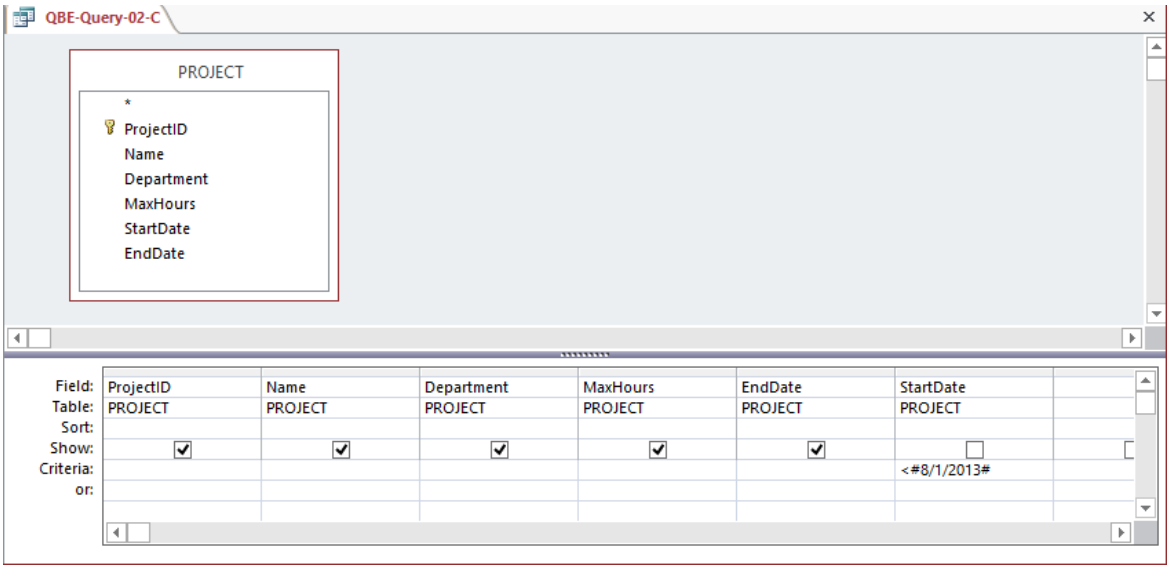
A. What projects are in the PROJECT table? Show all information for each project.



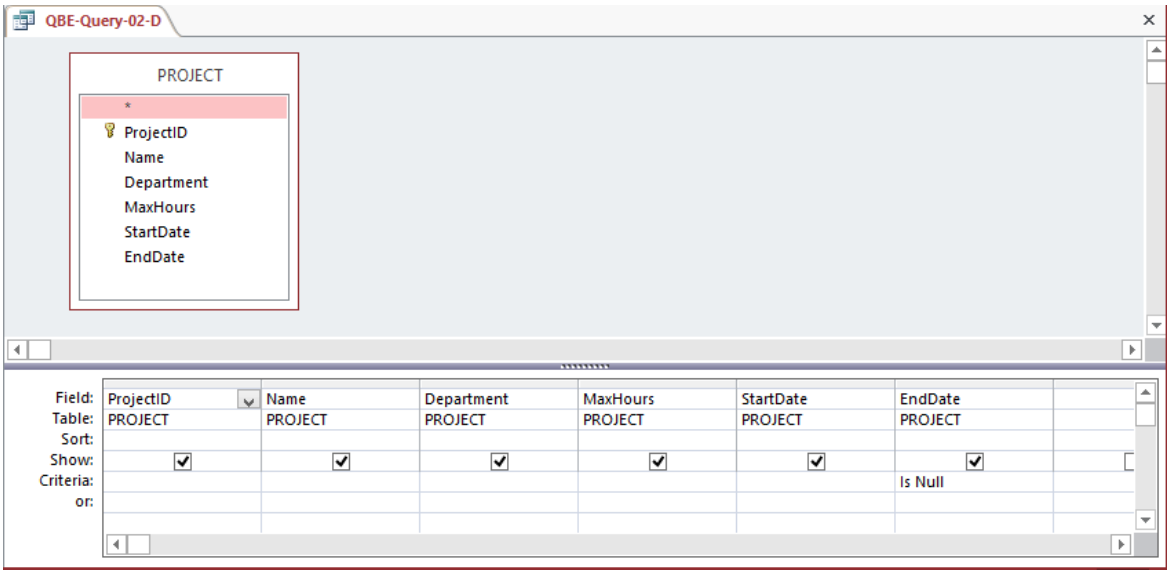
B. What are the ProjectID, Name, StartDate, and EndDate values of projects in the PROJECT table?



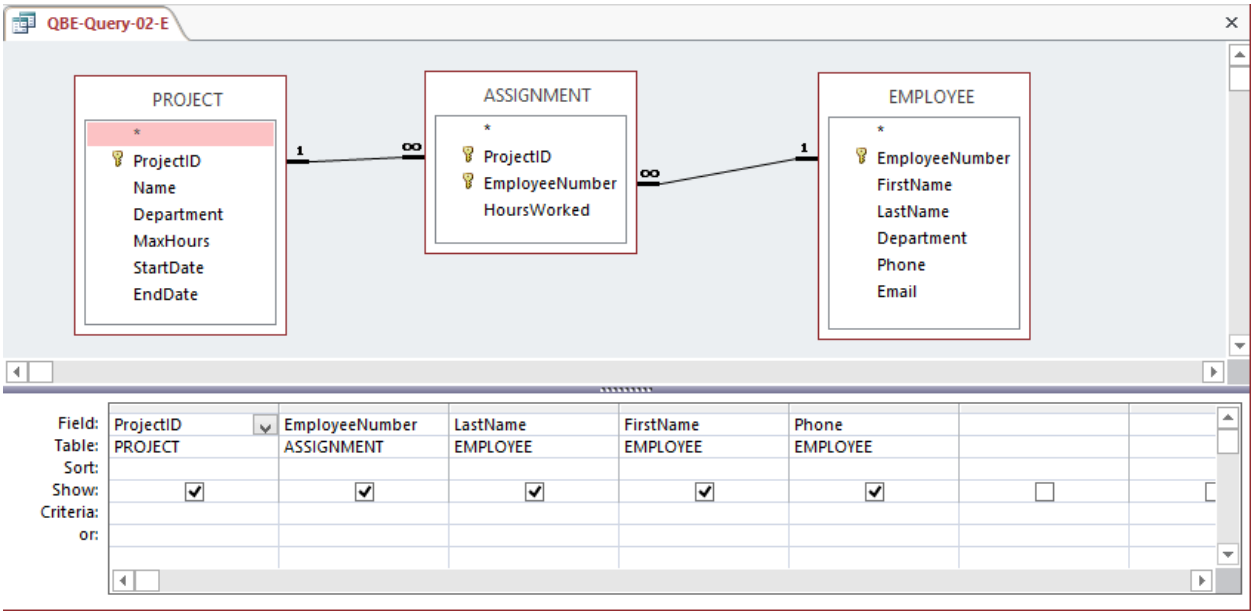
C. *What projects in the PROJECT table started before August 1, 2013? Show all the information for each project.*



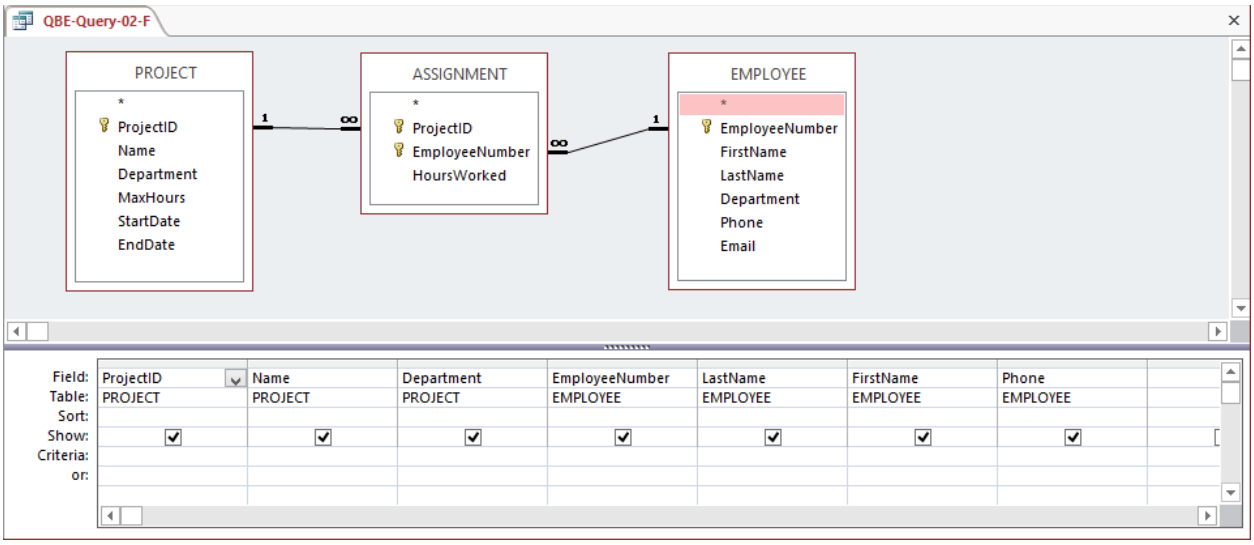
D. *What projects in the PROJECT table have not been completed? Show all the information for each project.*



E. Who are the employees assigned to each project? Show ProjectID, Employee-Number, LastName, FirstName, and Phone.



F. Who are the employees assigned to each project? Show the ProjectID, Name, and Department. Show EmployeeNumber, LastName, FirstName, and Phone.



G. Who are the employees assigned to each project? Show ProjectID, Name, Department, and Department Phone. Show EmployeeNumber, LastName, FirstName, and Employee Phone. Sort by ProjectID in ascending order.

This question is more complicated than it seems. It also raises the important question of why students need to know SQL, and provides one answer: QBE equivalents may not always work, or at least they don't work as intended. You should use this question as the basis for a discussion of this issue.

We have already run this query as an SQL query, and gotten the correct results. That SQL Query (from RQ 2.61-G) is

```
/*****   Question G - SQL-Query-02-G   ***********************/

SELECT     P.ProjectID, Name AS ProjectName,
           D.DepartmentName AS ProjectDepartment,
           D.Phone AS DepartmentPhone,
           E.EmployeeNumber, LastName, FirstName,
           E.Phone AS EmployeePhone
FROM       ((ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
              ON A.EmployeeNumber=E.EmployeeNumber)
           INNER JOIN PROJECT AS P
              ON A.ProjectID=P.ProjectID)
           INNER JOIN DEPARTMENT AS D
              ON P.Department=D.DepartmentName
ORDER BY   P.ProjectID;
```

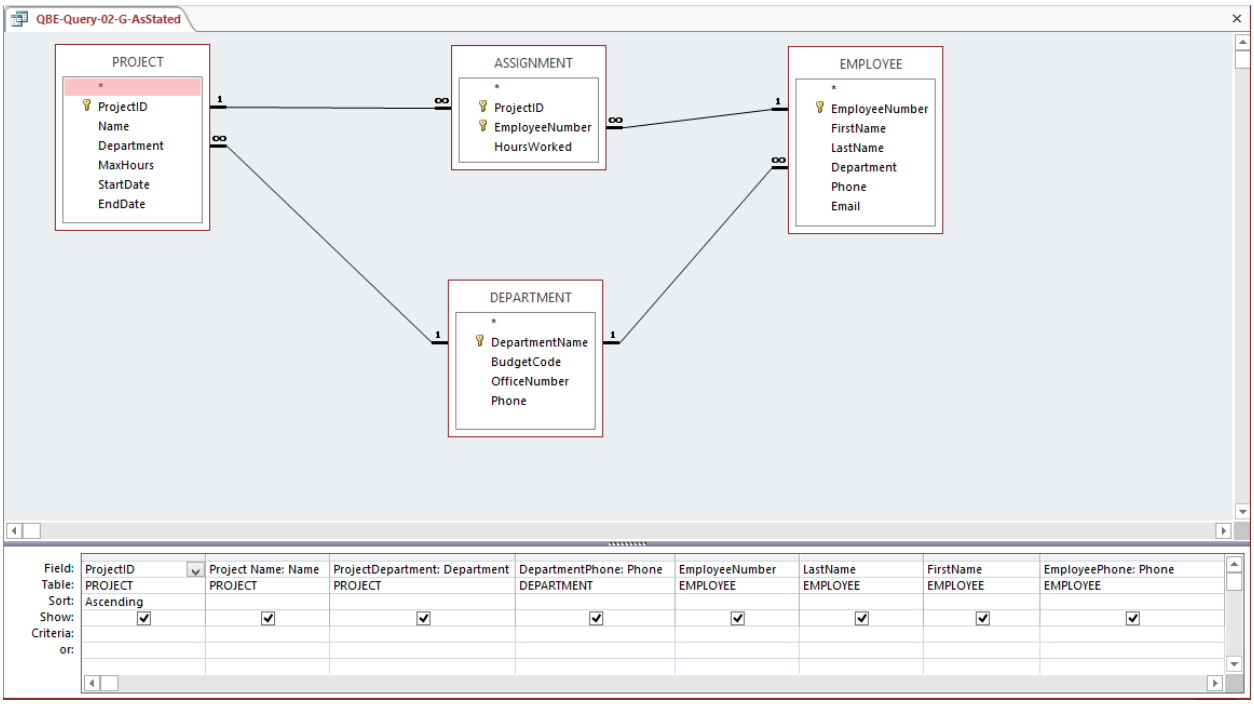
The results, which are correct, of this query are:

SQL-Query-02-G

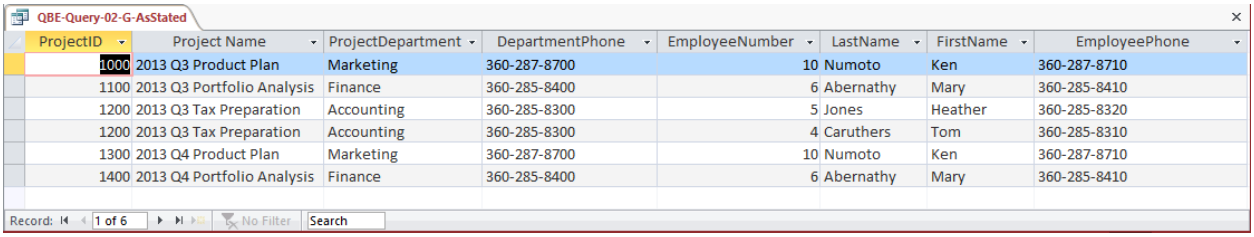
ProjectID	ProjectName	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1000	2013 Q3 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	5	Jones	Heather	360-285-8320
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	2	Jackson	Rosalie	360-285-8120
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1300	2013 Q4 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	5	Jones	Heather	360-285-8320
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310

Record: 1 of 15 No Filter Search

If we build the obvious corresponding QBE query we get (note the use of the aliases **ProjectName**, **ProjectDepartment**, **DepartmentPhone** and **EmployeePhone**):



This QBE query shows the solution to the question as stated, but it will not run correctly due to how Microsoft Access interprets the JOIN...ON commands in the QBE query it itself created! The QBE query results are:



ProjectID	Project Name	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	5	Jones	Heather	360-285-8320
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	4	Caruthers	Tom	360-285-8310
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410

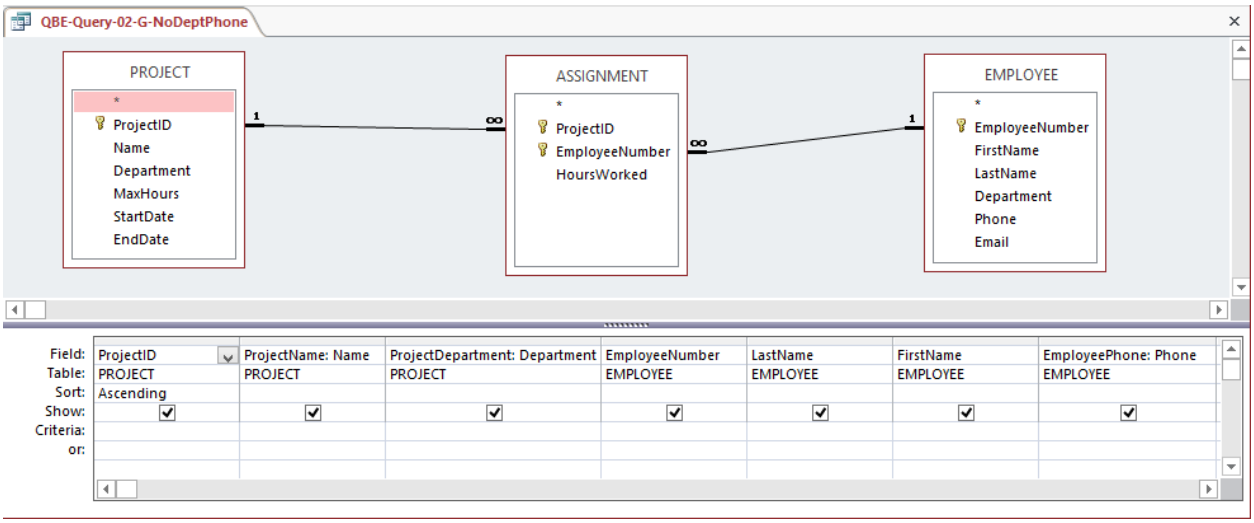
Compare these results with those shown for SQL-Query-2-G above, and you will see the difference and these results are clearly wrong. Looking at the data itself and thinking about what the query results *should* be also makes it obvious that there is a problem here.

For reference, here is the SQL code that Microsoft Access created from the QBE query:

```
SELECT     PROJECT.ProjectID, PROJECT.Name AS [Project Name],
           PROJECT.Department, DEPARTMENT.Phone AS DepartmentPhone,
           EMPLOYEE.EmployeeNumber, EMPLOYEE.LastName, EMPLOYEE.FirstName,
           EMPLOYEE.Phone AS EmployeePhone
FROM       ((DEPARTMENT INNER JOIN PROJECT ON
                  DEPARTMENT.DepartmentName = PROJECT.Department)
                  INNER JOIN EMPLOYEE ON
                  DEPARTMENT.DepartmentName = EMPLOYEE.Department)
                  INNER JOIN ASSIGNMENT ON
                      (PROJECT.ProjectID = ASSIGNMENT.ProjectID)
                       AND
                      (EMPLOYEE.EmployeeNumber = ASSIGNMENT.EmployeeNumber)
ORDER BY   PROJECT.ProjectID;
```

What can we do? There are two work arounds.

First, create the query *without* Department Phone. This is the only column needed from the DEPARTMENT table, which can thus be eliminated from the query. The QBE query is (note the use of the aliases **ProjectName**, **ProjectDepartment** and **EmployeePhone**):



The results will be correct, but without the DepartmentPhone column. The results are:

QBE-Query-02-G-NoDeptPhone

ProjectID	ProjectName	ProjectDepartment	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	10	Numoto	Ken	360-287-8710
1000	2013 Q3 Product Plan	Marketing	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	1	Jacobs	Mary	360-285-8110
1100	2013 Q3 Portfolio Analysis	Finance	6	Abernathy	Mary	360-285-8410
1100	2013 Q3 Portfolio Analysis	Finance	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	5	Jones	Heather	360-285-8320
1200	2013 Q3 Tax Preparation	Accounting	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	2	Jackson	Rosalie	360-285-8120
1300	2013 Q4 Product Plan	Marketing	10	Numoto	Ken	360-287-8710
1300	2013 Q4 Product Plan	Marketing	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	1	Jacobs	Mary	360-285-8110
1400	2013 Q4 Portfolio Analysis	Finance	6	Abernathy	Mary	360-285-8410
1400	2013 Q4 Portfolio Analysis	Finance	5	Jones	Heather	360-285-8320
1400	2013 Q4 Portfolio Analysis	Finance	4	Caruthers	Tom	360-285-8310
*			(New)			

Record: 1 of 15 No Filter Search

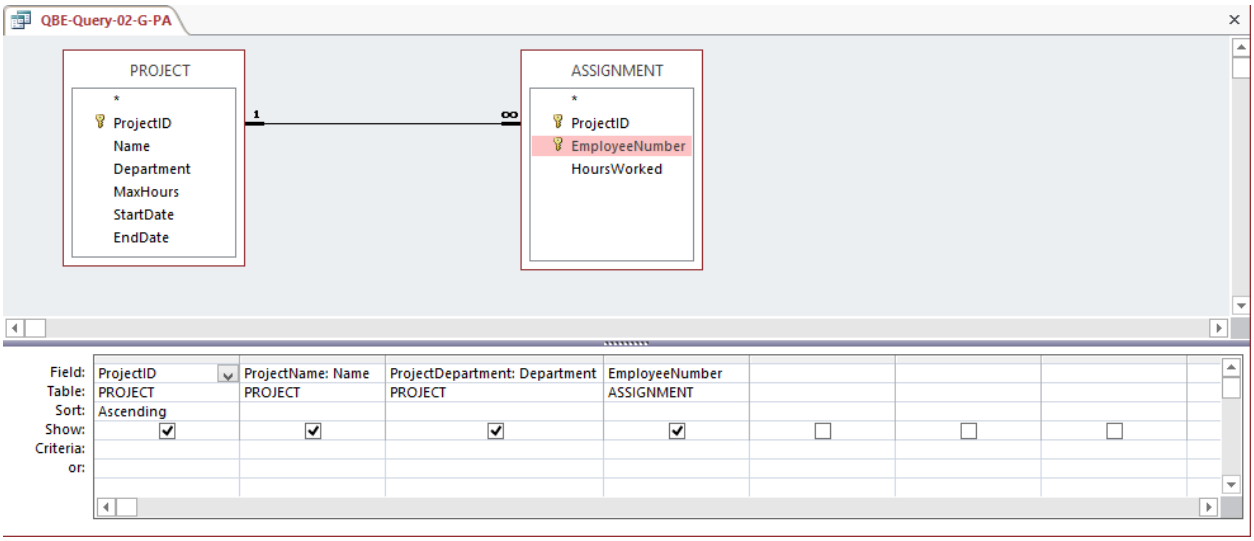
Alternatively, as devised by Professor John Schauf of Edgewood College, Madison, WI, you can illustrate building a set of queries, where each one uses the previous query and adds one additional table. This is possible because Microsoft Access allows saved queries to be used as the equivalent of a table in a query. By adding in one table at a time, you can control the JOIN...ON statement sequence, and obtain the correct answer.

This is a much better solution, because the end result is exactly what we want, rather than a truncated version of it.

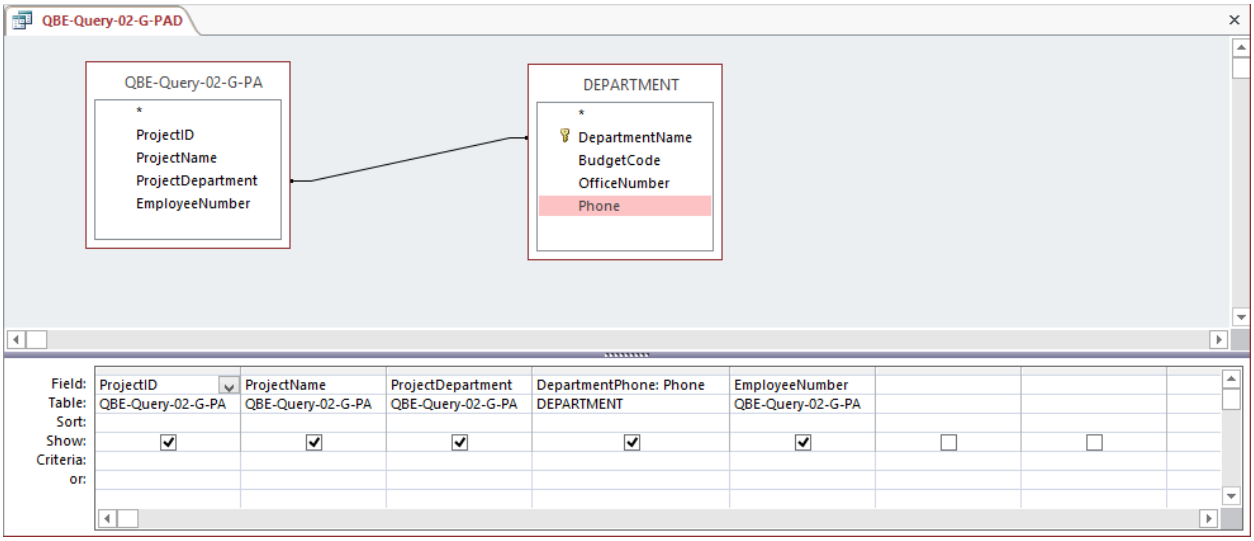
You should use this solution in class to illustrate how to use Microsoft Access query objects as pseudo tables in queries, and point out that they can also be used in forms and reports.

The steps below show how to create the needed sequence of QBE queries:

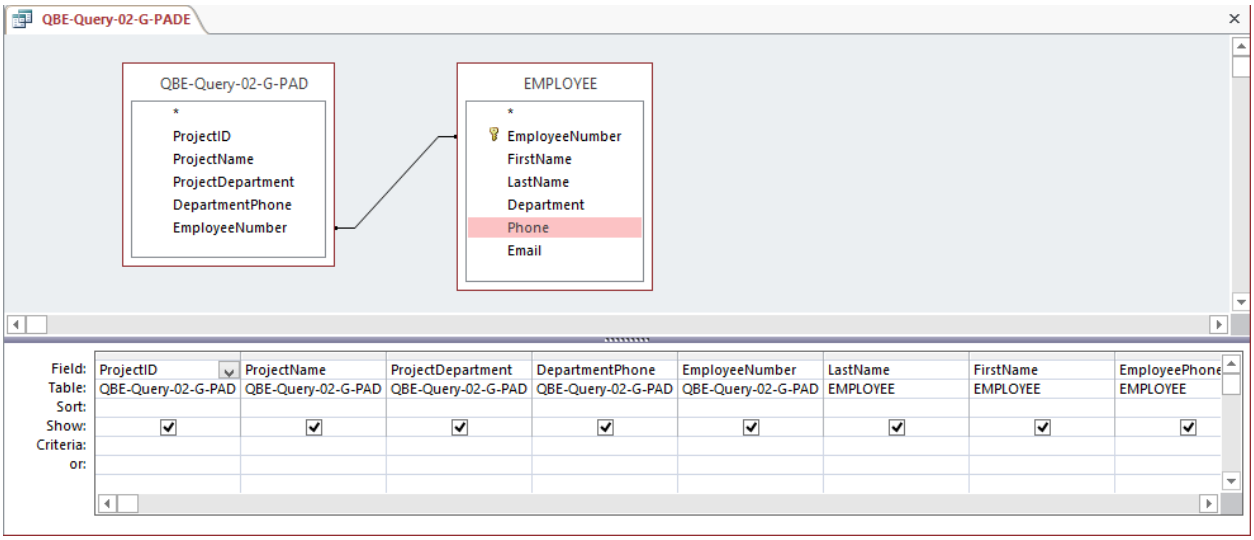
(1) Create a query that joins PROJECT and ASSIGNMENT, and name it QBE-Query-02-G-PA. Note that you must include ASSIGNMENT.EmployeeNumber in this query. Also note the use of the two aliases **ProjectName** and **ProjectDepartment**:



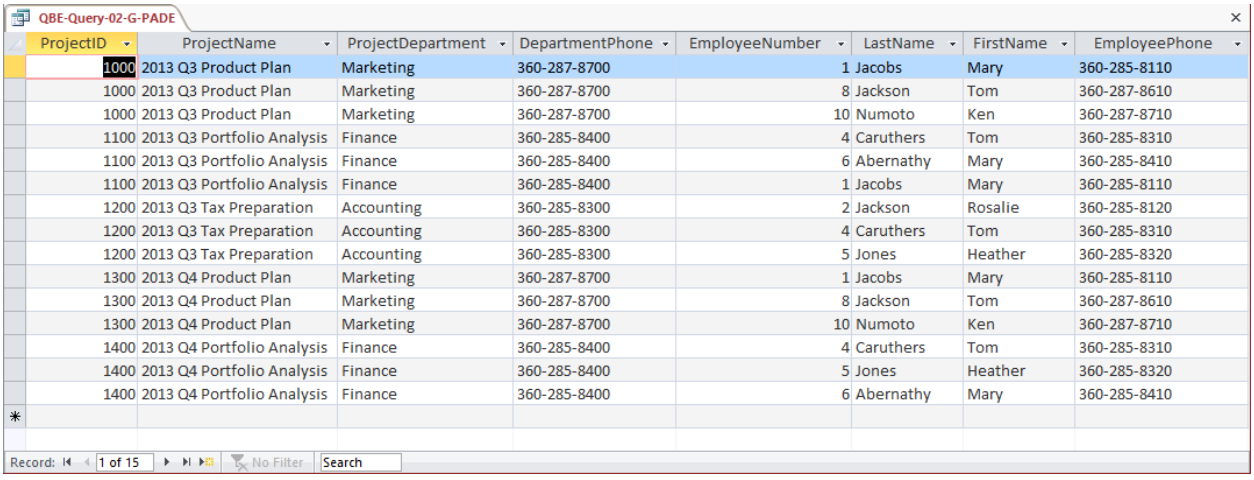
(2) Create a query that joins QBE-Query-02-G-PA and DEPARTMENT, and name it QBE-Query-02-G-PAD. Note that you will have to **manually link** the DEPARTMENT primary key to the foreign key in QBE-Query-02-G-PA. Also note the use of the alias **DepartmentPhone**:



(3) Create a query that joins QBE-Query-02-G-PAD and EMPLOYEE, and name it QBE-Query-02-G-PADE. Note that you will have to **manually link** the DEPARTMENT primary key to the foreign key in QBE-Query-02-G-PAD. Also note the use of the alias **EmployeePhone**:



The query results are now correct:



ProjectID	ProjectName	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1000	2013 Q3 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410
1100	2013 Q3 Portfolio Analysis	Finance	360-285-8400	1	Jacobs	Mary	360-285-8110
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	2	Jackson	Rosalie	360-285-8120
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	4	Caruthers	Tom	360-285-8310
1200	2013 Q3 Tax Preparation	Accounting	360-285-8300	5	Jones	Heather	360-285-8320
1300	2013 Q4 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1300	2013 Q4 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	4	Caruthers	Tom	360-285-8310
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	5	Jones	Heather	360-285-8320
1400	2013 Q4 Portfolio Analysis	Finance	360-285-8400	6	Abernathy	Mary	360-285-8410

H. Who are the employees assigned to projects run by the marketing department? Show ProjectID, Name, Department, and Department Phone. Show EmployeeNumber, LastName, FirstName, and Employee Phone. Sort by ProjectID in ascending order.

This question is identical to question G except for the restriction to marketing department projects. And, again, this question is more complicated than it seems. It also raises the important question of why students need to know SQL, and provides one answer: QBE equivalents may not always work, or at least they don't work as intended. You should use this question as the basis for a discussion of this issue.

We have already run this query as an SQL query, and gotten the correct results. That SQL Query (from RQ 2.61-H) is

```
/*****   Question H - SQL-Query-02-H   **********************/

SELECT     P.ProjectID, Name AS ProjectName,
           D.DepartmentName AS ProjectDepartment,
           D.Phone AS DepartmentPhone,
           E.EmployeeNumber, LastName, FirstName,
           E.Phone AS EmployeePhone
FROM       ((ASSIGNMENT AS A INNER JOIN EMPLOYEE AS E
              ON A.EmployeeNumber=E.EmployeeNumber)
           INNER JOIN PROJECT AS P
              ON A.ProjectID=P.ProjectID)
           INNER JOIN DEPARTMENT AS D
              ON P.Department=D.DepartmentName
WHERE      DepartmentName='Marketing'
ORDER BY   P.ProjectID;
```

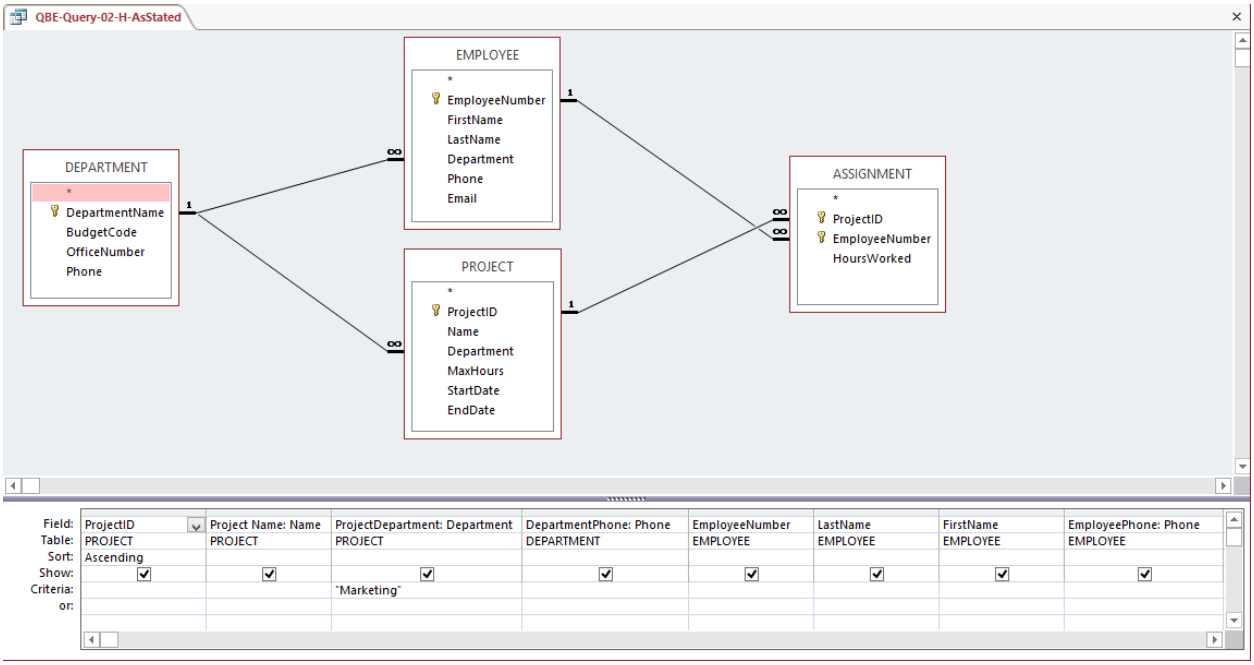
The results, which are correct, of this query are:

SQL-Query-02-H

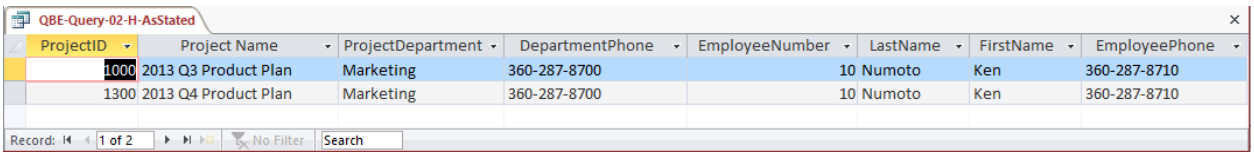
ProjectID	ProjectName	ProjectDepartment	DepartmentPhone	EmployeeNumber	LastName	FirstName	EmployeePhone
1000	2013 Q3 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1000	2013 Q3 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1000	2013 Q3 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110
1300	2013 Q4 Product Plan	Marketing	360-287-8700	10	Numoto	Ken	360-287-8710
1300	2013 Q4 Product Plan	Marketing	360-287-8700	8	Jackson	Tom	360-287-8610
1300	2013 Q4 Product Plan	Marketing	360-287-8700	1	Jacobs	Mary	360-285-8110

Record: 1 of 6 No Filter Search

If we build the obvious corresponding QBE query we get (note the use of the aliases **ProjectName**, **ProjectDepartment**, **DepartmentPhone** and **EmployeePhone**):



The results are:



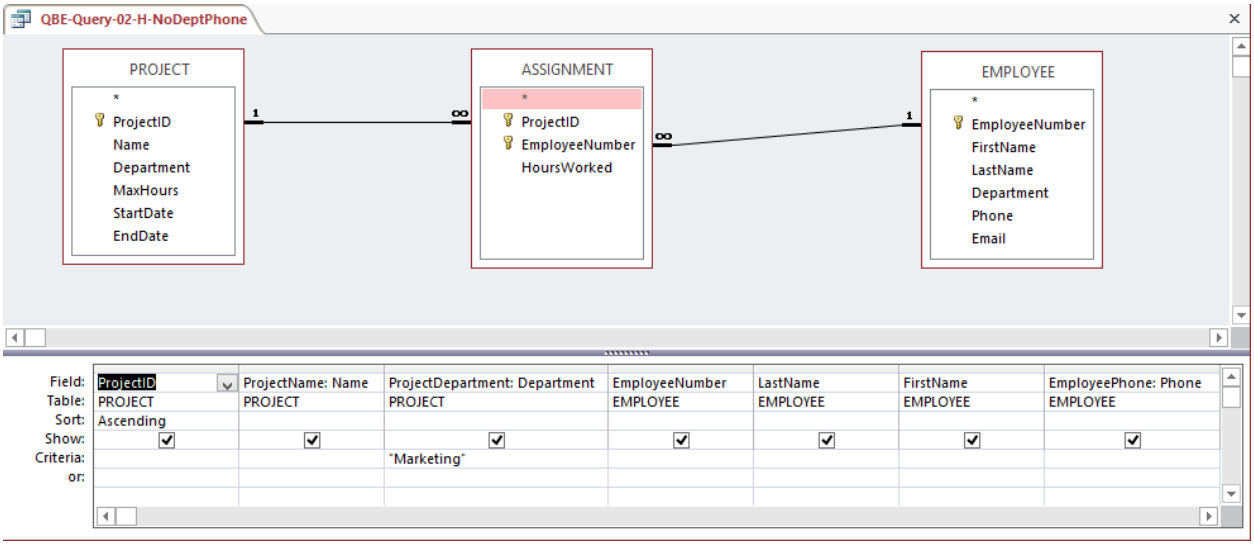
Compare these results with those shown for SQL-Query-02-H above, and you will see the difference.

For reference, here is the SQL code that Microsoft Access created from the QBE query:

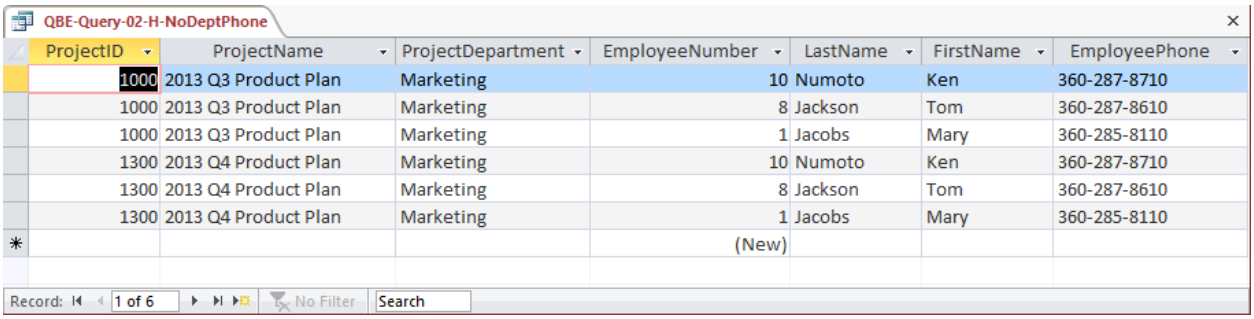
```
SELECT     PROJECT.ProjectID, PROJECT.Name AS [Project Name],
           PROJECT.Department AS ProjectDepartment,
           DEPARTMENT.Phone AS DepartmentPhone, EMPLOYEE.EmployeeNumber,
           EMPLOYEE.LastName, EMPLOYEE.FirstName,
           EMPLOYEE.Phone AS EmployeePhone
FROM       ((DEPARTMENT INNER JOIN PROJECT ON
              DEPARTMENT.DepartmentName = PROJECT.Department)
              INNER JOIN EMPLOYEE ON
                 DEPARTMENT.DepartmentName = EMPLOYEE.Department)
              INNER JOIN ASSIGNMENT ON
                 (PROJECT.ProjectID = ASSIGNMENT.ProjectID)
                  AND
                 (EMPLOYEE.EmployeeNumber = ASSIGNMENT.EmployeeNumber)
WHERE      (((PROJECT.Department)="Marketing"))
ORDER BY   PROJECT.ProjectID;
```

The problem we are encountering here is the same as described above in 2.64 G. Again, there are two work arounds. First, create the query *without* Department Phone. This is the only column needed from the DEPARTMENT table, which can thus be eliminated from the query.

The QBE Query is (note the use of the aliases **ProjectName**, **ProjectDepartment** and **EmployeePhone**):



The results will be correct, but without the DepartmentPhone column:



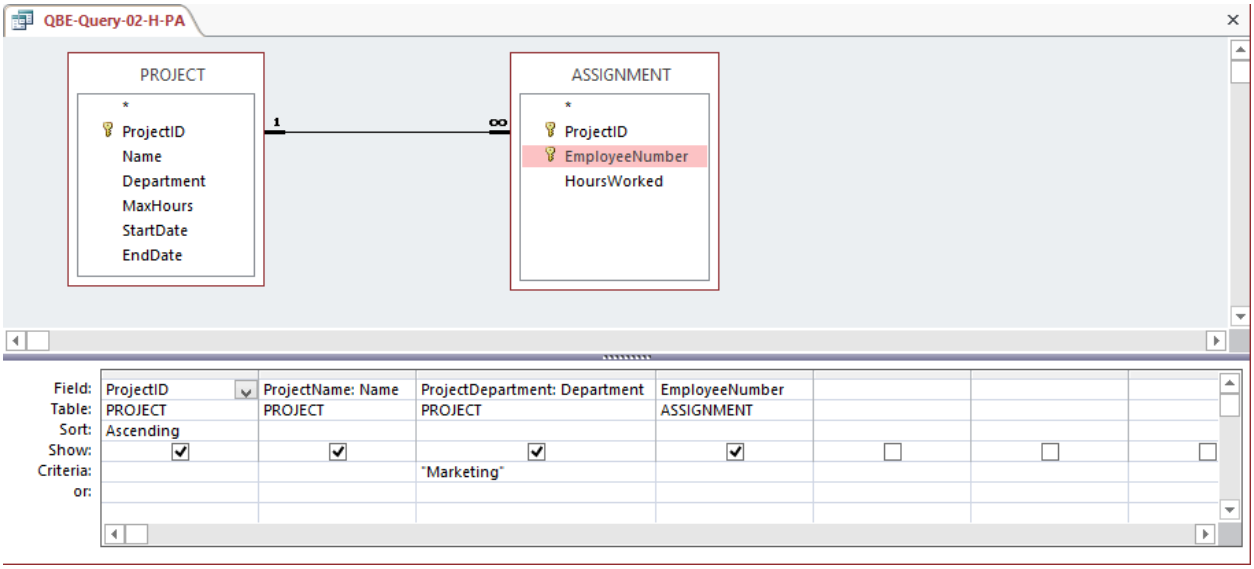
Alternatively, as devised by Professor John Schauf of Edgewood College, Madison, WI, you can illustrate building a set of queries, where each one uses the previous query and adds one additional table. This is possible because Microsoft Access allows saved queries to be used as the equivalent of a table in a query. By adding in one table at a time, you can control the JOIN...ON statement sequence, and obtain the correct answer.

This is a much better solution, because the end result is exactly what we want, rather than a truncated version of it.

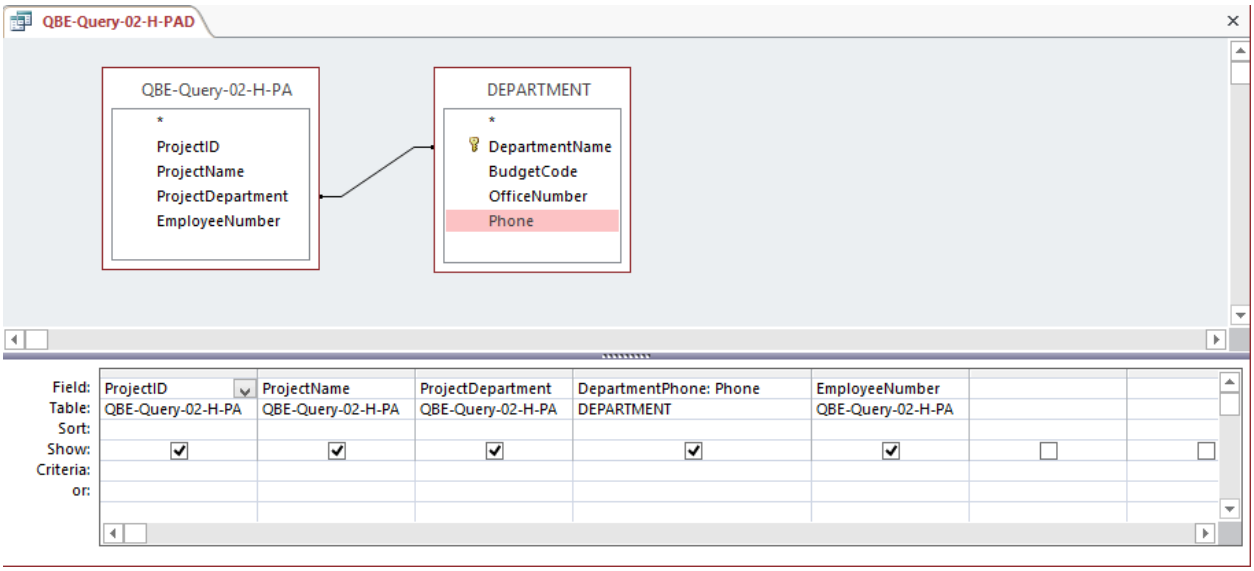
You should use this solution in class to illustrate how to use Microsoft Access query objects as pseudo tables in queries, and point out that they can also be used in forms and reports.

The steps below show how to create the needed sequence of QBE queries:

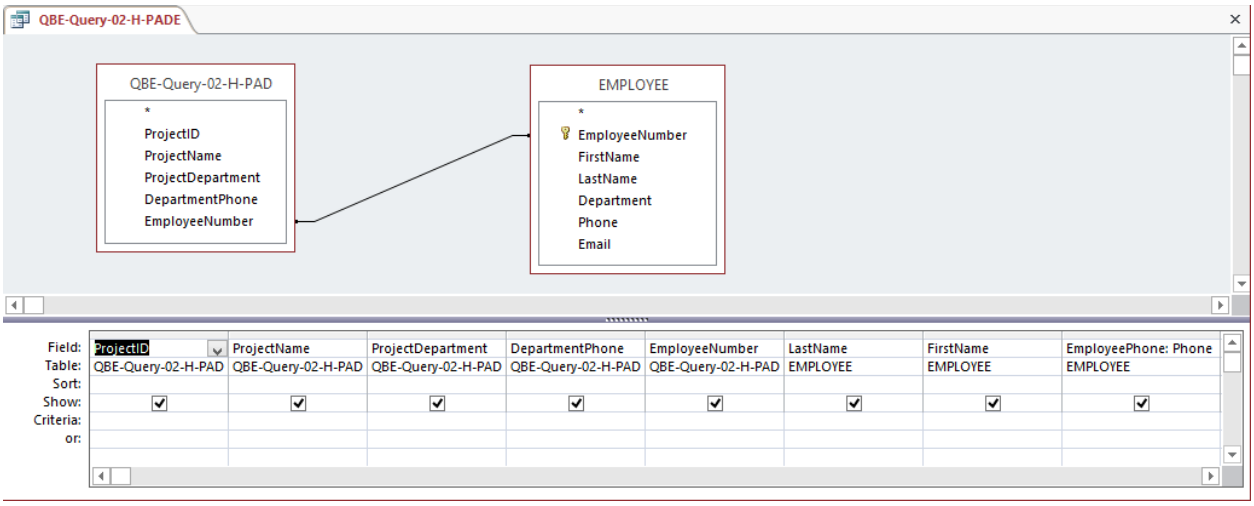
(1) Create a query that joins PROJECT and ASSIGNMENT, and name it QBE-Query-0H-G-PA. Note that you must include ASSIGNMENT.EmployeeNumber in this query, and note the use of the aliases **ProjectName** and **ProjectDepartment**:



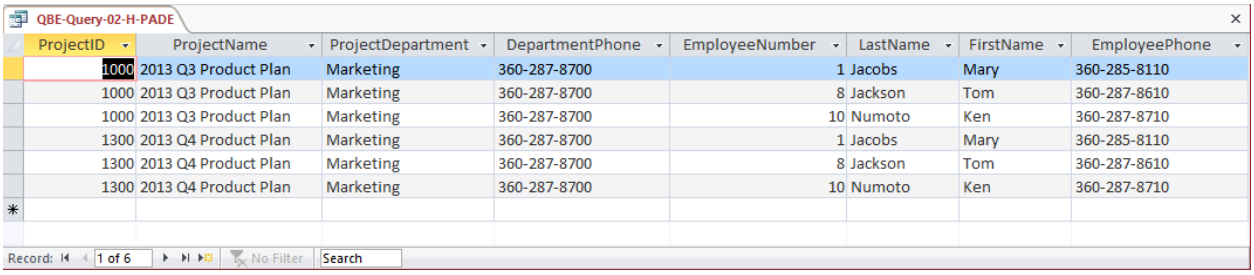
(2) Create a query that joins QBE-Query-02-H-PA and DEPARTMENT, and name it QBE-Query-02-H-PAD. Note that you will have to **manually link** the DEPARTMENT primary key to the foreign key in QBE-Query-02-H-PA, and note the use of the alias **DepartmentPhone**:



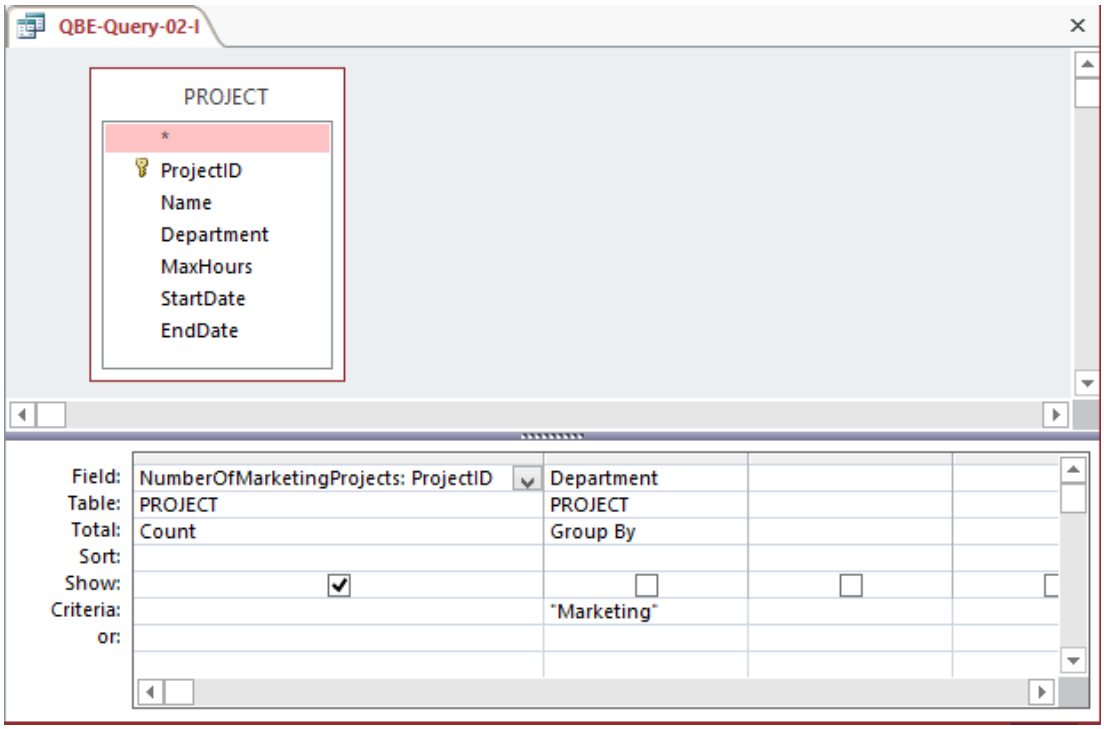
(3) Create a query that joins QBE-Query-02-H-PAD and EMPLOYEE, and name it QBE-Query-02-H-PADE. Note that you will have to **manually link** the DEPARTMENT primary key to the foreign key in QBE-Query-02-H-PAD, and note the use of the alias **EmployeePhone**:



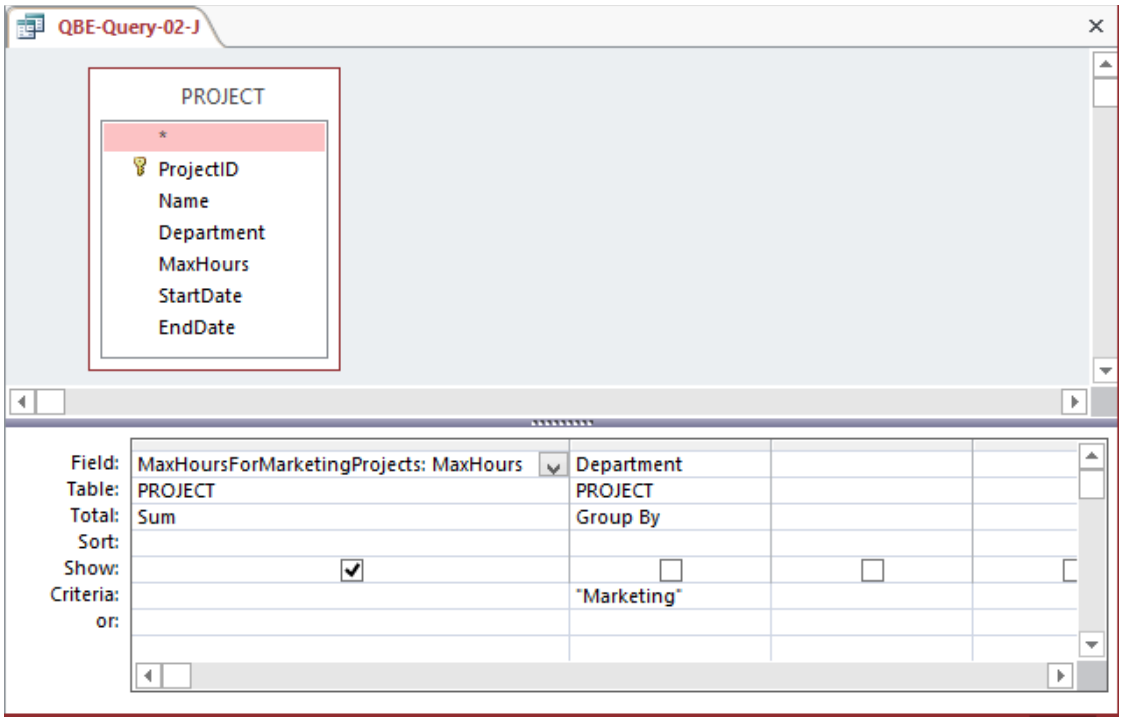
The query results are now correct:



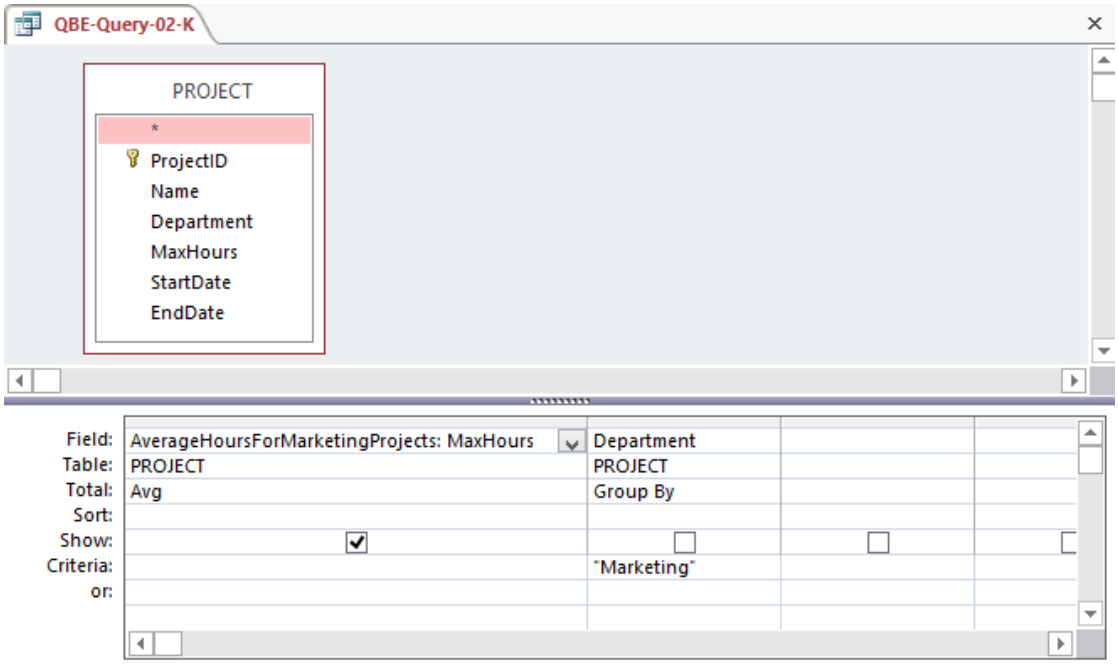
I. *How many projects are being run by the marketing department? Be sure to assign an appropriate column name to the computed results.*



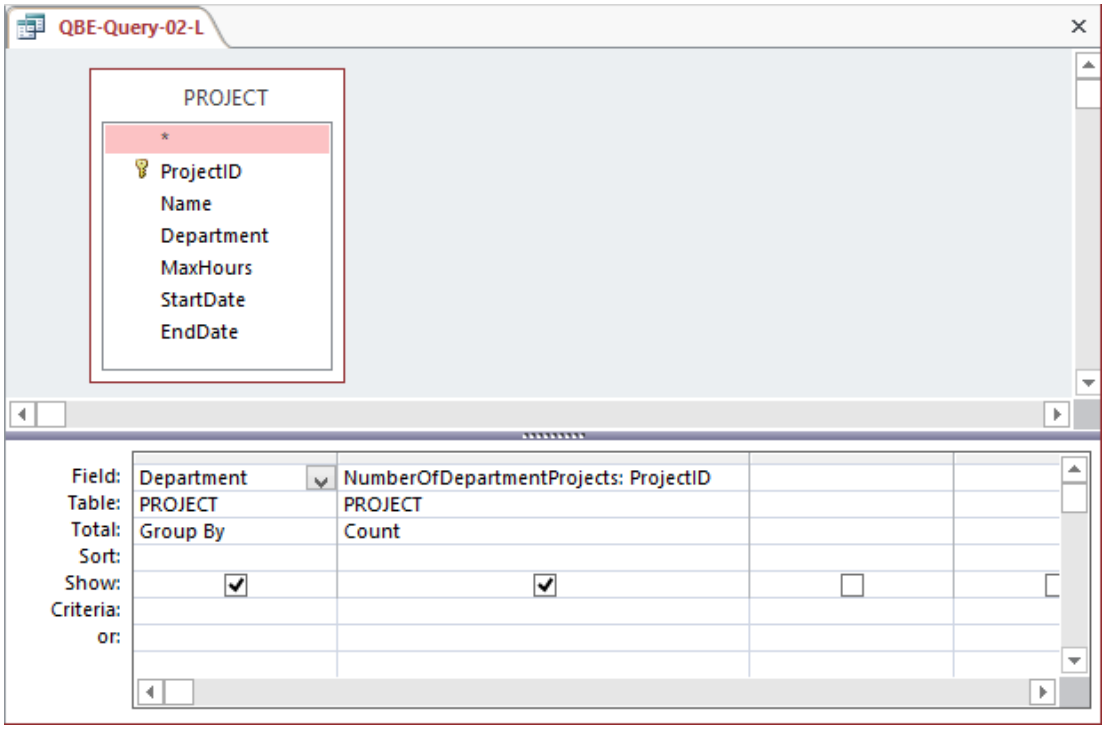
J. *What is the total MaxHours of projects being run by the marketing department? Be sure to assign an appropriate column name to the computed results.*



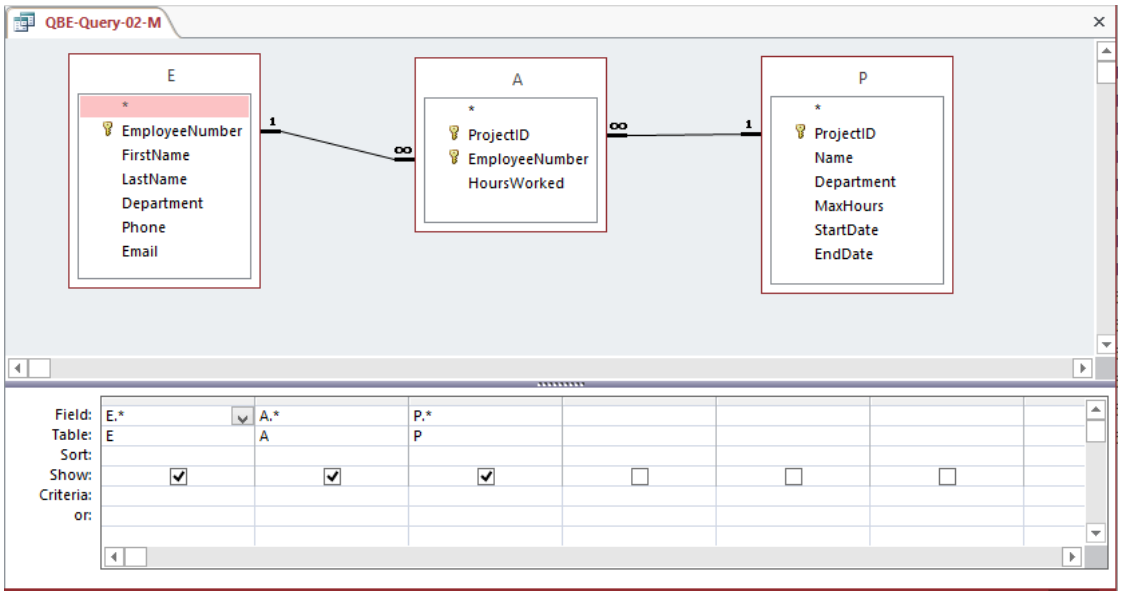
K. *What is the average MaxHours of projects being run by the marketing department? Be sure to assign an appropriate column name to the computed results.*



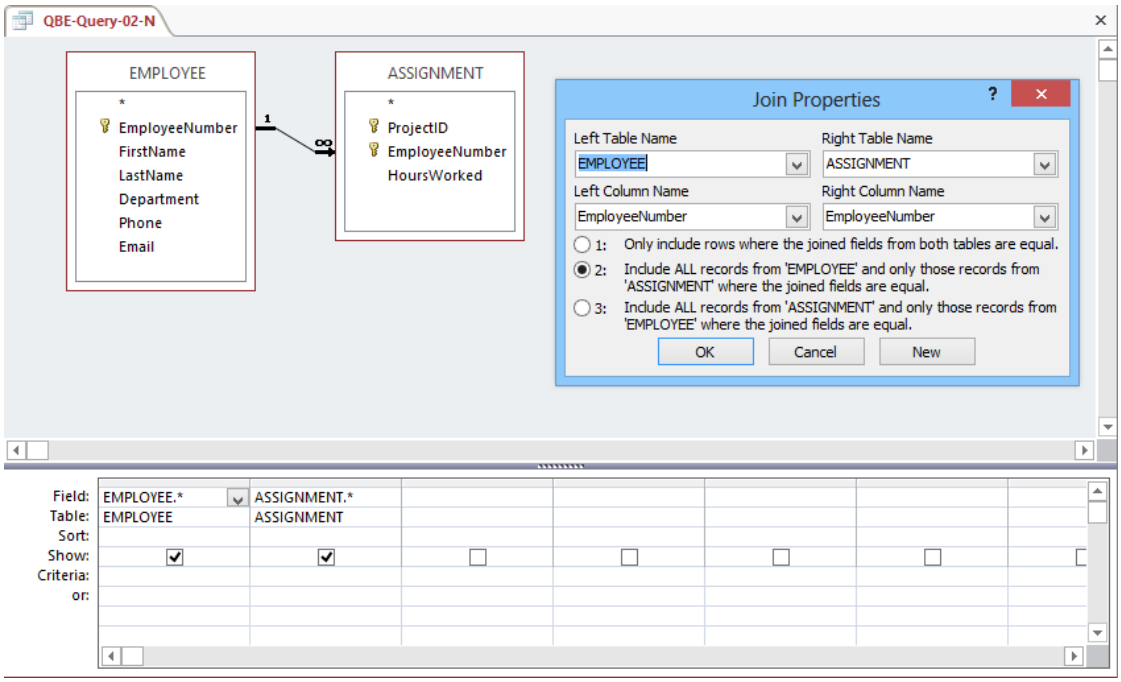
L. *How many projects are being run by each department? Be sure to display each DepartmentName and to assign an appropriate column name to the computed results.*



M. *Write an SQL statement to join EMPLOYEE, ASSIGNMENT, and PROJECT using the JOIN ON syntax. Run this statement.*



N. *Write an SQL statement to join EMPLOYEE and ASSIGNMENT and include all rows of EMPLOYEE in your answer, regardless of whether they have an ASSIGNMENT. Run this statement.*



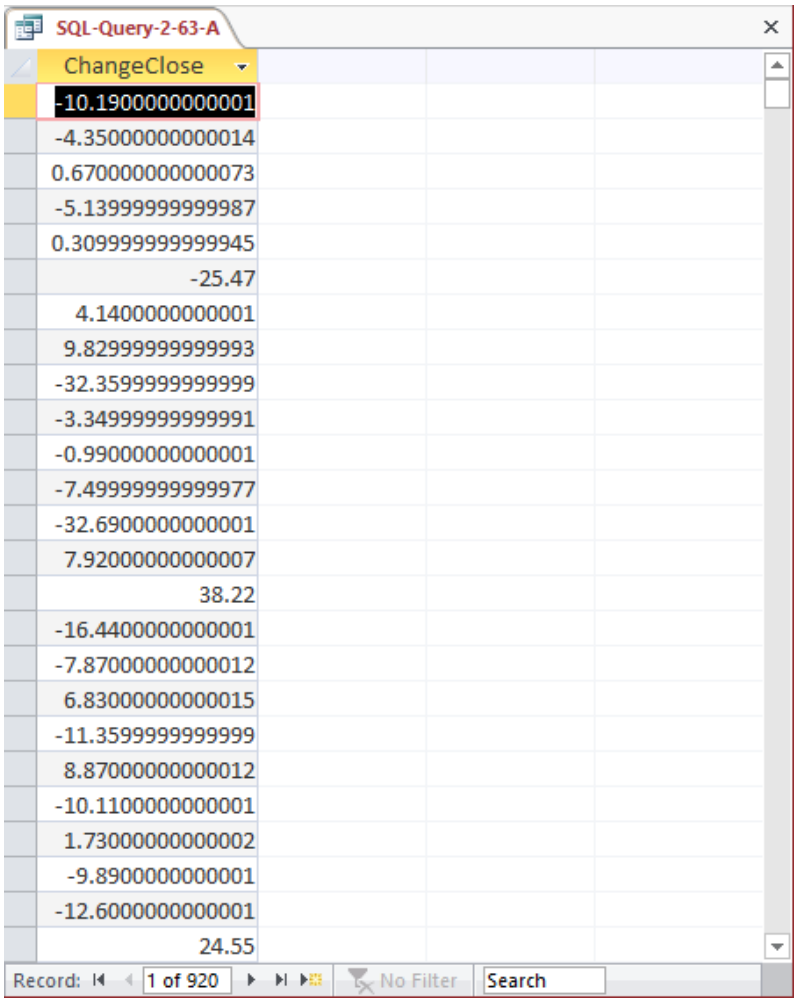
The following questions refer to the NDX table of data as described starting on page 72. You can obtain a copy of this data in the Access database, DBPe11-NDX.accdb located on this text's Web site at www.pearsonhighered.com/kroenke.

2.66 Write SQL queries to produce the following results:

A. The ChangeClose on Fridays.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP e13-IM-CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-2-63-A *** */

SELECT   ChangeClose
FROM     NDX
WHERE    TDayOfWeeK = 'Friday';
```



B. The minimum, maximum, and average ChangeClose on Fridays.

Solutions to Project Questions 2.63.A – 2.63.H are contained in the Microsoft Access database *DBP-e13-IM-CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-2-63-B *** */

SELECT    MIN (ChangeClose) AS MinFridayChangeClose,
          MAX (ChangeClose) AS MaxFridayChangeClose,
          AVG (ChangeClose) AS AverageFridayChangeClose
FROM      NDX
WHERE     TDayOfWeeK = 'Friday';
```

SQL-Query-2-63-B

MinFridayChangeClose	MaxFridayChangeClose	AverageFridayChangeClose
-345.85	273.32	0.146021739130452

Record: 1 of 1 No Filter Search

C. The average ChangeClose grouped by TYear. Show TYear.

Since TYear is being displayed, it makes sense to sort the results by TYear although this is not explicitly stated in the question.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM-CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-2-63-C *** */

SELECT      TYear, AVG (ChangeClose) AS AverageChangeClose
FROM        NDX
GROUP BY    TYear
ORDER BY    TYear;
```

SQL-Query-2-63-C

TYear	AverageChangeClose
1985	0.639841269841275
1986	0.0720158102766874
1987	0.117351778656135
1988	0.167272727272733
1989	0.368452380952389
1990	-0.184229249011848
1991	1.03023715415022
1992	0.230944881889775
1993	0.301146245059303
1994	-1.55670634920634
1995	0.682380952380964
1996	0.965078740157492
1997	0.669841897233221
1998	3.35388888888891
1999	7.42785714285718
2000	-5.42115079365074
2001	-3.08326612903223
2002	-2.37071999999998
2003	1.91884920634923
2004	8.75666666666666

Record: 1 of 20 No Filter Search

D. The average ChangeClose grouped by TYear and TMonth. Show TYear and TMonth.

Since TYear and TMonth are being displayed, it makes sense to sort the results by TYear and TMonth although this is not explicitly stated in the question.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM-CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-2-63-D-A *** */

SELECT   TYear, TMonth,
         AVG (ChangeClose) AS AverageChangeClose
FROM     NDX
GROUP BY TYear, TMonth
ORDER BY TYear, TMonth;
```

SQL-Query-2-63-D-A

TYear	TMonth	AverageChangeClose
1985	December	0.593809523809532
1985	November	1.058
1985	October	0.303636363636368
1986	April	0.550000000000009
1986	August	0.666190476190487
1986	December	-0.594090909090896
1986	February	0.789473684210538
1986	January	0.057272727272732
1986	July	-1.62818181818181
1986	June	-0.0519047619047553
1986	March	0.843500000000003
1986	May	0.785714285714291
1986	November	0.364210526315796
1986	October	0.60739130434783
1986	September	-1.35285714285714
1987	April	-0.115238095238088
1987	August	1.25952380952383
1987	December	1.73863636363637
1987	February	1.6921052631579
1987	January	2.40666666666668
1987	July	0.646363636363638

Record: 1 of 220 No Filter Search

Unfortunately, the table NDX does not contain a numeric value of the month, so in order to sort the months correctly, we need a TMonthNumber which has a column containing a representative number for each month (January = 1, February = 2, etc.)

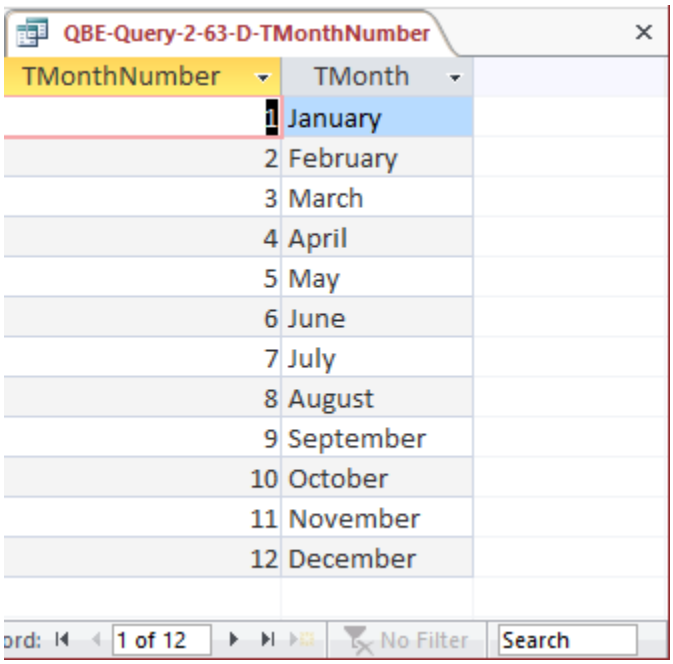
Although the SQL DDL and DML for doing this is not covered until Chapter 7, this is a good exercise in adding a column to an existing table, and you may want to show this to your students at this time.

We can create this column as follows (note that Microsoft Access can only run one SQL command at a time!):

```
/* *** SQL-ALTER-TABLE-2-63-D *** */

ALTER TABLE NDX
      ADD COLUMN TMonthNumber Int NULL;

/* *** SQL-UPDATES-2-63-D *** */

UPDATE NDX
      SET      TMonthNumber = 1
      WHERE    TMonth = 'January';

UPDATE NDX
      SET      TMonthNumber = 2
      WHERE    TMonth = 'February';

UPDATE NDX
      SET      TMonthNumber = 3
      WHERE    TMonth = 'March';

UPDATE NDX
      SET      TMonthNumber = 4
      WHERE    TMonth = 'April';

UPDATE NDX
      SET      TMonthNumber = 5
      WHERE    TMonth = 'May';

UPDATE NDX
      SET      TMonthNumber = 6
      WHERE    TMonth = 'June';

UPDATE NDX
      SET      TMonthNumber = 7
      WHERE    TMonth = 'July';

UPDATE NDX
      SET      TMonthNumber = 8
      WHERE    TMonth = 'August';

UPDATE NDX
      SET      TMonthNumber = 9
      WHERE    TMonth = 'September';
```

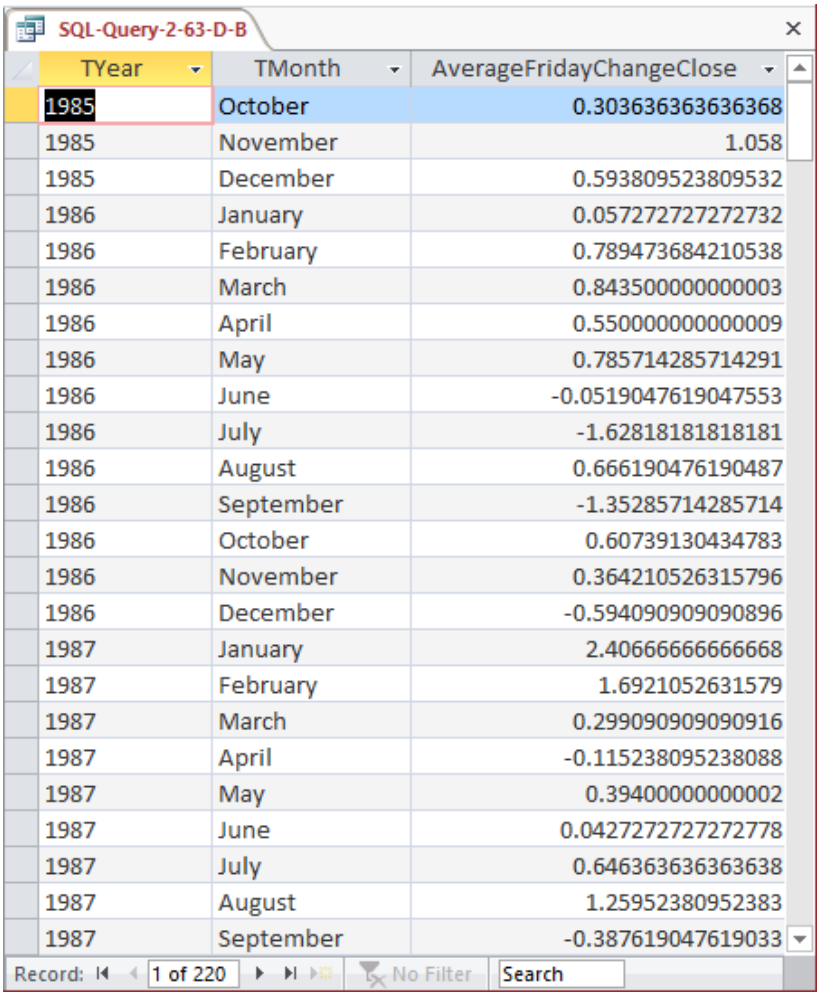
```
UPDATE NDX
      SET       TMonthNumber = 10
      WHERE     TMonth = 'October';

UPDATE NDX
      SET       TMonthNumber = 11
      WHERE     TMonth = 'November';

UPDATE NDX
      SET       TMonthNumber = 12
      WHERE     TMonth = 'December';

=====================================================================
```

An SQL or QBE Query can be used to show the data in the table (use GROUP BY):



Now that the NDX table includes this column, we can use it as follows to sort the data correctly:

```
/* *** SQL-Query-2-63-D-B *** */

SELECT    TYear, TMonth,
          AVG (ChangeClose) AS AverageFridayChangeClose
FROM      NDX
GROUP BY TYear, TMonth, TMonthNumber
ORDER BY TYear, TMonthNumber;
```



SQL-Query-2-63-D-B

TYear	TMonth	AverageFridayChangeClose
1985	October	0.303636363636368
1985	November	1.058
1985	December	0.593809523809532
1986	January	0.057272727272732
1986	February	0.789473684210538
1986	March	0.843500000000003
1986	April	0.550000000000009
1986	May	0.785714285714291
1986	June	-0.0519047619047553
1986	July	-1.62818181818181
1986	August	0.666190476190487
1986	September	-1.35285714285714
1986	October	0.60739130434783
1986	November	0.364210526315796
1986	December	-0.594090909090896
1987	January	2.40666666666668
1987	February	1.6921052631579
1987	March	0.299090909090916
1987	April	-0.115238095238088
1987	May	0.39400000000002
1987	June	0.0427272727272778
1987	July	0.646363636363638
1987	August	1.25952380952383
1987	September	-0.387619047619033

Record: 1 of 220 No Filter Search

E. The average ChangeClose grouped by TYear, TQuarter, TMonth shown in descending order of the average (you will have to give a name to the average in order to sort by it). Show TYear, TQuarter, and TMonth. Note that months appear in alphabetical and not calendar order. Explain what you need to do to obtain months in calendar order.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM—CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

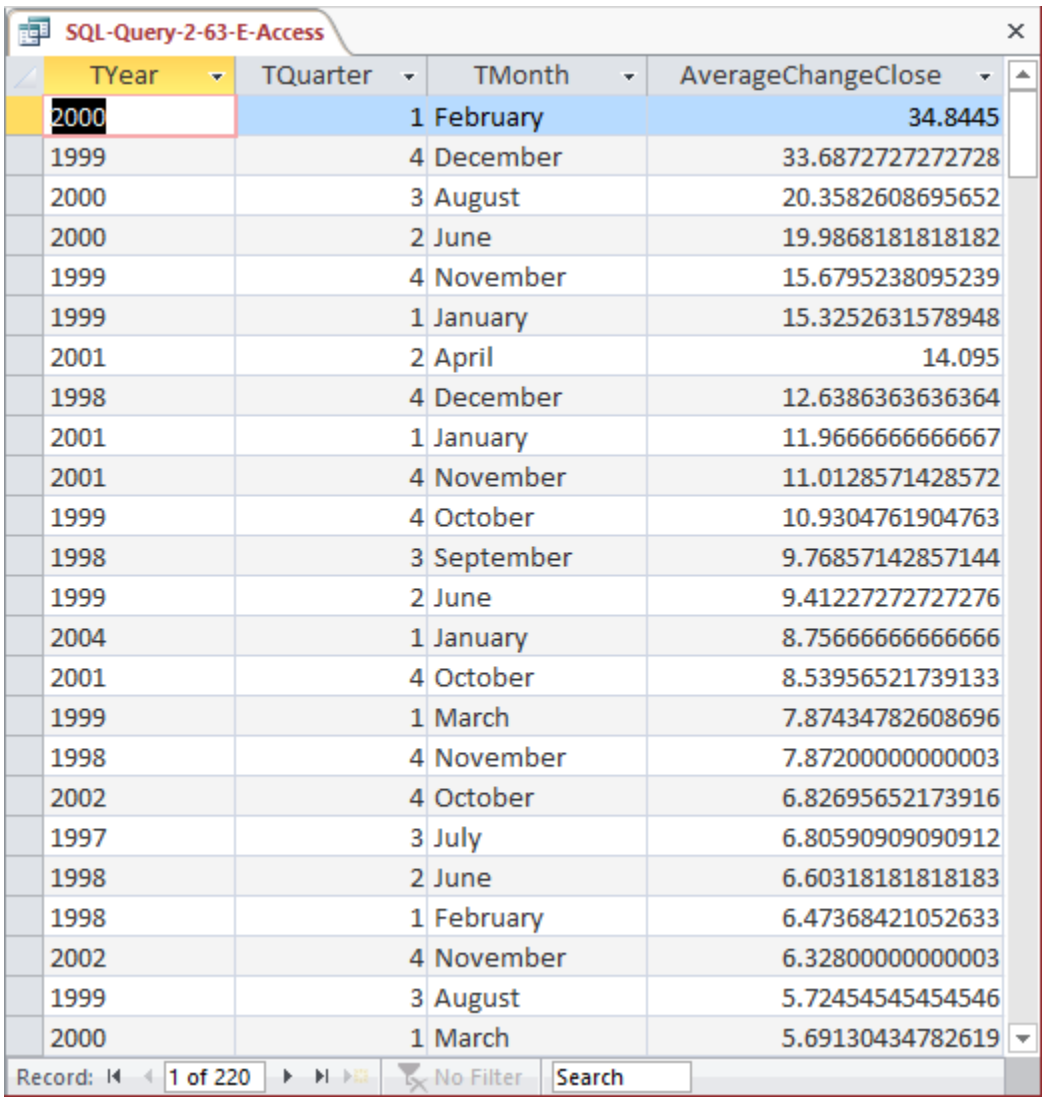
```
/* *** SQL-Query-2-63-E *** */

SELECT   TYear, TQuarter, TMonth,
         AVG(ChangeClose) AS AverageChangeClose
FROM     NDX
GROUP BY TYear, TQuarter, TMonth
ORDER BY AverageChangeClose DESC;
```

For Microsoft Access:

Unfortunately, as discussed above, Microsoft Access cannot process the ORDER BY clause correctly when an SQL built-in function is used in it. Therefore we rewrite the query as:

```
/* *** SQL-Query-2-63-E-Access *** */

SELECT   TYear, TQuarter, TMonth,
         AVG(ChangeClose) AS AverageChangeClose
FROM     NDX
GROUP BY TYear, TQuarter, TMonth
ORDER BY AVG(ChangeClose) DESC;
```

The result is:



SQL-Query-2-63-E-Access

TYear	TQuarter	TMonth	AverageChangeClose
2000	1	February	34.8445
1999	4	December	33.6872727272728
2000	3	August	20.3582608695652
2000	2	June	19.9868181818182
1999	4	November	15.6795238095239
1999	1	January	15.3252631578948
2001	2	April	14.095
1998	4	December	12.6386363636364
2001	1	January	11.9666666666667
2001	4	November	11.0128571428572
1999	4	October	10.9304761904763
1998	3	September	9.76857142857144
1999	2	June	9.41227272727276
2004	1	January	8.75666666666666
2001	4	October	8.53956521739133
1999	1	March	7.87434782608696
1998	4	November	7.87200000000003
2002	4	October	6.82695652173916
1997	3	July	6.80590909090912
1998	2	June	6.60318181818183
1998	1	February	6.47368421052633
2002	4	November	6.32800000000003
1999	3	August	5.72454545454546
2000	1	March	5.69130434782619

Record: 1 of 220 No Filter Search

In order to obtain the months in calendar order, we would have to use the TMonthNumber column we created in PQ 2.63-D with a numerical value for each month (1, 2, 3, …, 12) and sort by those values.

F. The difference between the maximum ChangeClose and the minimum ChangeClose grouped by TYear, TQuarter, TMonth shown in descending order of the difference (you will have to give a name to the difference in order to sort by it). Show TYear, TQuarter, and TMonth.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM—CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

```
/* *** SQL-Query-2-63-F *** */

SELECT   TYear, TQuarter, TMonth,
         (MAX(ChangeClose) - MIN(ChangeClose)) AS DifChangeClose
FROM     NDX
GROUP BY TYear, TQuarter, TMonth
ORDER BY DifChangeClose DESC;
```

For Microsoft Access:

Unfortunately, as discussed above, Microsoft Access cannot process the ORDER BY clause correctly when an SQL built-in function is used in it. Therefore we rewrite the query as:

```
/* *** SQL-Query-2-63-F-Access *** */

SELECT   TYear, TQuarter, TMonth,
         (MAX(ChangeClose) - MIN(ChangeClose)) AS DifChangeClose
FROM     NDX
GROUP BY TYear, TQuarter, TMonth
ORDER BY (MAX(ChangeClose) - MIN(ChangeClose)) DESC;
```

The query result is:

SQL-Query-2-63-F-Access

TYear	TQuarter	TMonth	DifChangeClose
2000	2	April	667.34
2001	1	January	612.52
2000	2	May	553.88
2000	4	October	518.97
2000	4	December	487.78
2000	1	January	433.14
2000	4	November	423.36
2000	1	March	423.13
1994	1	January	406.18
2000	2	June	402.58
2000	3	July	360.91
2000	1	February	360.59
2000	3	September	325.42
2001	2	April	280.96
2001	1	February	255.95
2001	1	March	242.47
2000	3	August	231.01
1999	3	September	224.96
2001	2	May	220.04
1999	4	December	213.6
1999	4	November	205.26
1999	1	February	199.38
1999	2	April	196.55
2001	3	July	190.98

Record: 1 of 220 No Filter Search

G. *The average ChangeClose grouped by TYear shown in descending order of the average (you will have to give a name to the average in order to sort by it). Show only groups for which the average is positive.*

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM—CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

For Microsoft SQL Server, Oracle Database and MySQL:

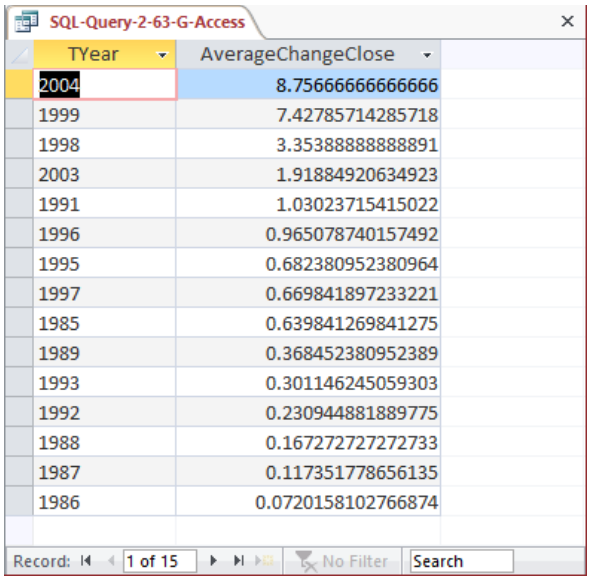
```
/* *** SQL-Query-2-63-G *** */

SELECT    TYear,
          AVG(ChangeClose) AS AverageChangeClose
FROM      NDX
GROUP BY  TYear
HAVING    AVG(ChangeClose) > 0
ORDER BY  AverageChangeClose DESC;
```

For Microsoft Access:

Unfortunately, as discussed above, Microsoft Access cannot process the ORDER BY clause correctly when an SQL built-in function is used in it. Therefore we rewrite the query as:

```
/* *** SQL-Query-2-63-G-Access *** */

SELECT    TYear,
          AVG(ChangeClose) AS AverageChangeClose
FROM      NDX
GROUP BY  TYear
HAVING    AVG(ChangeClose) > 0
ORDER BY  AVG(ChangeClose) DESC;
```

The result is:



H. Display a single field with the date in the form: day/month/year. Do not be concerned with trailing blanks.

Solutions to Project Questions 2.66.A – 2.66.H are contained in the Microsoft Access database *DBP-e13-IM—CH02-NDX.accdb* which is available on the text's Web site (www.pearsonhighered.com/kroenke).

The solution to this question requires the student to use the DBMS help function or other references to figure out a conversion function to convert the numerical day of the month to a character string that can be combined with other data already in character format. The original table NDX does not have a numeric value for month, so the names of the months will appear in the solution. If we want the numeric value of the month, we could use the modified NDX table, which has a numeric value TMonthNumber column. We would need to use the data type conversion on this field as well.

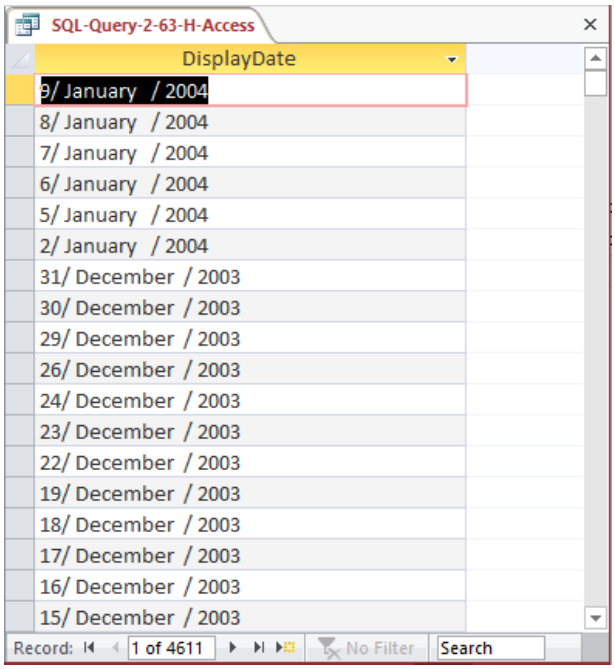
The SQL Statement using SQL Server 2012 character string functions is:

```
/* *** SQL-Query-2-63-H *** */

SELECT     CAST (TDayOfMonth AS Char (2)) + ' / ' +
           TMonth + ' / ' + TYear AS DisplayDate
FROM       NDX;
```

The SQL Statement (as created with Expression Builder) for Microsoft Access 2013 is:

```
/* *** SQL-Query-2-63-H-Access *** */

SELECT    [NDX]![TDayOfMonth]
          &'/'&[NDX]![TMonth]
          &'/'&[NDX]![TYear] AS DisplayDate
FROM      NDX;
```

The Microsoft Access 2013 result is:



2.67 *It is possible that volume (the number of shares traded) has some correlation with the direction of the stock market. Use the SQL you have learned in this chapter to investigate that possibility. Develop at least five different SQL statements in your investigation.*

If volume is correlated with the direction of the stock market, this means that there should be either:

(1) POSITIVE CORRELATION: Higher volume when the market closes higher, or

(2) NEGATIVE CORRELATION: Higher volume when the market closes lower.

When does the market close higher? When NDX.ChangeClose is positive.

```
/* *** SQL-Query-2-64-A *** */

SELECT    TMonth, TDayOfMonth, TYear, ChangeClose
FROM      NDX
WHERE     ChangeClose > 0;
```

SQL-Query-2-64-A

TMonth	TDayOfMonth	TYear	ChangeClose
January	8	2004	16.3900000000001
January	7	2004	13
January	6	2004	4.68000000000006
January	5	2004	33.01
December	29	2003	26.5100000000002
December	26	2003	0.670000000000073
December	23	2003	16.46
December	22	2003	5.53999999999996
December	18	2003	31.3099999999999
December	16	2003	6.46000000000004

Record: 1 of 2506 No Filter Search

When does the market close lower? When NDX.ChangeClose is negative.

```
/* *** SQL-Query-2-64-B *** */

SELECT    TMonth, TDayOfMonth, TYear, ChangeClose
FROM      NDX
WHERE     ChangeClose < 0;
```

SQL-Query-2-64-B

TMonth	TDayOfMonth	TYear	ChangeClose
January	9	2004	-10.1900000000001
January	2	2004	-4.35000000000014
December	31	2003	-2.08999999999992
December	30	2003	-0.3599999999999
December	24	2003	-4.98000000000002
December	19	2003	-5.13999999999987
December	17	2003	-3.27999999999997
December	15	2003	-20.45
December	9	2003	-34.3899999999999
December	5	2003	-25.47

Record: 1 of 2099 No Filter Search

Now, what are the average positive and negative changes?

```
/* *** SQL-Query-2-64-C *** */

SELECT    AVG (ChangeClose) AS AvgPositiveChange
FROM      NDX
WHERE     ChangeClose > 0;
```

SQL-Query-2-64-C

AvgPositiveChange
15.8756384676776

Record: 1 of 1 No Filter Search

```
/* *** SQL-Query-2-64-D *** */

SELECT    AVG (ChangeClose) AS AvgNegativeChange
FROM      NDX
WHERE     ChangeClose < 0;
```

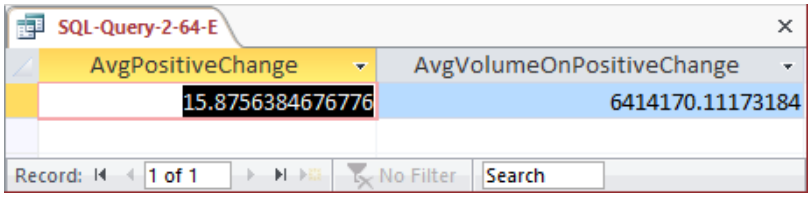
SQL-Query-2-64-D

AvgPositiveChange
-18.3364316341114

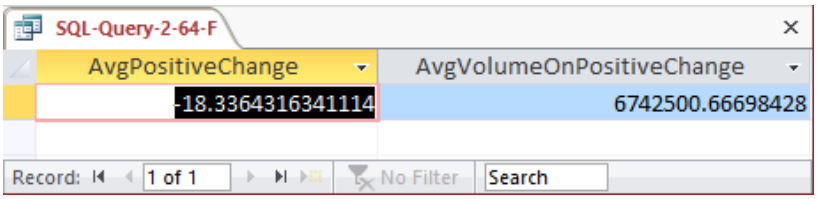
Record: 1 of 1 No Filter Search

Now, what are the average volumes associated with the positive and negative changes?

```
/* *** SQL-Query-2-64-E *** */

SELECT    AVG (ChangeClose) AS AvgPositiveChange,
          AVG (Volume) AS AvgVolumeOnPositiveChange
FROM      NDX
WHERE     ChangeClose > 0;
```




```
/* *** SQL-Query-2-64-F *** */

SELECT     AVG (ChangeClose) AS AvgNegativeChange,
           AVG (Volume) AS AvgVolumeOnNegativeChange
FROM       NDX
WHERE      ChangeClose < 0;
```



So, when there is a positive, or upward, change in the market we have an average volume of 641417.1117318 shares traded, and when we have a negative, or downward, change in the market we have an average volume of 6742500.66698428 shares. These numbers do not look significantly different, we will conclude that there is no correlation between the direction of the market movement and the volume of shares traded (if we wanted to be more formal, we could use a statistical procedure and do a hypothesis test as to whether or not there is really a statistically significant difference between these two numbers).

MARCIA'S DRY CLEANING CASE QUESTIONS

Marcia Wilson owns and operates Marcia's Dry Cleaning, which is an upscale dry cleaner in a well-to-do suburban neighborhood. Marcia makes her business stand out from the competition by providing superior customer service. She wants to keep track of each of her customers and their orders. Ultimately, she wants to notify them that their clothes are ready via e-mail. To provide this service, she has developed an initial database with several tables. Three of those tables are the following:

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

INVOICE (InvoiceNumber, *CustomerNumber*, DateIn, DateOut, TotalAmount)

INVOICE_ITEM (*InvoiceNumber*, ItemNumber, Item, Quantity, UnitPrice)

In the database schema above, the primary keys are underlined and the foreign keys are shown in italics. The database that Marcia has created is named MDC, and the three tables in the MDC database schema are shown in Figure 2-34.

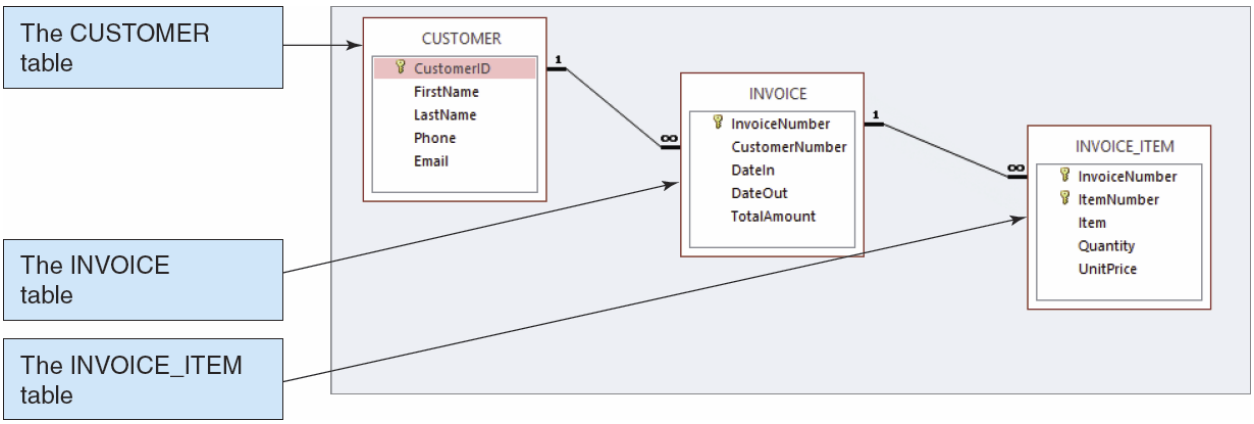


FIGURE 2-34 – The MDC Database

The column characteristics for the tables are shown in Figures 2-34, 2-35, and 2-36. The relationship between CUSTOMER and INVOICE should enforce referential integrity, but not cascade updates or deletions, while the relationship between INVOICE and INVOICE_ITEM should enforce referential integrity and cascade both updates and deletions. The data for these tables are shown in Figures 2-38, 2-39, and 2-40.

We recommend that you create a Microsoft Access 2013 database named MDC-CH02.accdb using the database schema, column characteristics, and data shown above, and then use this database to test your solutions to the questions in this section. Alternatively, SQL scripts for creating the MDC-CH02 database in SQL Server, Oracle Database, and MySQL are available on our Web site at www.pearsonhighered.com/kroenke .

CUSTOMER

Column Name	Type	Key	Required	Remarks
CustomerID	AutoNumber	Primary Key	Yes	Surrogate Key
FirstName	Text (25)	No	Yes	
LastName	Text (25)	No	Yes	
Phone	Text (12)	No	No	
Email	Text (100)	No	No	

Figure 2-35 - Column Characteristics for the CUSTOMER Table

INVOICE

Column Name	Type	Key	Required	Remarks
InvoiceNumber	Number	Primary Key	Yes	Long Integer
CustomerNumber	Number	Foreign Key	Yes	Long Integer
DateIn	Date	No	Yes	
DateOut	Date	No	No	
TotalAmount	Currency	No	No	Two Decimal Places

Figure 2-36 - Column Characteristics for the INVOICE Table

INVOICE_ITEM

Column Name	Type	Key	Required	Remarks
InvoiceNumber	Number	Primary Key, Foreign Key	Yes	Long Integer
ItemNumber	Number	Primary Key	Yes	Long Integer
Item	Text (50)	No	Yes	
Quantity	Number	No	Yes	Long Integer
UnitPrice	Currency	No	Yes	Two Decimal Places

Figure 2-37 - Column Characteristics for the INVOICE_ITEM Table

CustomerID	FirstName	LastName	Phone	Email
1	Nikki	Kaccaton	723-543-1233	Nikki.Kaccaton@somewhere.com
2	Brenda	Catnazaro	723-543-2344	Brenda.Catnazaro@somewhere.com
3	Bruce	LeCat	723-543-3455	Bruce.LeCat@somewhere.com
4	Betsy	Miller	725-654-3211	Betsy.Miller@somewhere.com
5	George	Miller	725-654-4322	George.Miller@somewhere.com
6	Kathy	Miller	723-514-9877	Kathy.Miller@somewhere.com
7	Betsy	Miller	723-514-8766	Betsy.Miller@elsewhere.com

Figure 2-38 - Sample Data for the MDC Database CUSTOMER table

InvoiceNumber	CustomerNumber	DateIn	DateOut	TotalAmount
2013001	1	04-Oct-13	06-Oct-13	$158.50
2013002	2	04-Oct-13	06-Oct-13	$25.00
2013003	1	06-Oct-13	08-Oct-13	$49.00
2013004	4	06-Oct-13	08-Oct-13	$17.50
2013005	6	07-Oct-13	11-Oct-13	$12.00
2013006	3	11-Oct-13	13-Oct-13	$152.50
2013007	3	11-Oct-13	13-Oct-13	$7.00
2013008	7	12-Oct-13	14-Oct-13	$140.50
2013009	5	12-Oct-13	14-Oct-13	$27.00

Figure 2-38 - Sample Data for the MDC Database INVOICE table

InvoiceNumber	ItemNumber	Item	Quantity	UnitPrice
2013001	1	Blouse	2	$3.50
2013001	2	Dress Shirt	5	$2.50
2013001	3	Formal Gown	2	$10.00
2013001	4	Slacks-Mens	10	$5.00
2013001	5	Slacks-Womens	10	$6.00
2013001	6	Suit-Mens	1	$9.00
2013002	1	Dress Shirt	10	$2.50
2013003	1	Slacks-Mens	5	$5.00
2013003	2	Slacks-Womens	4	$6.00
2013004	1	Dress Shirt	7	$2.50
2013005	1	Blouse	2	$3.50
2013005	2	Dress Shirt	2	$2.50
2013006	1	Blouse	5	$3.50
2013006	2	Dress Shirt	10	$2.50
2013006	3	Slacks-Mens	10	$5.00
2013006	4	Slacks-Womens	10	$6.00
2013007	1	Blouse	2	$3.50
2013008	1	Blouse	3	$3.50
2013008	2	Dress Shirt	12	$2.50
2013008	3	Slacks-Mens	8	$5.00
2013008	4	Slacks-Womens	10	$6.00
2013009	1	Suit-Mens	3	$9.00

Figure 2-39 - Sample Data for the MDC Database INVOICE_ITEM table

Write SQL statements and show the results based on the MDC data for each of the following:

A. Show all data in each of the tables.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MDC-A-CUSTOMER *** */

SELECT    *
FROM      CUSTOMER;
```

Note there are two customers both named Betsy Miller.

SQL-Query-MDC-A-CUSTOMER

CustomerID	FirstName	LastName	Phone	Email
1	Nikki	Kaccaton	723-543-1233	Nikki.Kaccaton@somewhere.com
2	Brenda	Catnazaro	723-543-2344	Brenda.Catnazaro@somewhere.com
3	Bruce	LeCat	723-543-3455	Bruce.LeCat@somewhere.com
4	Betsy	Miller	725-654-3211	Betsy.Miller@somewhere.com
5	George	Miller	725-654-4322	George.Miller@somewhere.com
6	Kathy	Miller	723-514-9877	Kathy.Miller@somewhere.com
7	Betsy	Miller	723-514-8766	Betsy.Miller@elsewhere.com

Record: 1 of 7 No Filter Search

```
/* *** SQL-Query-MDC-A-INVOICE *** */

SELECT    *
FROM      INVOICE;
```

SQL-Query-MDC-A-INVOICE

InvoiceNumber	CustomerNumber	DateIn	DateOut	TotalAmount
2013001	1	10/4/2013	10/6/2013	$158.50
2013002	2	10/4/2013	10/6/2013	$25.00
2013003	1	10/6/2013	10/8/2013	$49.00
2013004	4	10/6/2013	10/8/2013	$17.50
2013005	6	10/7/2013	10/11/2013	$12.00
2013006	3	10/11/2013	10/13/2013	$152.50
2013007	3	10/11/2013	10/13/2013	$7.00
2013008	7	10/12/2013	10/14/2013	$140.50
2013009	5	10/12/2013	10/14/2013	$27.00

Record: 1 of 9 No Filter Search

```
/* *** SQL-Query-MDC-A-INVOICE-ITEM *** */

SELECT    *
FROM      INVOICE_ITEM;
```

SQL-Query-MDC-A-INVOICE-ITEM

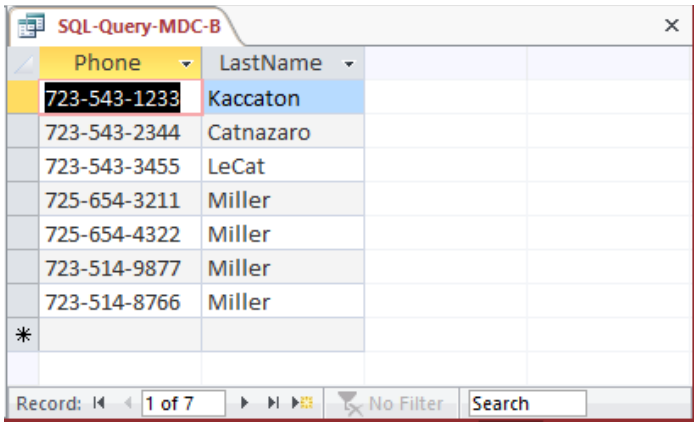
InvoiceNumber	ItemNumber	Item	Quantity	UnitPrice
2013001	1	Blouse	2	$3.50
2013001	2	Dress Shirt	5	$2.50
2013001	3	Formal Gown	2	$10.00
2013001	4	Slacks-Mens	10	$5.00
2013001	5	Slacks-Womens	10	$6.00
2013001	6	Suit-Mens	1	$9.00
2013002	1	Dress Shirt	10	$2.50
2013003	1	Slacks-Mens	5	$5.00
2013003	2	Slacks-Womens	4	$6.00
2013004	1	Dress Shirt	7	$2.50
2013005	1	Blouse	2	$3.50
2013005	2	Dress Shirt	2	$2.50
2013006	1	Blouse	5	$3.50
2013006	2	Dress Shirt	10	$2.50
2013006	3	Slacks-Mens	10	$5.00
2013006	4	Slacks-Womens	10	$6.00
2013007	1	Blouse	2	$3.50
2013008	1	Blouse	3	$3.50
2013008	2	Dress Shirt	12	$2.50
2013008	3	Slacks-Mens	8	$5.00
2013008	4	Slacks-Womens	10	$6.00
2013009	1	Suit-Mens	3	$9.00

Record: 1 of 22 No Filter Search

B. List the LastName, FirstName, and Phone of all customers.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

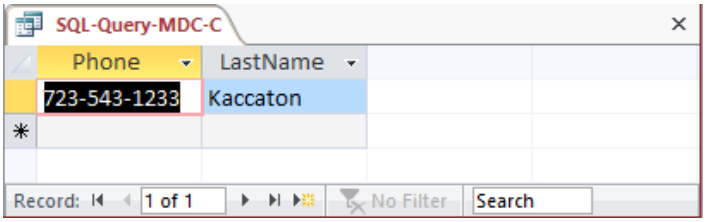
```
/* *** SQL-Query-MDC-B *** */

SELECT    Phone, LastName
FROM      CUSTOMER;
```



C. List the LastName, FirstName, and Phone for all customers with a FirstName of "Nikki".

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

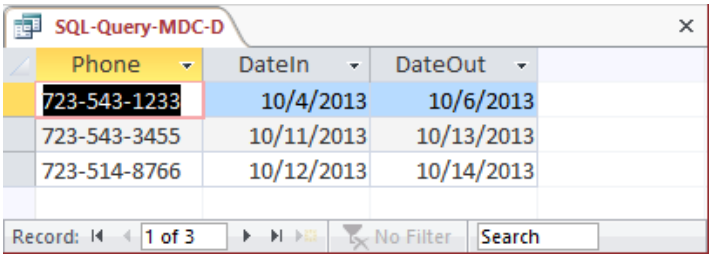
```
/* *** SQL-Query-MDC-C *** */

SELECT    Phone, LastName
FROM      CUSTOMER
WHERE     FirstName = 'Nikki';
```



D. List the LastName, FirstName, Phone, DateIn, and DateOut of all orders in excess of 100.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MDC-D *** */

SELECT   Phone, DateIn, DateOut
FROM     CUSTOMER, INVOICE
WHERE    TotalAmount >100
   AND   CUSTOMER.CustomerID = INVOICE.CustomerNumber;
```



E. List the LastName, FirstName, and Phone of all customers whose first name starts with 'B'.

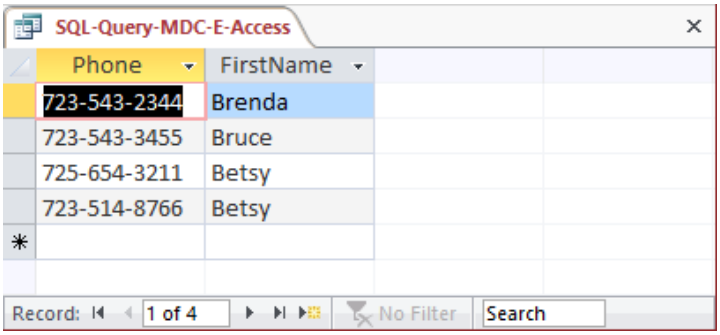
Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

The correct SQL-92 statement, which uses the wildcard %, is:

```
/* *** SQL-Query-MDC-E *** */

SELECT   Phone, FirstName
FROM     CUSTOMER
WHERE    FirstName LIKE 'B%';

/* *** SQL-Query-MDC-E-Access *** */
```

However, Microsoft Access uses the wildcard *, which gives the following SQL statement:

```
/* *** SQL-Query-MDC-E-Access *** */

SELECT   Phone, FirstName
FROM     CUSTOMER
WHERE    FirstName LIKE 'B*';
```



F. *List the LastName, FirstName, and Phone of all customers whose last name includes the characters, 'cat'.*

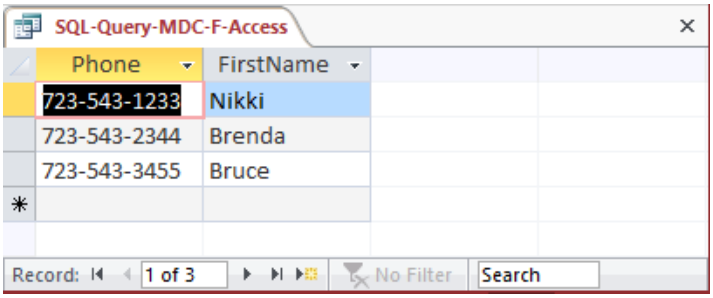
Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

The correct SQL-92 statement, which uses the wildcard %, is:

```
/* *** SQL-Query-MDC-F *** */

SELECT   Phone, FirstName
FROM     CUSTOMER
WHERE    LastName LIKE '%cat%';
```

However, Microsoft Access uses the wildcard *, which give the following SQL statement:

```
/* *** SQL-Query-MDC-F-Access *** */

SELECT   Phone, FirstName
FROM     CUSTOMER
WHERE    LastName LIKE '*cat*';
```



G. *List the LastName, FirstName, and Phone for all customers whose second and third characters of phone number is 23.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

Note that since the phone numbers in this database include the area code, we are really finding phone numbers with '23' as the second and third numbers in the area code. We

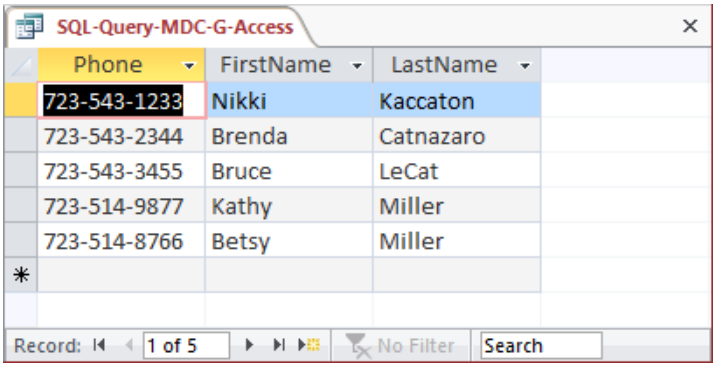
could, off course, write statements to find '23' in the prefix or in the 4-digit sequence portion of the phone number.

The correct SQL-92 statement, which uses the wildcards % and _, is:

```
/* *** SQL-Query-MDC-G *** */

SELECT    Phone, FirstName, LastName
FROM      CUSTOMER
WHERE     Phone LIKE '_23%';
```

However, Microsoft Access uses the wildcards * and ?, which give the following SQL statement:

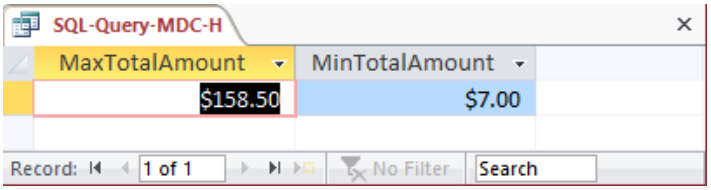
```
/* *** SQL-Query-MDC-G-Access *** */

SELECT    Phone, FirstName, LastName
FROM      CUSTOMER
WHERE     Phone LIKE '?23*';
```



H. Determine the maximum and minimum TotalAmounts.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MDC-H *** */

SELECT    MAX (TotalAmt) AS MaxTotalAmount,
          MIN (TotalAmt) AS MinTotalAmount
FROM      INVOICE;
```

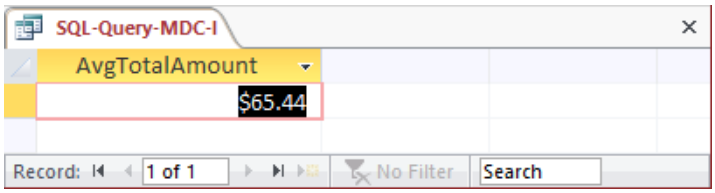


I. Determine the average TotalAmount.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

Note that since ORDER is an SQL reserved word, it must be enclosed in delimiters (square brackets [ ]).

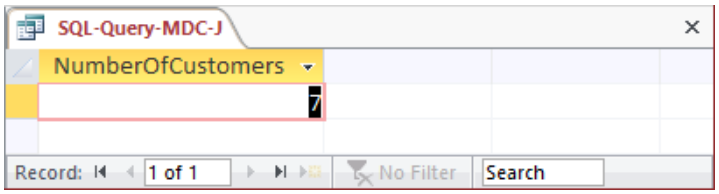
```
/* *** SQL-Query-MDC-I *** */

SELECT    AVG (TotalAmt) AS AvgTotalAmount
FROM      [ORDER];
```



J. Count the number of customers.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

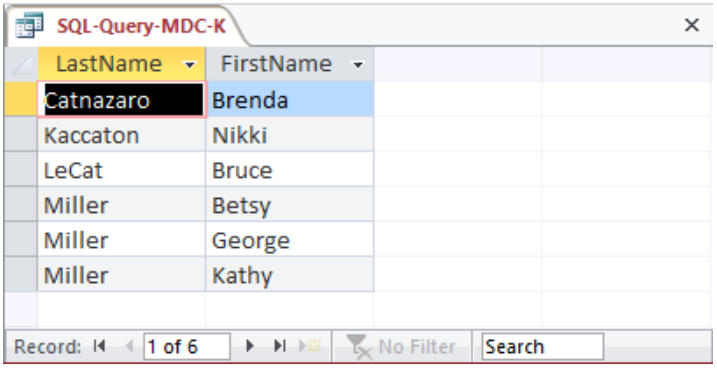
```
/* *** SQL-Query-MDC-J *** */

SELECT    Count (*)AS NumberOfCustomers
FROM      CUSTOMER;
```



K. Group customers by LastName and then by FirstName.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

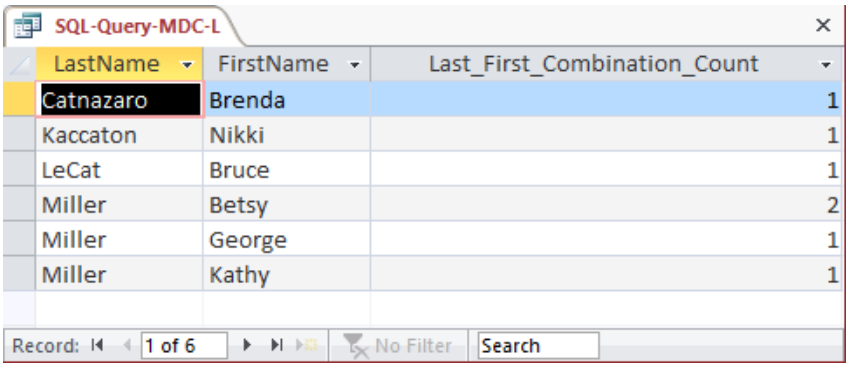
```
/* *** SQL-Query-MDC-K *** */

SELECT   LastName, FirstName
FROM     CUSTOMER
GROUP BY LastName, FirstName;
```



L. Count the number of customers having each combination of LastName and FirstName.

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

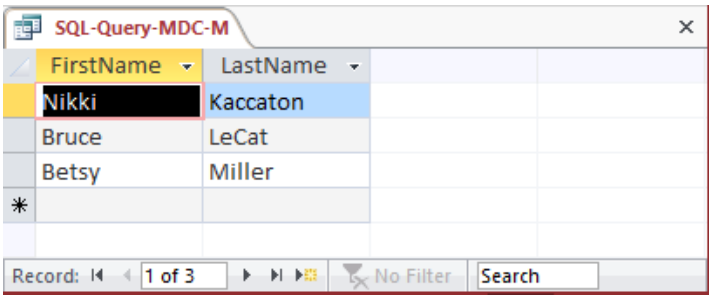
```
/* *** SQL-Query-MDC-L *** */

SELECT   LastName, FirstName,
         COUNT (*) AS Last_First_Combination_Count
FROM     CUSTOMER
GROUP BY LastName, FirstName;
```



M. *Show the FirstName and LastName of all customers who have had an order with TotalAmount greater than $100.00. Use a subquery. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

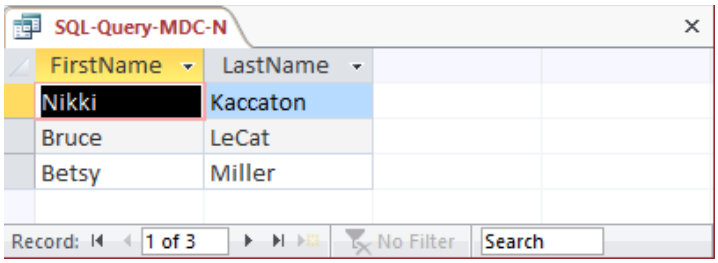
```
/* *** SQL-Query-MDC-M *** */

SELECT   FirstName, LastName
FROM     CUSTOMER
WHERE    CustomerID IN
         (SELECT CustomerNumber
         FROM INVOICE
         WHERE TotalAmount > 100)
ORDER BY LastName, FirstName DESC;
```



N. *Show the LastName, FirstName and Phone of all customers who have had an order with TotalAmount greater than 100. Use a join, but do not use JOIN ON syntax. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MDC-N *** */

SELECT   FirstName, LastName
FROM     CUSTOMER, INVOICE
WHERE    CUSTOMER.CustomerID = INVOICE.CustomerNumber
   AND   TotalAmount > 100
ORDER BY LastName, FirstName DESC;
```

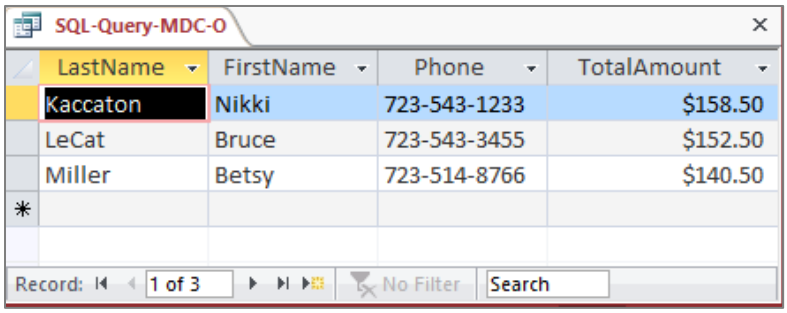


O. *Show the LastName, FirstName and Phone of all customers who have had an order with TotalAmount greater than 100. Use a join using JOIN ON syntax. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

```
/* *** SQL-Query-MDC-O *** */

SELECT    CUSTOMER.LastName, CUSTOMER.FirstName, CUSTOMER.Phone,
          INVOICE.TotalAmount
FROM      CUSTOMER JOIN INVOICE
    ON    CUSTOMER.CustomerID = INVOICE.CustomerNumber
WHERE     INVOICE.TotalAmount>100;
```

Note that for Microsoft Access, we must use the INNER JOIN syntax:

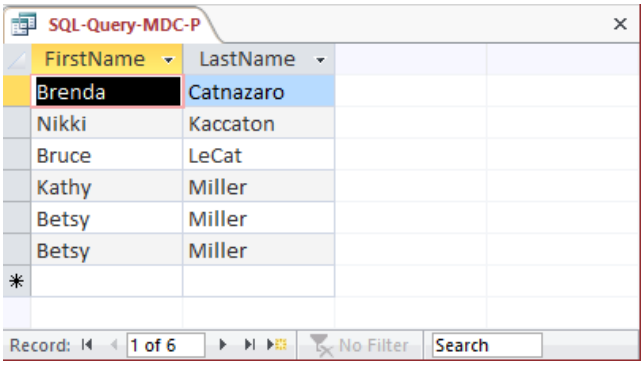
```
/* *** SQL-Query-MDC-O *** */

SELECT    CUSTOMER.LastName, CUSTOMER.FirstName, CUSTOMER.Phone,
          INVOICE.TotalAmount
FROM      CUSTOMER INNER JOIN INVOICE
    ON    CUSTOMER.CustomerID = INVOICE.CustomerNumber
WHERE     INVOICE.TotalAmount>100;
```



P. *Show the LastName, FirstName and Phone of all customers who have had an order with an Item named "Dress Shirt". Use a subquery. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

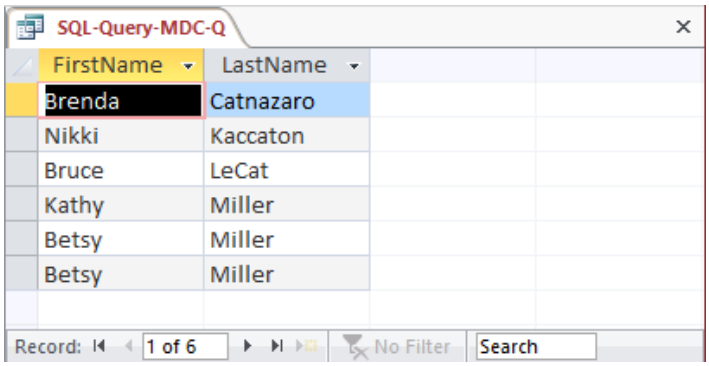
```
/* *** SQL-Query-MDC-P *** */

SELECT    FirstName, LastName
FROM      CUSTOMER
WHERE     CustomerID IN
          (SELECT  CustomerNumber
          FROM     INVOICE
          WHERE    InvoiceNumber IN
                   (SELECT InvoiceNumber
                   FROM INVOICE_ITEM
                   WHERE Item = 'Dress Shirt'))
ORDER BY LastName, FirstName DESC;
```



Q. *Show the LastName, FirstName and Phone of all customers who have had an order with an Item named "Dress Shirt". Use a join, but do not use JOIN ON syntax. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MDC-Q-Access *** */

SELECT   FirstName, LastName
FROM     CUSTOMER, INVOICE, INVOICE_ITEM
WHERE    CUSTOMER.CustomerID = INVOICE.CustomerNumber
   AND   INVOICE.InvoiceNumber = INVOICE_ITEM.InvoiceNumber
   AND   INVOICE_ITEM.Item = 'Dress Shirt'
ORDER BY LastName, FirstName DESC;
```



R. *Show the LastName, FirstName and Phone of all customers who have had an order with an Item named "Dress Shirt". Use a join using JOIN ON syntax. Present the results sorted by LastName in ascending order and then FirstName in descending order.*

```
/* *** SQL-Query-MDC-R *** */

SELECT   CUSTOMER.LastName, CUSTOMER.FirstName,
         CUSTOMER.Phone
FROM     CUSTOMER JOIN INVOICE
      ON CUSTOMER.CustomerID = INVOICE.CustomerNumber)
         JOIN INVOICE_ITEM
               ON INVOICE.InvoiceNumber = INVOICE_ITEM.InvoiceNumber
WHERE    INVOICE_ITEM.Item='Dress Shirt';
```

Note that for Microsoft Access, we must use the INNER JOIN syntax:

```
/* *** SQL-Query-MDC-R-Access *** */

SELECT   CUSTOMER.LastName, CUSTOMER.FirstName,
         CUSTOMER.Phone
FROM     (CUSTOMER INNER JOIN INVOICE
      ON  CUSTOMER.CustomerID = INVOICE.CustomerNumber)
          INNER JOIN INVOICE_ITEM
               ON INVOICE.InvoiceNumber = INVOICE_ITEM.InvoiceNumber
WHERE    (((INVOICE_ITEM.Item)='Dress Shirt'));
```

SQL-Query-MDC-R

LastName	FirstName	Phone
Kaccaton	Nikki	723-543-1233
Catnazaro	Brenda	723-543-2344
Miller	Betsy	725-654-3211
Miller	Kathy	723-514-9877
LeCat	Bruce	723-543-3455
Miller	Betsy	723-514-8766

Record: 1 of 6 No Filter Search

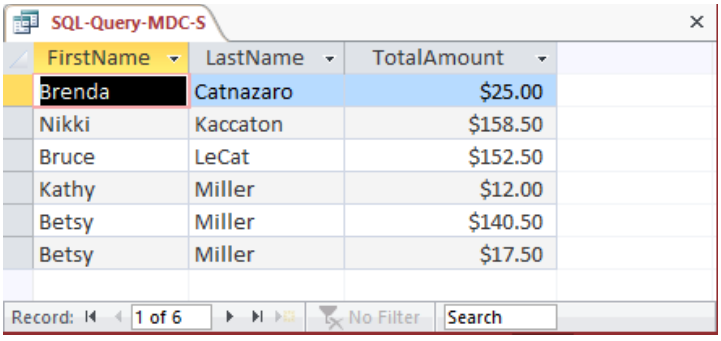
S. *Show the LastName, FirstName, Phone and Total Amount of all customers who have had an order with an Item named "Dress Shirt". Use a combination of a join with a subquery. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to Marcia's Dry Cleaning questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MDC.accdb* which is available at the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

Since we want to display data in fields from two tables, these tables must be combined with a join. Data in a table without displayed fields can still be brought into the query with a

subquery. Therefore, we will join CUSTOMER and INVOICE, while using a subquery with INVOICE_ITEM.

```
/* *** SQL-Query-MDC-s *** */

SELECT   FirstName, LastName, TotalAmount
FROM     CUSTOMER, INVOICE
WHERE    CUSTOMER.CustomerID = INVOICE.CustomerNumber
   AND   INVOICE.InvoiceNumber IN
                 (SELECT InvoiceNumber
                  FROM INVOICE_ITEM
                  WHERE Item = 'Dress Shirt')
ORDER BY LastName, FirstName DESC;
```

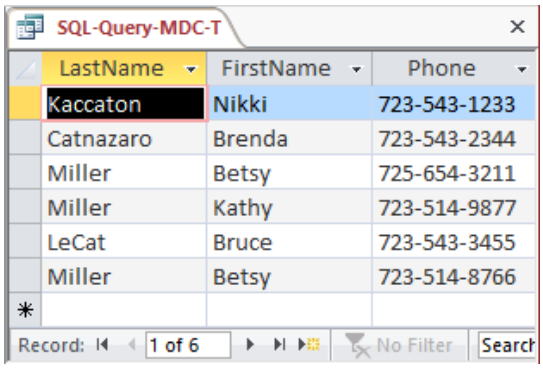


T. *Show the LastName, FirstName, Phone and Total Amount of all customers who have had an order with an Item named "Dress Shirt". Also show the LastName, FirstName and Phone of all other customers. Present results sorted by LastName in ascending order and then FirstName in descending order.*

```
/* *** SQL-Query-MDC- *** */

SELECT   CUSTOMER.LastName, CUSTOMER.FirstName, CUSTOMER.Phone
FROM     CUSTOMER LEFT JOIN INVOICE
    ON   CUSTOMER.CustomerID = INVOICE.CustomerNumber
         LEFT JOIN INVOICE_ITEM
               ON INVOICE.InvoiceNumber = INVOICE_ITEM.InvoiceNumber
WHERE    INVOICE_ITEM.Item='Dress Shirt';
```

Note that for Microsoft Access, we must use the OUTER JOIN syntax:

```
/* *** SQL-Query-MDC-T-Access *** */

SELECT   CUSTOMER.LastName, CUSTOMER.FirstName, CUSTOMER.Phone
FROM     (CUSTOMER LEFT OUTER JOIN INVOICE
    ON    CUSTOMER.CustomerID = INVOICE.CustomerNumber)
          LEFT OUTER JOIN INVOICE_ITEM
               ON INVOICE.InvoiceNumber = INVOICE_ITEM.InvoiceNumber
WHERE    (((INVOICE_ITEM.Item)='Dress Shirt'));
```



ANSWERS TO THE QUEEN ANNE CURIOSITY SHOP PROJECT QUESTIONS

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

Assume that The Queen Anne Curiosity Shop designs a database with the following tables:

CUSTOMER (CustomerID, LastName, FirstName, Address, City, State, ZIP, Phone, Email)

ITEM (ItemID, ItemDescription, CompanyName, PurchaseDate, ItemCost, ItemPrice)

SALE (SaleID, *CustomerID*, SaleDate, SubTotal, Tax, Total)

SALE_ITEM (*SaleID*, SaleItemID, *ItemID*, ItemPrice)

The referential integrity constraints are:

CustomerID in SALE must exist in CustomerID in CUSTOMER

SaleID in SALE_ITEM must exist in SaleID in SALE

ItemID in SALE_ITEM must exist in ItemID in ITEM

Assume that CustomerID of CUSTOMER, ItemID of ITEM, SaleID of SALE, and SaleItemID of SALE_ITEM are all surrogate keys with values as follows:

CustomerID Start at 1 Increment by 1

ItemID Start at 1 Increment by 1

SaleID Start at 1 Increment by 1

The database that The Queen Anne Curiosity Shop has created is named QACS, and the four tables in the QACS database schema are shown in Figure 2-41.

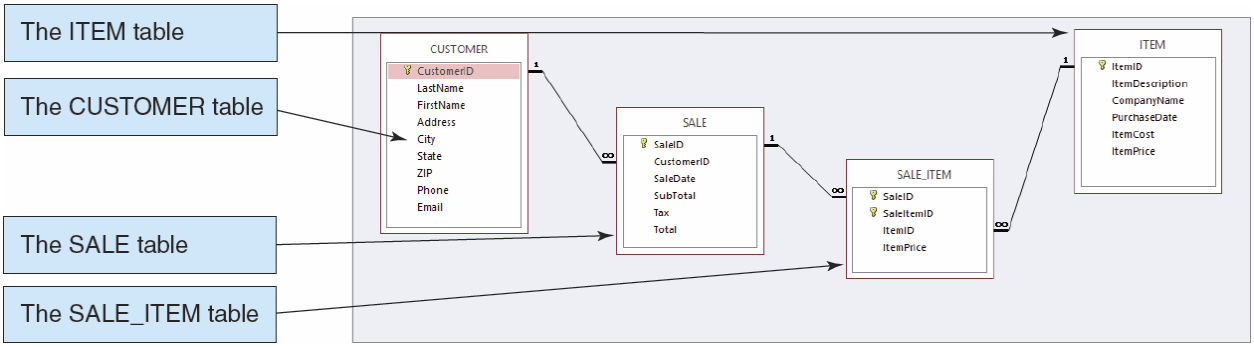


Figure 2-41 – The QACS Database

The column characteristics for the tables are shown in Figures 2-42, 2-43, 2-44, and 2-45. The relationships CUSTOMER-to-SALE and ITEM-to-SALE_ITEM should enforce referential integrity, but not cascade updates nor deletions, while the relationship between SALE and SALE_ITEM should enforce referential integrity and cascade both updates and deletions. The data for these tables are shown in Figures 2-46, 2-47, 2-48, and 2-49.

CUSTOMER

Column Name	Type	Key	Required	Remarks
CustomerID	AutoNumber	Primary Key	Yes	Surrogate Key
LastName	Text (25)	No	Yes	
FirstName	Text (25)	No	Yes	
Address	Text (35)	No	No	
City	Text (35)	No	No	
State	Text (2)	No	No	
ZIP	Text (10)	No	No	
Phone	Text (12)	No	Yes	
Email	Text (100)	No	Yes	

Figure 2-42 - Column Characteristics for the QACS Database CUSTOMER Table

SALE

Column Name	Type	Key	Required	Remarks
SaleID	AutoNumber	Primary Key	Yes	Surrogate Key
CustomerID	Number	Foreign Key	Yes	Long Integer
SaleDate	Date	No	Yes	
SubTotal	Number	No	No	Currency, 2 decimal places
Tax	Number	No	No	Currency, 2 decimal places
Total	Number	No	No	Currency, 2 decimal places

Figure 2-43 - Column Characteristics for the QACS Database SALE Table

SALE_ITEM

Column Name	Type	Key	Required	Remarks
SaleID	Number	Primary Key, Foreign Key	Yes	Long Integer
SaleItemID	Number	Primary Key	Yes	Long Integer
ItemID	Number	Number	Yes	Long Integer
ItemPrice	Number	No	No	Currency, 2 decimal places

Figure 2-44 - Column Characteristics for the QACS Database SALE_ITEM Table

ITEM

Column Name	Type	Key	Required	Remarks
ItemID	AutoNumber	Primary Key	Yes	Surrogate Key
ItemDescription	Text (255)	No	Yes	
CompanyName	Text (100)	No	Yes	
PurchaseDate	Date	No	Yes	
ItemCost	Number	No	Yes	Currency, 2 decimal places
ItemPrice	Number	No	Yes	Currency, 2 decimal places

Figure 2-45 - Column Characteristics for the QACS Database ITEM Table

CustomerID	LastName	FirstName	Address	City	State	ZIP	Phone	Email
1	Shire	Robert	6225 Evanston Ave N	Seattle	WA	98103	206-524-2433	Rober.Shire@somewhere.com
2	Goodyear	Katherine	7335 11th Ave NE	Seattle	WA	98105	206-524-3544	Katherine.Goodyear@somewhere.com
3	Bancroft	Chris	12605 NE 6th Street	Bellevue	WA	98005	425-635-9788	Chris.Bancroft@somewhere.com
4	Griffith	John	335 Aloha Street	Seattle	WA	98109	206-524-4655	John.Griffith@somewhere.com
5	Tierney	Doris	14510 NE 4th Street	Bellevue	WA	98005	425-635-8677	Doris.Tierney@somewhere.com
6	Anderson	Donna	1410 Hillcrest Parkway	Mt. Vernon	WA	98273	360-538-7566	Donna.Anderson@elsewhere.com
7	Svane	Jack	3211 42nd Street	Seattle	WA	98115	206-524-5766	Jack.Svane@somewhere.com
8	Walsh	Denesha	6712 24th Avenue NE	Redmond	WA	98053	425-635-7566	Denesha.Walsh@somewhere.com
9	Enquist	Craig	534 15th Street	Bellingham	WA	98225	360-538-6455	Craig.Enquist@elsewhere.com
10	Anderson	Rose	6823 17th Ave NE	Seattle	WA	98105	206-524-6877	Rose.Anderson@elsewhere.com

Figure 2-46 – Sample Data for the QACS Database CUSTOMER Table

SaleID	CustomerID	SaleDate	SubTotal	Tax	Total
1	1	12/14/2012	$3,500.00	$290.50	$3,790.50
2	2	12/15/2012	$1,000.00	$83.00	$1,083.00
3	3	12/15/2012	$50.00	$4.15	$54.15
4	4	12/23/2012	$45.00	$3.74	$48.74
5	1	1/5/2013	$250.00	$20.75	$270.75
6	5	1/10/2013	$750.00	$62.25	$812.25
7	6	1/12/2013	$250.00	$20.75	$270.75
8	2	1/15/2013	$3,000.00	$249.00	$3,249.00
9	5	1/25/2013	$350.00	$29.05	$379.05
10	7	2/4/2013	$14,250.00	$1,182.75	$15,432.75
11	8	2/4/2013	$250.00	$20.75	$270.75
12	5	2/7/2013	$50.00	$4.15	$54.15
13	9	2/7/2013	$4,500.00	$373.50	$4,873.50
14	10	2/11/2013	$3,675.00	$305.03	$3,980.03
15	2	2/11/2013	$800.00	$66.40	$866.40

Figure 2-47 - Sample Data for the QACS Database SALE Table

SaleID	SaleItemID	ItemID	ItemPrice
1	1	1	$3,000.00
1	2	2	$500.00
2	1	3	$1,000.00
3	1	4	$50.00
4	1	5	$45.00
5	1	6	$250.00
6	1	7	$750.00
7	1	8	$250.00
8	1	9	$1,250.00
8	2	10	$1,750.00
9	1	11	$350.00
10	1	19	$5,000.00
10	2	21	$8,500.00
10	3	22	$750.00
11	1	17	$250.00
12	1	24	$50.00
13	1	20	$4,500.00
14	1	12	$3,200.00
14	2	14	$475.00
15	1	23	$800.00

Figure 2-48 - Sample Data for the QACS Database SALE_ITEM Table

ItemID	ItemDescription	CompanyName	PurchaseDate	ItemCost	ItemPrice
1	Antique Desk	European Specialties	11/7/2012	$1,800.00	$3,000.00
2	Antique Desk Chair	Andrew Lee	11/10/2012	$300.00	$500.00
3	Dining Table Linens	Linens and Things	11/14/2012	$600.00	$1,000.00
4	Candles	Linens and Things	11/14/2012	$30.00	$50.00
5	Candles	Linens and Things	11/14/2012	$27.00	$45.00
6	Desk Lamp	Lamps and Lighting	11/14/2012	$150.00	$250.00
7	Dining Table Linens	Linens and Things	11/14/2012	$450.00	$750.00
8	Book Shelf	Denise Harrion	11/21/2012	$150.00	$250.00
9	Antique Chair	New York Brokerage	11/21/2012	$750.00	$1,250.00
10	Antique Chair	New York Brokerage	11/21/2012	$1,050.00	$1,750.00
11	Antique Candle Holder	European Specialties	11/28/2012	$210.00	$350.00
12	Antique Desk	European Specialties	1/5/2013	$1,920.00	$3,200.00
13	Antique Desk	European Specialties	1/5/2013	$2,100.00	$3,500.00
14	Antique Desk Chair	Specialty Antiques	1/6/2013	$285.00	$475.00
15	Antique Desk Chair	Specialty Antiques	1/6/2013	$339.00	$565.00
16	Desk Lamp	General Antiques	1/6/2013	$150.00	$250.00
17	Desk Lamp	General Antiques	1/6/2013	$150.00	$250.00
18	Desk Lamp	Lamps and Lighting	1/6/2013	$144.00	$240.00
19	Antique Dining Table	Denesha Walsh	1/10/2013	$3,000.00	$5,000.00
20	Antique Sideboard	Chris Bancroft	1/11/2013	$2,700.00	$4,500.00
21	Dining Table Chairs	Specialty Antiques	1/11/2013	$5,100.00	$8,500.00
22	Dining Table Linens	Linens and Things	1/12/2013	$450.00	$750.00
23	Dining Table Linens	Linens and Things	1/12/2013	$480.00	$800.00
24	Candles	Linens and Things	1/17/2013	$30.00	$50.00
25	Candles	Linens and Things	1/17/2013	$36.00	$60.00

Figure 2-49 - Sample Data for the QACS Database ITEM Table

We recommend that you create a Microsoft Access 2013 database named QACS-CH02.accdb using the database schema, column characteristics, and data shown above and then use this database to test your solutions to the questions in this section. Alternatively, SQL scripts for creating the QACS-CH02 database in Microsoft SQL Server, Oracle Database, and MySQL are available on our Web site at www.pearsonhighered.com/kroenke.

Write SQL statements and show the results based on the QACS data for each of the following:

A. Show all data in each of the tables.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-A-CUSTOMER *** */

SELECT    *
FROM      CUSTOMER;
```

SQL-Query-QACS-A-CUSTOMER

CustomerID	LastName	FirstName	Address	City	State	ZIP	Phone	Email
1	Shire	Robert	6225 Evanston	Seattle	WA	98103	206-524-2433	Robert.Shire@somewhere.com
2	Goodyear	Katherine	7335 11th Ave I	Seattle	WA	98105	206-524-3544	Katherine.Goodyear@somewhere.com
3	Bancroft	Chris	12605 NE 6th St	Bellevue	WA	98005	425-635-9788	Chris.Bancroft@somewhere.com
4	Griffith	John	335 Aloha Stree	Seattle	WA	98109	206-524-4655	John.Griffith@somewhere.com
5	Tierney	Doris	14510 NE 4th St	Bellevue	WA	98005	425-635-8677	Doris.Tierney@somewhere.com
6	Anderson	Donna	1410 Hillcrest P	Mt. Vernon	WA	98273	360-538-7566	Donna.Anderson@elsewhere.com
7	Svane	Jack	3211 42nd Stre	Seattle	WA	98115	206-524-5766	Jack.Svane@somewhere.com
8	Walsh	Denesha	6712 24th Aven	Redmond	WA	98053	425-635-7566	Denesha.Walsh@somewhere.com
9	Enquist	Craig	534 15th Street	Bellingham	WA	98225	360-538-6455	Craig.Enquist@elsewhere.com
10	Anderson	Rose	6823 17th Ave I	Seattle	WA	98105	206-524-6877	Rose.Anderson@elsewhere.com
(New)								

Record: 1 of 10 No Filter Search

```
/* *** SQL-Query-QACS-A-SALE *** */

SELECT    *
FROM      SALE;
```

SQL-Query-QACS-A-SALE

SaleID	CustomerID	SaleDate	SubTotal	Tax	Total
1	1	12/14/2012	$3,500.00	$290.50	$3,790.50
2	2	12/15/2012	$1,000.00	$83.00	$1,083.00
3	3	12/15/2012	$50.00	$4.15	$54.15
4	4	12/23/2012	$45.00	$3.74	$48.74
5	1	1/5/2013	$250.00	$20.75	$270.75
6	5	1/10/2013	$750.00	$62.25	$812.25
7	6	1/12/2013	$250.00	$20.75	$270.75
8	2	1/15/2013	$3,000.00	$249.00	$3,249.00
9	5	1/25/2013	$350.00	$29.05	$379.05
10	7	2/4/2013	$14,250.00	$1,182.75	$15,432.75
11	8	2/4/2013	$250.00	$20.75	$270.75
12	5	2/7/2013	$50.00	$4.15	$54.15
13	9	2/7/2013	$4,500.00	$373.50	$4,873.50
14	10	2/11/2013	$3,675.00	$305.03	$3,980.03
15	2	2/11/2013	$800.00	$66.40	$866.40
*	(New)				

Record: 1 of 15 No Filter Search

```
/* *** SQL-Query-QACS-A-SALE-ITEM *** */

SELECT    *
FROM      SALE_ITEM;
```

SQL-Query-QACS-A-SALE-ITEM

SaleID	SaleItemID	ItemID	ItemPrice
1	1	1	$3,000.00
1	2	2	$500.00
2	1	3	$1,000.00
3	1	4	$50.00
4	1	5	$45.00
5	1	6	$250.00
6	1	7	$750.00
7	1	8	$250.00
8	1	9	$1,250.00
8	2	10	$1,750.00
9	1	11	$350.00
10	1	19	$5,000.00
10	2	21	$8,500.00
10	3	22	$750.00
11	1	17	$250.00
12	1	24	$50.00
13	1	20	$4,500.00
14	1	12	$3,200.00
14	2	14	$475.00
15	1	23	$800.00
*			

Record: 1 of 20 No Filter Search

```
/* *** SQL-Query-QACS-A-ITEM *** */

SELECT    *
FROM      ITEM;
```

SQL-Query-QACS-01-ITEM

ItemID	ItemDescription	CompanyName	PurchaseDate	ItemCost	ItemPrice	VendorID
1	Antique Desk	European Specialties	11/7/2012	$1,800.00	$3,000.00	2
2	Antique Desk Chair	Andrew Lee	11/10/2012	$300.00	$500.00	4
3	Dining Table Linens	Linens and Things	11/14/2012	$600.00	$1,000.00	1
4	Candles	Linens and Things	11/14/2012	$30.00	$50.00	1
5	Candles	Linens and Things	11/14/2012	$27.00	$45.00	1
6	Desk Lamp	Lamps and Lighting	11/14/2012	$150.00	$250.00	3
7	Dining Table Linens	Linens and Things	11/14/2012	$450.00	$750.00	1
8	Book Shelf	Denise Harrison	11/21/2012	$150.00	$250.00	5
9	Antique Chair	New York Brokerage	11/21/2012	$750.00	$1,250.00	6
10	Antique Chair	New York Brokerage	11/21/2012	$1,050.00	$1,750.00	6
11	Antique Candle Holder	European Specialties	11/28/2012	$210.00	$350.00	2
12	Antique Desk	European Specialties	1/5/2013	$1,920.00	$3,200.00	2
13	Antique Desk	European Specialties	1/5/2013	$2,100.00	$3,500.00	2
14	Antique Desk Chair	Specialty Antiques	1/6/2013	$285.00	$475.00	9
15	Antique Desk Chair	Specialty Antiques	1/6/2013	$339.00	$565.00	9
16	Desk Lamp	General Antiques	1/6/2013	$150.00	$250.00	10
17	Desk Lamp	General Antiques	1/6/2013	$150.00	$250.00	10
18	Desk Lamp	Lamps and Lighting	1/6/2013	$144.00	$240.00	3
19	Antique Dining Table	Denesha Walsh	1/10/2013	$3,000.00	$5,000.00	7
20	Antique Sideboard	Chris Bancroft	1/11/2013	$2,700.00	$4,500.00	8
21	Dining Table Chairs	Specialty Antiques	1/11/2013	$5,100.00	$8,500.00	9
22	Dining Table Linens	Linens and Things	1/12/2013	$450.00	$750.00	1
23	Dining Table Linens	Linens and Things	1/12/2013	$480.00	$800.00	1
24	Candles	Linens and Things	1/17/2013	$30.00	$50.00	1
25	Candles	Linens and Things	1/17/2013	$36.00	$60.00	1
*	(New)					

Record: 1 of 25 No Filter Search

B. List the LastName, FirstName, and Phone of all customers.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-B *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER;
```

SQL-Query-QACS-B

LastName	FirstName	Phone
Shire	Robert	206-524-2433
Goodyear	Katherine	206-524-3544
Bancroft	Chris	425-635-9788
Griffith	John	206-524-4655
Tierney	Doris	425-635-8677
Anderson	Donna	360-538-7566
Svane	Jack	206-524-5766
Walsh	Denesha	425-635-7566
Enquist	Craig	360-538-6455
Anderson	Rose	206-524-6877

Record: 1 of 10 No Filter Search

C. List the LastName, FirstName, and Phone for all customers with a FirstName of 'John'.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-C *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         FirstName = 'John';
```

SQL-Query-QACS-C

LastName	FirstName	Phone
Griffith	John	206-524-4655

Record: 1 of 1 No Filter Search

D. *List the LastName, FirstName, and Phone of all customers with a last name of 'Anderson'.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-D *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         LastName = 'Anderson';
```

SQL-Query-QACS-D

LastName	FirstName	Phone
Anderson	Donna	360-538-7566
Anderson	Rose	206-524-6877

Record: 1 of 2 No Filter Search

E. *List the LastName, FirstName, and Phone of all customers whose first name starts with 'D'.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

For SQL Server, Oracle Database and MySQL:

```
/* *** SQL-Query-QACS-E *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         FirstName LIKE 'D%';
```

For Microsoft Access:

```
/* *** SQL-Query-QACS-E *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         FirstName LIKE 'D*';
```

SQL-Query-QACS-E

LastName	FirstName	Phone
Tierney	Doris	425-635-8677
Anderson	Donna	360-538-7566
Walsh	Denesha	425-635-7566

Record: 1 of 3 No Filter Search

F. *List the LastName, FirstName, and Phone of all customers whose last name includes the characters 'ne'.*

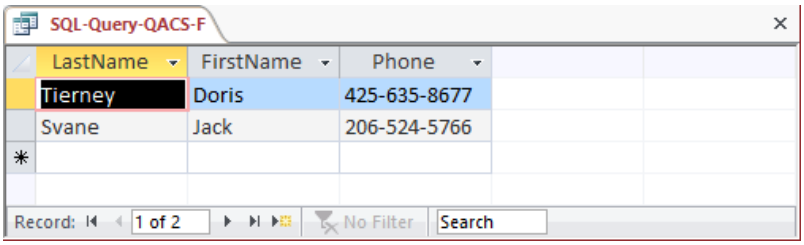
Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

For SQL Server, Oracle Database and MySQL:

```
/* *** SQL-Query-QACS-F *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         LastName LIKE '%ne%';
```

For Microsoft Access:

```
/* *** SQL-Query-QACS-F *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         LastName LIKE '*ne*';
```



G. *List the LastName, FirstName, and Phone for all customers whose second and third numbers (from the right) of their phone number are 56.*

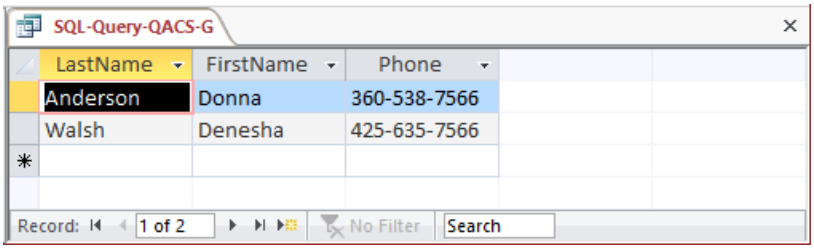
Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

For SQL Server, Oracle Database and MySQL:

```
/* *** SQL-Query-QACS-G *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         Phone LIKE '%56_';
```

For Microsoft Access:

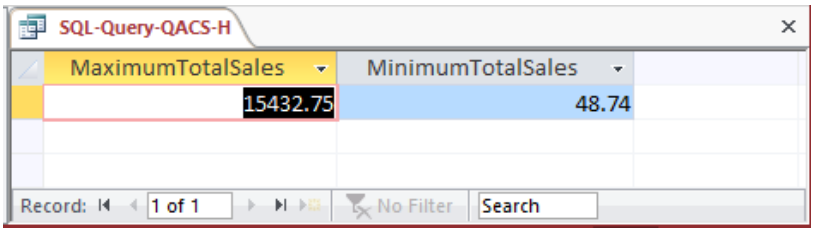
```
/* *** SQL-Query-QACS-G *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         Phone LIKE '*56?';
```



H. Determine the maximum and minimum sales Total.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

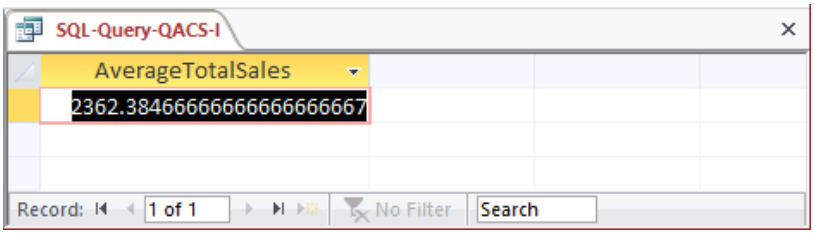
```
/* *** SQL-Query-QACS-H *** */

SELECT        MAX (Total) as MaximumTotalSales,
              MIN (Total) as MinimumTotalSales
FROM          SALE;
```



I. Determine the average sales Total.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

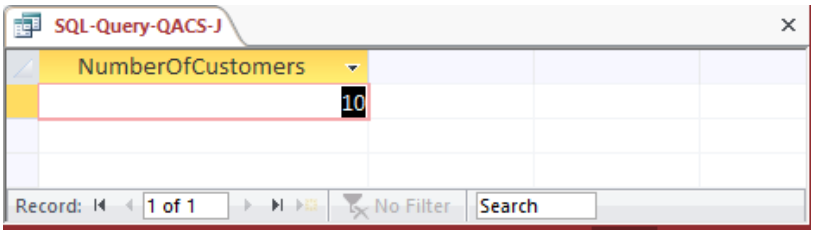
```
/* *** SQL-Query-QACS-I *** */

SELECT        AVG (Total) as AverageTotalSales
FROM          SALE;
```



J. *Count the number of customers.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-J *** */

SELECT        COUNT (*) AS NumberOfCustomers
FROM          CUSTOMER;
```



K. *Group customers by LastName and then by FirstName.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-K *** */

SELECT        LastName, FirstName
FROM          CUSTOMER
GROUP BY      LastName, FirstName;
```

SQL-Query-QACS-K

LastName	FirstName
Anderson	Donna
Anderson	Rose
Bancroft	Chris
Enquist	Craig
Goodyear	Katherine
Griffith	John
Shire	Robert
Svane	Jack
Tierney	Doris
Walsh	Denesha

Record: 1 of 10 No Filter Search

L. Count the number of customers having each combination of LastName and FirstName.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-L *** */

SELECT        LastName, FirstName, COUNT (*) AS NumberOfCustomers
FROM          CUSTOMER
GROUP BY      LastName, FirstName;
```

SQL-Query-QACS-L

LastName	FirstName	NumberOfC
Anderson	Donna	1
Anderson	Rose	1
Bancroft	Chris	1
Enquist	Craig	1
Goodyear	Katherine	1
Griffith	John	1
Shire	Robert	1
Svane	Jack	1
Tierney	Doris	1
Walsh	Denesha	1

Record: 1 of 10 No Filter Search

M. Show the LastName, FirstName, and Phone of all customers who have had an order with Total greater than $100.00. Use a subquery. Present the results sorted by LastName in ascending order and then FirstName in descending order.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-M *** */

SELECT        LastName, FirstName, Phone
FROM          CUSTOMER
WHERE         CustomerID IN
                    (SELECT CustomerID
                     FROM SALE
                     WHERE Total > 100)
ORDER BY      LastName, FirstName DESC;
```

SQL-Query-QACS-M

LastName	FirstName	Phone
Anderson	Rose	206-524-6877
Anderson	Donna	360-538-7566
Enquist	Craig	360-538-6455
Goodyear	Katherine	206-524-3544
Shire	Robert	206-524-2433
Svane	Jack	206-524-5766
Tierney	Doris	425-635-8677
Walsh	Denesha	425-635-7566

Record: 1 of 8 No Filter Search

N. Show the LastName, FirstName, and Phone of all customers who have had an order with Total greater than $100.00. Use a join, but do not use JOIN ON syntax. Present results sorted by LastName in ascending order and then FirstName in descending order.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-E *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER, SALE
WHERE      CUSTOMER.CustomerID = SALE.CustomerID
   AND     Total > 100;

/*       For each CUSTOMER only once:                                    */

SELECT     DISTINCT LastName, FirstName, Phone
FROM       CUSTOMER, SALE
WHERE      CUSTOMER.CustomerID = SALE.CustomerID
   AND     Total > 100;
```

SQL-Query-QACS-N

LastName	FirstName	Phone
Anderson	Donna	360-538-7566
Anderson	Rose	206-524-6877
Enquist	Craig	360-538-6455
Goodyear	Katherine	206-524-3544
Shire	Robert	206-524-2433
Svane	Jack	206-524-5766
Tierney	Doris	425-635-8677
Walsh	Denesha	425-635-7566

Record: 1 of 8 No Filter Search

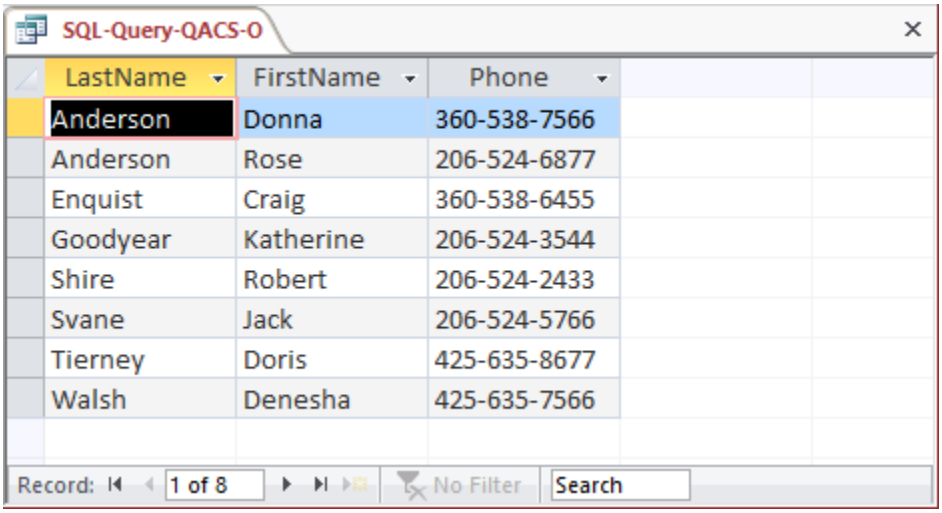
O. *Show the LastName, FirstName, and Phone of all customers who have had an order with Total greater than $100.00. Use a join using JOIN ON syntax. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-O *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER JOIN SALE
   ON      CUSTOMER.CustomerID = SALE.CustomerID
WHERE      Total  > 100;

/*       For each CUSTOMER only once:                                    */

SELECT     DISTINCT LastName, FirstName, Phone
FROM       CUSTOMER JOIN SALE
   ON      CUSTOMER.CustomerID = SALE.CustomerID
WHERE      Total  > 100;
```

Note that for Microsoft Access, we must use the INNER JOIN syntax:

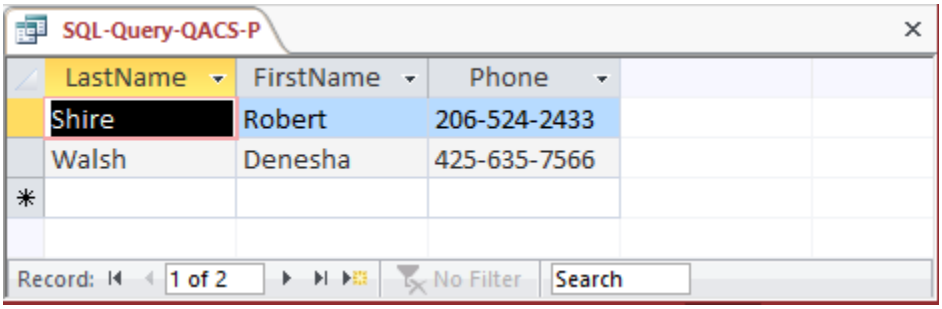
```
SELECT     DISTINCT LastName, FirstName, Phone
FROM       CUSTOMER INNER JOIN SALE
   ON      CUSTOMER.CustomerID = SALE.CustomerID
WHERE      Total  > 100;
```



P. *Show the LastName, FirstName, and Phone of all customers who who have bought an Item named 'Desk Lamp'. Use a subquery. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

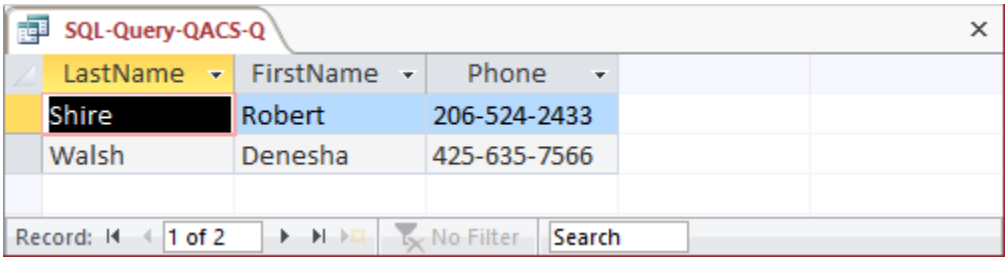
```
/* *** SQL-Query-QACS-E *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER
WHERE      CustomerID IN
           (SELECT    CustomerID
            FROM      SALE
            WHERE     SaleID IN
                      (SELECT    SaleID
                       FROM      SALE_ITEM
                       WHERE     ItemID IN
                                 (SELECT    ItemID
                                  FROM      ITEM
                                  WHERE     ItemDescription = 'Desk Lamp')));
```



Q. *Show the LastName, FirstName, and Phone of all customers who have bought an Item named 'Desk Lamp'. Use a join, but do not use JOIN ON syntax. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-E *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER AS C,
           SALE AS S,
           SALE_ITEM AS SI,
           ITEM AS I
WHERE      C.CustomerID = S.CustomerID
    AND    S.SaleID = SI.SaleID
    AND    SI.ItemID = I.ItemID
    AND    ItemDescription = 'Desk Lamp';
```



R. *Show the LastName, FirstName, and Phone of all customers who have bought an Item named 'Desk Lamp'. Use a join using JOIN ON syntax. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-R *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER AS C JOIN SALE AS S
    ON     C.CustomerID = S.CustomerID
           JOIN   SALE_ITEM AS SI
               ON  S.SaleID = SI.SaleID
                   JOIN   ITEM AS I
                       ON  SI.ItemID = I.ItemID
WHERE      ItemDescription = 'Desk Lamp';
```

Note that for Microsoft Access, we must use the INNER JOIN syntax with grouping of the INNER JOINS:

```
SELECT     LastName, FirstName, Phone
FROM       CUSTOMER AS C INNER JOIN SALE AS S
    ON     ((C.CustomerID = S.CustomerID
           INNER JOIN    SALE_ITEM AS SI)
               ON        S.SaleID = SI.SaleID)
                   INNER JOIN    ITEM AS I
                      ON         SI.ItemID = I.ItemID
WHERE      ItemDescription = 'Desk Lamp';
```

SQL-Query-QACS-R

LastName	FirstName	Phone
Shire	Robert	206-524-2433
Walsh	Denesha	425-635-7566
*		

Record: 1 of 2 No Filter Search

S. Show the LastName, FirstName, and Phone of all customers who have bought an Item named 'Desk Lamp'. Use a combination of a join and a subquery. Present results sorted by LastName in ascending order and then FirstName in descending order.

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-S *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER AS C,
           SALE AS S
WHERE      C.CustomerID = S.CustomerID
    AND    SaleID IN
                    (SELECT    SaleID
                     FROM      SALE_ITEM
                     WHERE     ItemID IN
                               (SELECT    ItemID
                                FROM      ITEM
                                WHERE     ItemDescription = 'Desk Lamp'));
```

SQL-Query-QACS-S

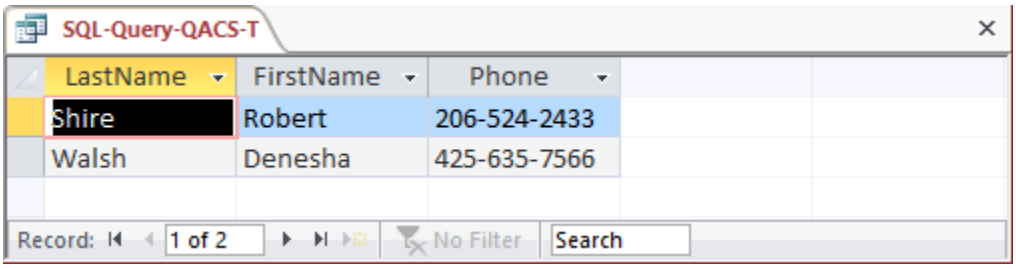
LastName	FirstName	Phone
Shire	Robert	206-524-2433
Walsh	Denesha	425-635-7566

Record: 1 of 2 No Filter Search

T. *Show the LastName, FirstName, and Phone of all customers who have bought an Item named 'Desk Lamp'. Use a combination of a join and a subquery that is different from the combination used for question S. Present results sorted by LastName in ascending order and then FirstName in descending order.*

Solutions to The Queen Anne Curiosity Shop questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-QACS.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-QACS-S *** */

SELECT     LastName, FirstName, Phone
FROM       CUSTOMER AS C,
           SALE AS S,
           SALE_ITEM AS SI
WHERE      C.CustomerID = S.CustomerID
    AND    S.SaleID = SI.SaleID
    AND    ItemID IN
               (SELECT       ItemID
                FROM      ITEM AS I
                WHERE     ItemDescription = 'Desk Lamp');
```



ANSWERS TO MORGAN IMPORTING PROJECT QUESTIONS

James Morgan owns and operates Morgan Importing, which purchases antiques and home furnishings in Asia, ships those items to a warehouse facility in Los Angeles, and then sells these items in the United States. James tracks the Asian purchases and subsequent shipments of these items to Los Angeles by using a database to keep a list of items purchased, shipments of the purchased items, and the items in each shipment. His database includes the following tables:

ITEM (ItemID, Description, PurchaseDate, Store, City, Quantity, LocalCurrencyAmount, ExchangeRate)

SHIPMENT (ShipmentID, ShipperName, ShipperInvoiceNumber, DepartureDate, ArrivalDate, InsuredValue)

SHIPMENT_ITEM (*ShipmentID*, ShipmentItemID, *ItemID*, Value)

In the database schema above, the primary keys are underlined and the foreign keys are shown in italics. The database that James has created is named MI, and the three tables in the MI database schema are shown in Figure 2-50.

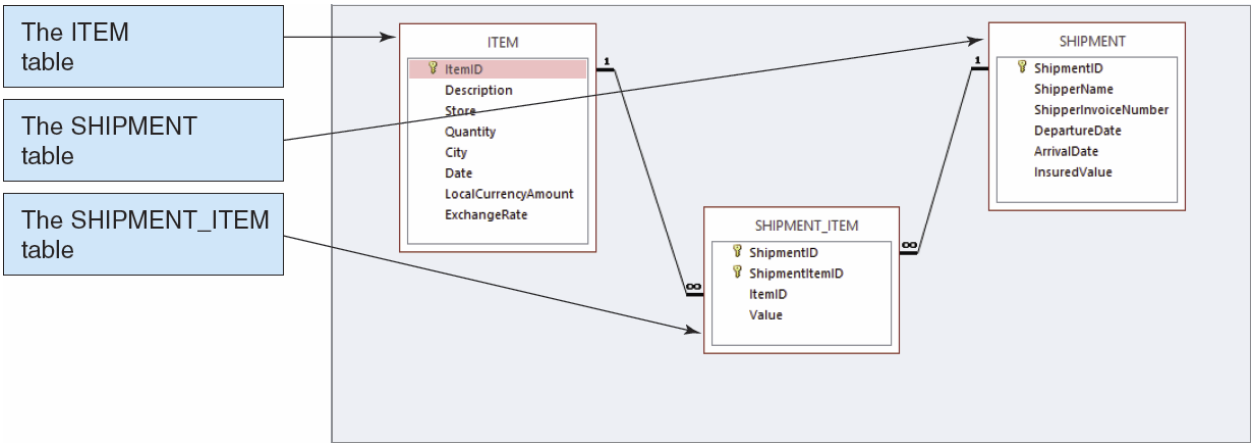


Figure 2-50 – The MI Database

The column characteristics for the tables are shown in Figures 2-51, 2-52, and 2-53. The data for the tables are shown in Figures 2-44, 2-45, and 2-46. The relationship between ITEM and SHIPMENT_ITEM should enforce referential integrity, and although it should cascade updates, it should not cascade deletions. The relationship between SHIPMENT and SHIPMENT_ITEM should enforce referential integrity and cascade both updates and deletions.

We recommend that you create a Microsoft Access 2013 database named MI-Ch02.accdb using the database schema, column characteristics, and data shown above, and then use this database to test your solutions to the questions in this section. Alternatively, SQL scripts for creating the MI-CH02 database in SQL Server, Oracle Database, and MySQL are available on our Web site at www.pearsonhighered.com/kroenke.

ITEM

Column Name	Type	Key	Required	Remarks
ItemID	AutoNumber	Primary Key	Yes	Surrogate Key
Description	Text (255)	No	Yes	Long Integer
PurchaseDate	Date	No	Yes	
Store	Text (50)	No	Yes	
City	Text (35)	No	Yes	
Quantity	Number	No	Yes	Long Integer
LocalCurrencyAmount	Number	No	Yes	Decimal, 18 Auto
ExchangeRate	Number	No	Yes	Decimal, 12 Auto

Figure 2-51 - Column Characteristics for the MI Database ITEM Table

SHIPMENT

Column Name	Type	Key	Required	Remarks
ShipmentID	AutoNumber	Primary Key	Yes	Surrogate Key
ShipperName	Text (35)	No	Yes	
ShipperInvoiceNumber	Number	No	Yes	Long Integer
DepartureDate	Date	No	No	
ArrivalDate	Date	No	No	
InsuredValue	Currency	No	No	Two Decimal Places

Figure 2-52 - Column Characteristics for the MI Database SHIPMENT Table

SHIPMENT_ITEM

Column Name	Type	Key	Required	Remarks
ShipmentID	Number	Primary Key, Foreign Key	Yes	Long Integer
ShipmentItemID	Number	Primary Key	Yes	Long Integer
ItemID	Number	Foreign Key	Yes	Long Integer
Value	Currency	No	Yes	Two Decimal Places

Figure 2-53 - Column Characteristics for the MI Database SHIPMENT_ITEM Table

ItemID	Description	PurchaseDate	Store	City	Quantity	LocalCurrencyAmount	ExchangeRate
1	QE Dining Set	07-Apr-13	Eastern Treasures	Manila	2	403405	0.01774
2	Willow Serving Dishes	15-Jul-13	Jade Antiques	Singapore	75	102	0.5903
3	Large Bureau	17-Jul-13	Eastern Sales	Singapore	8	2000	0.5903
4	Brass Lamps	20-Jul-13	Jade Antiques	Singapore	40	50	0.5903

Figure 2-54 - Sample Data for the MI Database ITEM Table

ShipmentID	ShipperName	ShipperInvoiceNumber	DepartureDate	ArrivalDate	InsuredValue
1	ABC Trans-Oceanic	2008651	10-Dec-12	15-Mar-13	$15,000.00
2	ABC Trans-Oceanic	2009012	10-Jan-13	20-Mar-13	$12,000.00
3	Worldwide	49100300	05-May-13	17-Jun-13	$20,000.00
4	International	399400	02-Jun-13	17-Jul-13	$17,500.00
5	Worldwide	84899440	10-Jul-13	28-Jul-13	$25,000.00
6	International	488955	05-Aug-13	11-Sep-13	$18,000.00

Figure 2-55 - Sample Data for the MI Database SHIPMENT Table

ShipmentID	ShipmentItemID	ItemID	Value
3	1	1	$15,000.00
4	1	4	$1,200.00
4	2	3	$9,500.00
4	3	2	$4,500.00

Figure 2-56 - Sample Data for the MI Database SHIPMENT_ITEM Table

Write SQL statements and show the results based on the MI data for each of the following:

A. Show all data in each of the tables.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-A-ITEM *** */

SELECT    *
FROM      ITEM;
```

SQL-Query-MI-A-ITEM

ItemID	Description	PurchaseDate	Store	City	Quantity	LocalCurrencyAmount	ExchangeRate
1	QE Dining Set	4/7/2013	Eastern Treasures	Manila	2	403405	0.01774
2	Willow Serving Dishes	7/15/2013	Jade Antiques	Singapore	75	102	0.5903
3	Large Bureau	7/17/2013	Eastern Sales	Singapore	8	2000	0.5903
4	Brass Lamps	7/20/2013	Jade Antiques	Singapore	40	50	0.5903
*						0	0

Record: 1 of 4 No Filter Search

```
/* *** SQL-Query-MI-A-SHIPMENT *** */

SELECT    *
FROM      SHIPMENT;
```

SQL-Query-MI-A-SHIPMENT

ShipmentID	ShipperName	ShipperInvoiceNumber	DepartureDate	ArrivalDate	InsuredValue
1	ABC Trans-Oceanic	2008651	12/10/2013	3/15/2013	$15,000.00
2	ABC Trans-Oceanic	2009012	1/10/2013	3/20/2013	$12,000.00
3	Worldwide	49100300	5/5/2013	6/17/2013	$20,000.00
4	International	399400	6/2/2013	7/17/2013	$17,500.00
5	Worldwide	84899440	7/10/2013	7/28/2013	$25,000.00
6	International	488955	8/5/2013	9/11/2013	$18,000.00

Record: 1 of 6 No Filter Search

```
/* *** SQL-Query-MI-A-SHIPMENT-ITEM *** */

SELECT    *
FROM      SHIPMENT_ITEM;
```

SQL-Query-MI-A-SHIPMENT_ITEM

ShipmentID	ShipmentItemID	ItemID	Value
3	1	1	$15,000.00
4	1	4	$1,200.00
4	2	3	$9,500.00
4	3	2	$4,500.00

Record: 1 of 4 No Filter Search

B. List the ShipmentID, ShipperName, and ShipperInvoiceNumber of all shipments.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-B *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber
FROM     SHIPMENT;
```

SQL-Query-MI-B

ShipmentID	ShipperName	ShipperInvoiceNumber
1	ABC Trans-Oceanic	2008651
2	ABC Trans-Oceanic	2009012
3	Worldwide	49100300
4	International	399400
5	Worldwide	84899440
6	International	488955

Record: 1 of 6 No Filter Search

C. List the ShipmentID, ShipperName, and ShipperInvoiceNumber for all shipments with an insured value greater than $10,000.00.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-C *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber
FROM     SHIPMENT
WHERE    InsuredValue > 10000;
```

SQL-Query-MI-C

ShipmentID	ShipperName	ShipperInvoiceNumber
1	ABC Trans-Oceanic	2008651
2	ABC Trans-Oceanic	2009012
3	Worldwide	49100300
4	International	399400
5	Worldwide	84899440
6	International	488955

Record: 1 of 6 No Filter Search

D. List the ShipmentID, ShipperName, and ShipperInvoiceNumber of all shippers whose name starts with "AB".

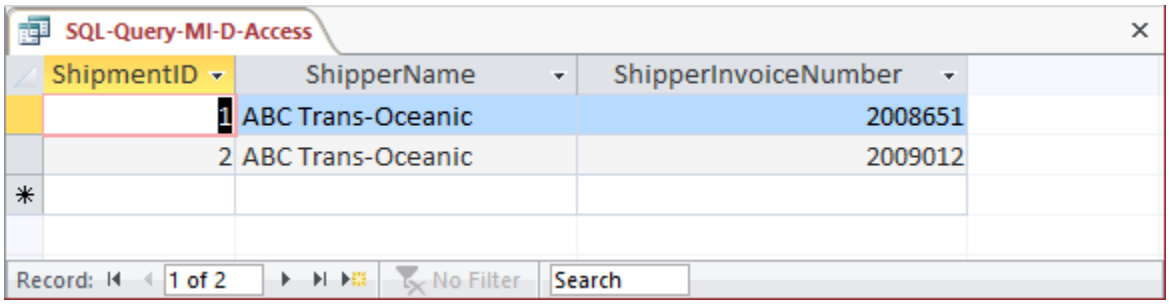
Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

The correct SQL-92 statement, which uses the wildcard %, is:

```
/* *** SQL-Query-MI-D *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber
FROM     SHIPMENT
WHERE    Shipper LIKE 'AB%';
```

However, Microsoft Access uses the wildcard *, which give the following SQL statement:

```
/* *** SQL-Query-MI-D-Access *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber
FROM     SHIPMENT
WHERE    Shipper LIKE 'AB*';
```



E. Assume DepartureDate and ArrivalDate are in the format MM/DD/YY. List the ShipmentID, ShipperName, and ShipperInvoiceNumber and ArrivalDate of all shipments that departed in December.

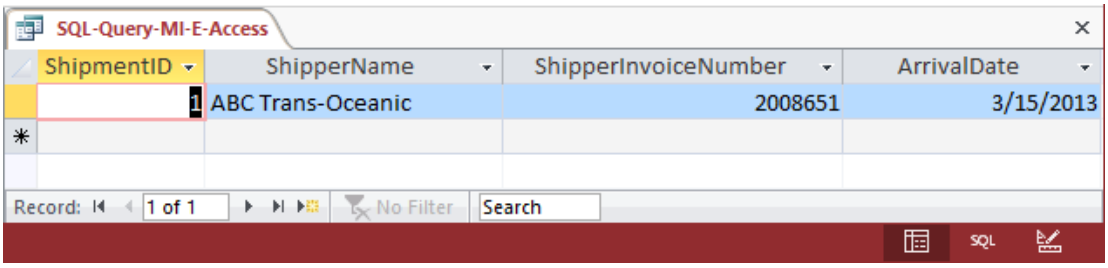
Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

The correct SQL-92 statement, which uses the wildcard %, is:

```
/* *** SQL-Query-MI-E *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber, ArrivalDate
FROM     SHIPMENT
WHERE    DepartureDate LIKE '12%';
```

However, Microsoft Access uses the wildcard *, which gives the following SQL statement:

```
/* *** SQL-Query-MI-E-Access *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber, ArrivalDate
FROM     SHIPMENT
WHERE    DepartureDate LIKE '12*';
```



F. Assume DepartureDate and ArrivalDate are in the format MM/DD/YY. List the ShipmentID, ShipperName, and ShipperInvoiceNumber and ArrivalDate of all shipments that departed on the tenth day of any month.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

The correct SQL-92 statement, which uses the wildcards % and _, is:

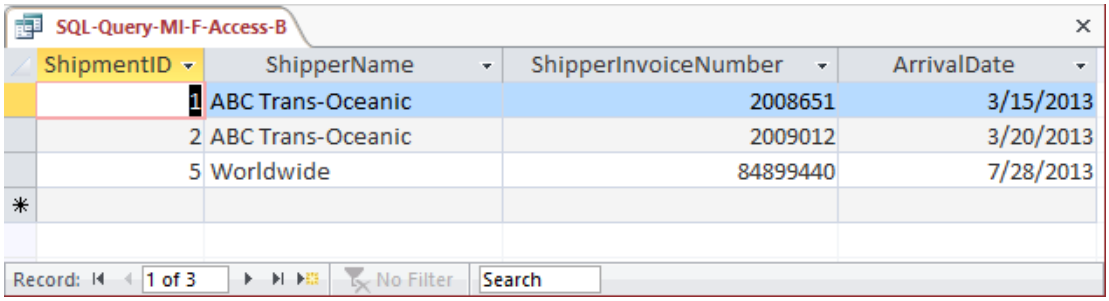
```
/* *** SQL-Query-MI-F *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber, ArrivalDate
FROM     SHIPMENT
WHERE    DepartureDate LIKE '___10%';
```

However, Microsoft Access uses the wildcards * and ?, which give the following SQL statement:

```
/* *** SQL-Query-MI-F-Access-A *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber, ArrivalDate
FROM     SHIPMENT
WHERE    DepartureDate LIKE '???10*';
```

Further, Microsoft Access does NOT show the leading zero in MM, so we must add a compound WHERE clause to get months without the leading zeros:

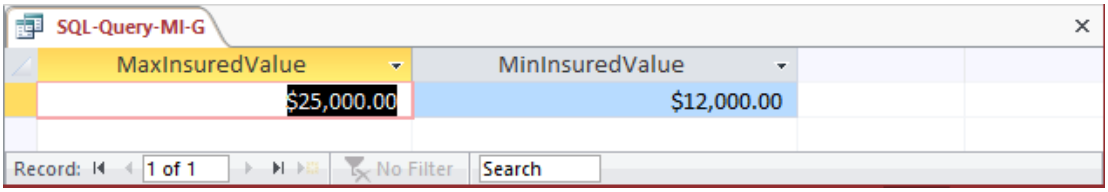
```
/* *** SQL-Query-MI-F-Access-B *** */

SELECT   ShipmentID, ShipperName, ShipperInvoiceNumber, ArrivalDate
FROM     SHIPMENT
WHERE    DepartureDate LIKE '???10*'
   OR    DepartureDate LIKE '??10*';
```



G. *Determine the maximum and minimum InsuredValue.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

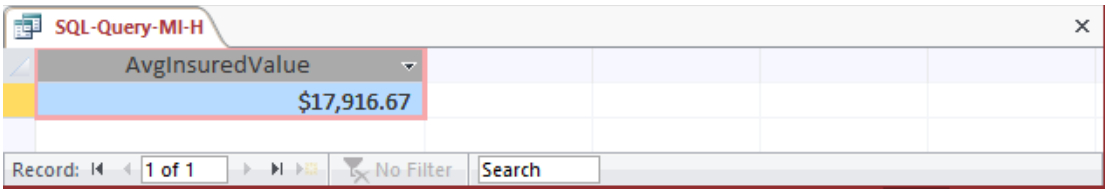
```
/* *** SQL-Query-MI-G *** */

SELECT   MAX (InsuredValue) AS MaxInsuredValue,
         MIN (InsuredValue) AS MinInsuredValue,
FROM     SHIPMENT;
```



H. Determine the average InsuredValue.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

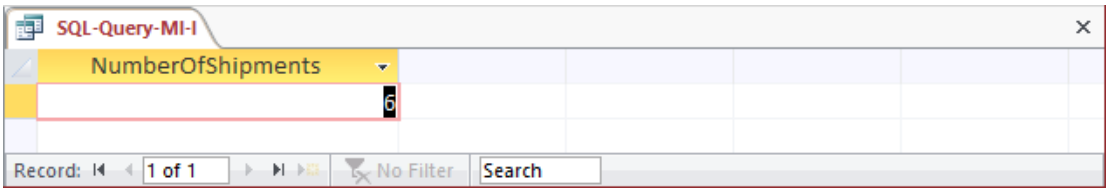
```
/* *** SQL-Query-MI-H *** */

SELECT   AVG (InsuredValue) AS AvgInsuredValue
FROM     SHIPMENT;
```



I. Count the number of shipments.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-I *** */

SELECT   COUNT (*) AS NumberOfShipments
FROM     SHIPMENT;
```



J. Show ItemID, Description, Store, and a calculated column named StdCurrencyAmount that is equal to LocalCurrencyAmt times the ExchangeRate for all rows of ITEM.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-J *** */

SELECT   Item, Store,
         LocalCurrencyAmt * ExchangeRate AS StdCurrencyAmount
FROM     ITEM;
```

SQL-Query-MI-J

Description	Store	StdCurrencyAmount
QE Dining Set	Eastern Treasures	7156.4047
Willow Serving Dishes	Jade Antiques	60.2106
Large Bureau	Eastern Sales	1180.6
Brass Lamps	Jade Antiques	29.515

Record: 1 of 4 No Filter Search

K. Group item purchases by City and Store.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-K *** */

SELECT      City, Store
FROM        ITEM
GROUP BY    City, Store;
```

SQL-Query-MI-K

City	Store
Manila	Eastern Treasures
Singapore	Eastern Sales
Singapore	Jade Antiques

Record: 1 of 3 No Filter Search

L. Count the number of purchases having each combination of City and Store.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-L *** */

SELECT      City, Store
            COUNT (*) AS City_Store_Combination_Count
FROM        ITEM
GROUP BY    City, Store;
```

SQL-Query-MI-L

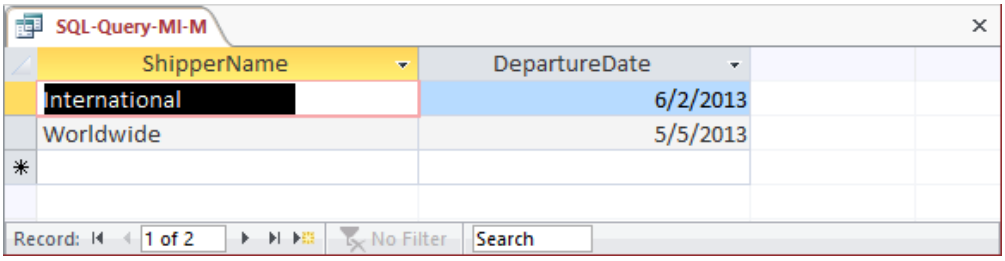
City	Store	City_Store_Combination_Count
Manila	Eastern Treasures	1
Singapore	Eastern Sales	1
Singapore	Jade Antiques	2

Record: 1 of 3 No Filter Search

M. Show the ShipperName, ShipmentID and DepartureDate of all shipments that have an item with a value of 1000 or more. Use a subquery. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-M *** */

SELECT   ShipperName, DepartureDate
FROM     SHIPMENT
WHERE    ShipmentID IN
         (SELECT  ShipmentID
          FROM    SHIPMENT_ITEM
          WHERE   Value = 1000
             OR   Value > 1000)
ORDER BY ShipperName, DepartureDate DESC;
```



N. Show the ShipperName, ShipmentID and DepartureDate of all shipments that have an item with a value of 1000 or more. Use a join. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

This question is a little more complicated than it appears. Note how the following three queries determine that there are actually only two shipments that meet the criteria.

```
/* *** SQL-Query-MI-N-A *** */

SELECT      ShipperName, DepartureDate
FROM        SHIPMENT, SHIPMENT_ITEM
WHERE       SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID
   AND      (Value = 1000 OR Value > 1000)
ORDER BY    ShipperName, DepartureDate DESC;
```

SQL-Query-MI-N-A

ShipperName	DepartureDate
International	6/2/2013
International	6/2/2013
International	6/2/2013
Worldwide	5/5/2013

Record: 1 of 4 No Filter Search

We'll add some more details to confirm the fact that the three lines for International are actually only one shipment. Note that we can use the *greater than or equal to* operator >= to simplify the WHERE clause:

```
/* *** SQL-Query-MI-N-B *** */

SELECT   SHIPMENT.ShipmentID, ShipmentItemID, Description,
         ShipperName, DepartureDate
FROM     SHIPMENT, SHIPMENT_ITEM, ITEM
WHERE    SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID
   AND   SHIPMENT_ITEM.ItemID = ITEM.ItemID
   AND   Value >= 1000
ORDER BY ShipperName, DepartureDate DESC;
```

SQL-Query-MI-N-B

ShipmentID	ShipmentItemID	Description	ShipperName	DepartureDate
4	1	Brass Lamps	International	6/2/2013
4	2	Large Bureau	International	6/2/2013
4	3	Willow Serving Dishes	International	6/2/2013
3	1	QE Dining Set	Worldwide	5/5/2013

Record: 1 of 4 No Filter Search

Now that we can see that all three lines for International are for ShipmentID 4, we'll get the proper results from the revised query by adding the DISTINCT keyword:

```
/* *** SQL-Query-MI-N-C *** */

SELECT   DISTINCT ShipperName, DepartureDate
FROM     SHIPMENT, SHIPMENT_ITEM
WHERE    SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID
   AND   Value >= 1000
ORDER BY ShipperName, DepartureDate DESC;
```

SQL-Query-MI-N-C

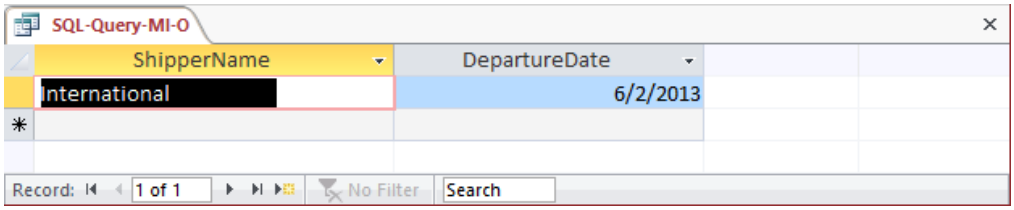
ShipperName	DepartureDate
International	6/2/2013
Worldwide	5/5/2013

Record: 1 of 2 No Filter Search

O. *Show the ShipperName, ShipmentID and DepartureDate of all shipments that have an item that was purchased in Singapore. Use a subquery. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-O *** */

SELECT   ShipperName, DepartureDate
FROM     SHIPMENT
WHERE    ShipmentID IN
         (SELECT  ShipmentID
          FROM    SHIPMENT_ITEM
          WHERE   ItemID IN
                  (SELECT  ItemID
                   FROM    ITEM
                   WHERE   City = 'Singapore'))
ORDER BY ShipperName, DepartureDate DESC;
```

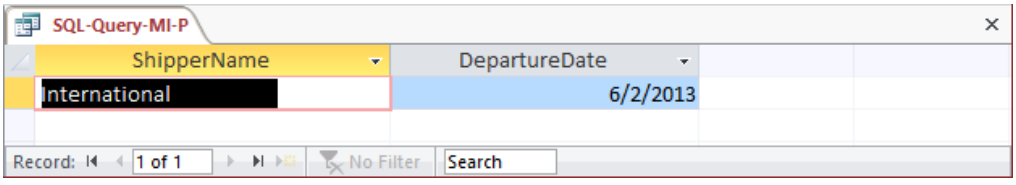


P. *Show the ShipperName, ShipmentID and DepartureDate of all shipments that have an item that was purchased in Singapore. Use a join, but do not use JOIN ON syntax. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

As in question N, we will have to use a DISTINCT keyword to get the appropriate answer.

```
/* *** SQL-Query-MI-P *** */

SELECT   DISTINCT ShipperName, DepartureDate
FROM     SHIPMENT, SHIPMENT_ITEM, ITEM
WHERE    SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID
   AND   SHIPMENT_ITEM.ItemID = ITEM.ItemID
   AND   City = 'Singapore'
ORDER BY ShipperName, DepartureDate DESC;
```



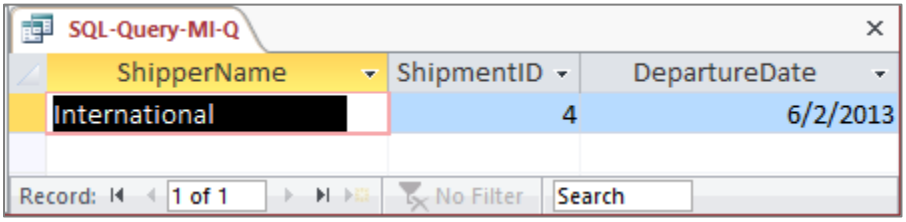
Q. *Show the ShipperName, ShipmentID and DepartureDate of all shipments that have an item that was purchased in Singapore. Use a join using JOIN ON syntax. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-Q *** */

SELECT   SHIPMENT.ShipperName, SHIPMENT_ITEM.ShipmentID,
         SHIPMENT.DepartureDate
FROM     ITEM JOIN SHIPMENT INNER
   ON    ITEM.ItemID = SHIPMENT_ITEM.ItemID
         JOIN SHIPMENT_ITEM
         ON    SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID)
GROUP BY SHIPMENT.ShipperName, SHIPMENT_ITEM.ShipmentID,
         SHIPMENT.DepartureDate, ITEM.City
HAVING   ITEM.City='Singapore';
```

Note that for Microsoft Access, we must use the INNER JOIN syntax:

```
/* *** SQL-Query-MI-Q *** */

SELECT   SHIPMENT.ShipperName, SHIPMENT_ITEM.ShipmentID,
         SHIPMENT.DepartureDate
FROM     ITEM INNER JOIN (SHIPMENT INNER JOIN SHIPMENT_ITEM
    ON   SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID)
         ON ITEM.ItemID = SHIPMENT_ITEM.ItemID
GROUP BY SHIPMENT.ShipperName, SHIPMENT_ITEM.ShipmentID,
          SHIPMENT.DepartureDate, ITEM.City
HAVING   (((ITEM.City)='Singapore'));
```



R. *Show the ShipperName, ShipmentID, DepartureDate of shipment, and Value for items that were purchased in Singapore. Use a combination of a join and a subquery. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-R *** */

SELECT   ShipperName, DepartureDate, Value
FROM     SHIPMENT, SHIPMENT_ITEM
WHERE    SHIPMENT.ShipmentID = SHIPMENT_ITEM.ShipmentID
   AND   ItemID IN
         (SELECT  ItemID
          FROM    ITEM
          WHERE   City = 'Singapore')
ORDER BY ShipperName, DepartureDate DESC;
```

SQL-Query-MI-R

ShipperName	DepartureDate	Value
International	6/2/2013	$1,200.00
International	6/2/2013	$9,500.00
International	6/2/2013	$4,500.00

Record: 1 of 3 No Filter Search

S. *Show the ShipperName, ShipmentID, DepartureDate of shipment, and Value for items that were purchased in Singapore. Also show ShipperName, ShipmentID, DepartureDate for all other shipments. Use a combination of a join and a subquery. Present results sorted by ShipperName in ascending order and then DepartureDate in descending order.*

Solutions to Morgan Importing questions are contained in the Microsoft Access database *DBP-e13-IM-CH02-MI.accdb* which is available in the Instructor's Resource Center on the text's Web site (www.pearsonhighered.com/kroenke).

```
/* *** SQL-Query-MI-S *** */

SELECT   SHIPMENT.ShipperName, SHIPMENT_ITEM.ShipmentID,
         SHIPMENT.DepartureDate, SHIPMENT_ITEM.Value
FROM     (ITEM RIGHT JOIN SHIPMENT_ITEM
    ON    ITEM.ItemID = SHIPMENT_ITEM.ItemID)
          RIGHT JOIN SHIPMENT
          ON SHIPMENT_ITEM.ShipmentID = SHIPMENT.ShipmentID
WHERE    (((ITEM.City)='Singapore')
    AND  ((SHIPMENT.ShipmentID)=[SHIPMENT_ITEM].[ShipmentID])
    AND  ((SHIPMENT_ITEM.[ItemID]) IN
               (SELECT  ItemID
                FROM       ITEM
                WHERE   City = 'Singapore')))
ORDER BY SHIPMENT.ShipperName, SHIPMENT.DepartureDate DESC;
```

SQL-Query-NI-S

ShipperName	ShipmentID	DepartureDate	Value
International	4	6/2/2013	$1,200.00
International	4	6/2/2013	$9,500.00
International	4	6/2/2013	$4,500.00

Record: 1 of 3 No Filter Search