Chapter Introduction

Learning Objectives

- Describe the relational database model
- Explain Query-By-Example (QBE)
- Use criteria in QBE
- Create calculated fields in QBE
- Summarize data by applying aggregate functions in QBE
- Sort data in QBE
- Join tables in QBE
- Update data using action queries in QBE
- Apply relational algebra

Introduction

The database management approach implemented by most systems is the relational model. In this module, you will study the relational database model and examine a visual interface that helps you retrieve data from relational databases, called **Query-By-Example (QBE)** (A visual interface that helps you retrieve data from relational databases.) . Finally, you will learn about relational algebra, which provides the fundamental concepts upon which the manipulation of data in a relational database is rooted.

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-1 Examining Relational Databases
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-1 Examining Relational Databases

A **relational database** is a collection of tables like the ones you viewed for JC Consulting (JCC) in Module 1. These tables also appear in Figure 2-1. Formally, these tables are called **relations** (A two-dimensional table-style collection of data in which all entries are single-valued, each column has a distinct name, all the values in a column are values of the attribute that is identified by the column name, the order of columns is immaterial, each row is distinct, and the order of rows is immaterial. Also called a *table*.) .

Figure 2-1

JC Consulting Data

Employees

EmployeeID	LastName	FirstName	HireDate	Title	Salary
19	Kohn	Ali	01-Jan-20	Project Leader	$5,000.00
22	Kaplan	Franco	01-Feb-20	Programmer	$5,500.00
35	Prohm	Nada	29-Feb-20	Customer Support Specialist	$4,000.00
47	Alvarez	Benito	31-Mar-20	Front End Developer	$5,200.00
51	Shields	Simone	30-Apr-20	Network Specialist	$7,000.00
52	Novak	Stefan	01-Jan-19	Project Leader	$8,000.00
53	Anad	Sergei	01-Jan-19	Front End Developer	$5,300.00
54	Allen	Sasha	01-Jan-19	Programmer	$7,000.00
55	Winter	Wendy	31-Dec-20	Front End Developer	$4,300.00
56	Reddy	Kamal	01-Sep-19	Programmer	$6,200.00
57	Yang	Tam	30-Apr-21	Front End Developer	$5,000.00
58	Young	Solomon	01-Jan-19	Programmer	$5,500.00
59	Santana	Carmen	01-Jan-19	Front End Developer	$4,800.00
60	Lu	Chang	01-Mar-19	Database Developer	$7,900.00

EmployeeID	LastName	FirstName	HireDate	Title	Salary
61	Smirnov	Tovah	01-Oct-19	Programmer	$6,000.00
62	Turner	Jake	31-Mar-21	Database Developer	$7,800.00
63	Geller	Nathan	01-Jan-19	Project Leader	$8,100.00
64	Lopez	Miguel	01-Jan-19	Programmer	$6,200.00
65	Garcia	Hector	01-Apr-23	UI Designer	$7,000.00
66	Roth	Elena	31-Oct-20	Network Specialist	$7,000.00
67	Horvat	Nigel	30-Apr-24	UI Designer	$6,300.00

Clients

ClientID	ClientName	Street	Zip	Government
1	Tri-Lakes Realtors	135 E Jefferson St	02447	FALSE
2	Project Lead The Way	762 Saratoga Blvd	02446	TRUE
3	Midstates Auto Auction	9787 S Campbell Ln	01355	FALSE
4	Bretz & Hanna Law Firm	8101 N Olive Dr	01431	FALSE
5	Aspire Associates	5673 South Ave	01431	FALSE
6	Bounteous	9898 Ohio Ave	02770	FALSE
7	Netsmart Solutions	4091 Brentwood Ln	01354	FALSE
8	Loren Group	9565 Ridge Rd	02466	FALSE
9	Associated Grocers	231 Tecumsa Rd	02532	FALSE
10	Jobot Developers	1368 E 1000 St	02330	FALSE
11	Harper State Bank	1865 Forrest Dr	01571	FALSE
12	MarketPoint Sales	826 Hosta St	01983	FALSE
13	SecureCom Wireless	5280 Industrial Dr	01852	FALSE
14	The HELPCard	840 Boonville Ave	02466	TRUE
15	Jillian Henry & Associates	815 E California St	02113	FALSE
16	Pediatric Group	4940 W Farm Rd	02113	FALSE
17	SkyFactor	1736 Sunshine Dr	02726	FALSE
18	NuCamp	2500 E Kearny St	01431	FALSE
19	Wu Electric	5520 S Michigan	02447	FALSE

ClientID	ClientName	Street	Zip	Government
20	Juxly Engineering	4238 Rumsfield Rd	02148	FALSE
21	Carta Training	2445 N Airport Dr	02446	FALSE

Projects

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes
1	06-Feb-19	1	52	Client wants digital solutions to emphasize commercial real estate.
2	07-Feb-19	10	63	Client needs help converting, organizing, and managing donor and donation data.
3	11-Mar-19	3	52	Client wants to establish SEO goals.
4	10-Apr-20	4	52	Client wants to set up an internal server as well as help with a domain name.
7	02-Sep-19	2	63	Client has used the database for several months and now needs new reports.
8	06-Jan-20	3	52	Develop and implement website SEO strategy.
9	10-Feb-20	6	63	Needs help to manage and organize internal data.
10	31-Mar-21	7	19	Develop new website content.
11	30-Apr-20	9	19	Client needs internal database to manage personnel.
13	30-Nov-20	10	64	Client needs subcontracting help installing a new database for a WordPress site.
14	09-Dec-20	15	19	Client needs new functionality for current JavaScript application.
15	21-Dec-20	14	19	Client needs new functionality for current Ruby/Rails application.

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes
16	04-Jan-21	11	52	Client needs help with server security.
17	15-Feb-21	12	52	Current online sales solution is unreliable.
18	14-Apr-21	6	63	Client needs internal database to manage inventory.
19	04-Jun-21	13	52	Client needs new functionality for current C# / ASP.NET application.
20	30-Jul-21	22	63	Client needs full website reskin.
21	31-Aug-21	16	19	Client needs help with data analytics.
22	30-Sep-21	20	19	Client needs an online reference database
23	12-Nov-21	18	63	Client needs to include responsive web design principles for mobile devices.
24	14-Jan-22	17	63	Client wants an audit on current website performance.

ProjectLineItems

ProjectLineItemID	ProjectID	TaskID	TaskDate	Quantity	Factor	ProjectLineItemNotes
1	1	MEET00	06-Feb-19	1	1.00	
2	1	PLAN01	06-Feb-19	1	1.00	
4	2	MEET00	07-Feb-19	1	1.00	
5	2	PLAN01	07-Feb-19	1	1.00	
6	2	DB01	15-Mar-19	1	1.30	Data is stored in multiple spreadsheets.
7	2	DB02	15-Apr-19	20	1.30	Data is not consistent between spreadsheets.
8	3	MEET00	11-Mar-19	1	1.00	

ProjectLineItemID	ProjectID	TaskID	TaskDate	Quantity	Factor	ProjectLineItemNotes
9	3	PLAN01	11-Mar-19	1	1.20	Owner is difficult to pin down.
10	4	MEET00	10-Apr-20	1	1.00	
11	4	PLAN01	10-Apr-20	1	1.20	Two principal attorneys must agree.
12	4	SERV01	11-May-20	1	1.00	
13	4	SERV02	10-Jun-20	1	1.30	Security is a paramount issue.
17	11	MEET00	30-Apr-20	1	1.00	
18	11	PLAN01	30-Apr-20	1	1.00	
19	9	MEET00	10-Feb-20	1	1.00	
20	9	PLAN01	10-Feb-20	1	1.00	
25	9	PLAN10	17-Feb-20	1	1.00	
26	18	MEET00	14-Apr-21	1	1.00	
27	20	MEET00	30-Jul-21	1	1.00	
28	20	PLAN01	30-Jul-21	1	1.00	
29	20	PLAN02	30-Jul-21	1	1.00	

TaskMasterList

TaskID	Description	CategoryID	Per	Estimate
CODE01	Code PHP	Coding	Hour	$150.00
CODE02	Code C# in ASP.NET	Coding	Hour	$150.00
CODE03	Code Ruby on Rails	Coding	Hour	$150.00
CODE04	Code SQL	Coding	Hour	$150.00
CODE05	Code HTML	Coding	Hour	$100.00
CODE06	Code CSS	Coding	Hour	$100.00
CODE07	Code JavaScript	Coding	Hour	$125.00
CODE08	Perform analytics	Coding	Hour	$100.00
CODE09	Select technology stack	Coding	Hour	$200.00
CODE10	Apply SEO	Coding	Hour	$125.00

TaskID	Description	CategoryID	Per	Estimate
CODE12	Create prototype	Coding	Hour	$150.00
CODE13	Code WordPress	Coding	Hour	$100.00
CODE14	Code Python	Coding	Hour	$150.00
CODE15	Create shopping cart	Coding	Hour	$125.00
CODE16	Code Other	Coding	Hour	$150.00
DB01	Design relational database	Database	Project	$1,000.00
DB02	Convert data	Database	Hour	$125.00
DB03	Install MySQL database	Database	Project	$500.00
DB04	Install SQL Server database	Database	Project	$500.00
DB05	Install Access Database	Database	Project	$400.00
MEET00	Initial customer meeting	Meeting	Project	$0.00

How does a relational database handle entities, attributes of entities, and relationships between entities? Each entity is stored in its own table. For example, the JCC database has a table for employees, for clients, and so on as shown in Figure 2-1. The attributes of an entity become the fields or columns in the table. The table for employees, for example, has a column for the employee ID, a column for the employee's first name, the employee's last name, and so on.

What about relationships? At JC Consulting, there is a one-to-many relationship between clients and projects. (Each client may be related to *many* project estimates created for that client.) How is this relationship implemented in a relational database? The answer is through common columns in the two tables. Consider Figure 2-1 again. The ClientID columns in the Clients and Projects tables implement the relationship between clients and projects. For any client, you can use these columns to determine all the projects that were created for that client. If the Projects table did not include the ClientID value, you could not identify which client that particular project estimate belonged to.

A relation is a two-dimensional table. In Figure 2-1, you might see certain patterns or restrictions that you can place on relations. Each column in a table should have a short but descriptive unique name, and all entries in each column should be consistent with this column name. (For example, in the Salary column, all entries should be for the same length of time, one month.) Each row should contain information for a new item in that entity and should not be repeated. In addition, the order in which columns and rows appear in a table should be immaterial. Rows in a table (relation) are often called **records** (A collection of related fields; can be thought of as a row in a table. Also called a *tuple*.) or **tuples** (A collection of related fields; can be thought of as a row in a table. Also called a *record* or *row*.) . Columns in a table (relation) are often called **fields** or **attributes**. A **relation** is a two-dimensional table (rows and columns) in which the following are true:

1. The entries in the table are single-valued; that is, each intersection of the row and column in the table contains only one value.
2. Each column has a distinct name (technically called the attribute name).
3. All values in a column are values of the same attribute (that is, all entries must match the column name).

4. The order of the columns is not important.

5. Each row is distinct.

6. The order of rows is immaterial.

Later in this text, you will encounter structures in which some of the entries contain **repeating groups** (Multiple entries for a single record in a table.) and, thus, are not single-valued. Such a structure is called an **unnormalized relation** (A structure that satisfies the properties required to be a relation (table) with the exception of allowing repeating groups (the entries in the table do not have to be single-valued).) . (This jargon is a little strange in that an unnormalized relation is not really a relation at all.) The table shown in Figure 2-2 is an example of an unnormalized relation.

Figure 2-2

Structure of an Unnormalized Relation

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes	TaskID	TaskDate	Quantity	Factor	Project
1	06-Feb-19	1	52	Client wants digital solutions to emphasize commercial real estate.	MEET00 PLAN01	06-Feb-19 06-Feb-19	1 1	1.00 1.00	
2	07-Feb-19	10	63	Client needs help converting, organizing, and managing donor and donation data.	MEET00 PLAN01 DB01 DB02 CODE04 TEST01 TEST02 MEET01 SUPP03	07-Feb-19 07-Feb-19 15-Mar-19 15-Apr-19 15-May-19 03-Jun-19 03-Jun-19 03-Jun-19 03-Jun-19	1 1 1 20 4 8 8 2 8	1.00 1.00 1.30 1.30 1.00 1.00 1.00 1.00 1.00	Data is multiple Data is betweei spreads SQL to
3	11-Mar-19	3	52	Client wants to establish SEO goals.	MEET00 PLAN01	11-Mar-19	1 1	1.00 1.20	Owner i down.

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes	TaskID	TaskDate	Quantity	Factor	Project
						11-Mar-19			
4	10-Apr-20	4	52	Client wants to set up an internal server as well as help with a domain name.	MEET00 PLAN01 SERV01 SERV02 TEST01 TEST02 SUPP03	10-Apr-20 10-Apr-20 11-May-20 10-Jun-20 15-Jun-20 15-Jun-20 15-Jun-20	1 1 1 1 16 16 4	1.00 1.20 1.00 1.30 1.00 1.00 1.00	Two pri must ag a param

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-1a Relational Database Shorthand
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-1a Relational Database Shorthand

A commonly accepted shorthand representation shows the structure of a relational database. You write the name of the table and then, within parentheses, list all the columns in the table. In addition, each table should appear on its own line. Using this method, you would write the JC Consulting database as follows:

```
Employees (EmployeeID, LastName, FirstName, HireDate, Title, Salary)
Clients (ClientID, ClientName, Street, Zip, Government)
Projects (ProjectID, ProjectStartDate, ClientID, EmployeeID, ProjectNotes)
ProjectLineItems (ProjectLineItemID, ProjectID, TaskID, TaskDate,
  Quantity, Factor, ProjectLineItemNotes)
TaskMasterList (TaskID, Description, CategoryID, Per, Estimate)
```

The JC Consulting database contains some duplicate column names. For example, the ClientID column appears in *both* the Clients table *and* the Projects table. In some situations, a reference to the column might be confused. For example, if you write ClientID, how would the computer or another user know which table you intend to use? That could be a problem. When a database has duplicate column names, you need a way to indicate the column to which you are referring. One common approach to this problem is to write both the table name and the column name, separated by a period. You would write the ClientID column in the Clients table as Clients.ClientID. You would write ClientID column in the Projects table as Projects.ClientID. Technically, when you combine a column name with a table name, you say that you **qualify** (To indicate the table (relation) of which a given column (attribute) is a part by preceding the column name with the table name. For example, Clients.Street indicates the column named Street in the table named Clients.) the column names. It *always* is acceptable to qualify column names, even when there is no possibility of confusion. If confusion may arise, however, it is *essential* to qualify column names.

The **primary key** (The column or columns that uniquely identify a row in a table.) of a table (relation) is the column or columns that uniquely identify a given row in that table. In the Clients table, the ClientID number uniquely identifies a given row (Figure 2-1). For example, ClientID 6 occurs in only one row of the Clients table. Because each row contains a unique number, the ClientID is the primary key for the Clients table. The primary key provides an important way of distinguishing one row in a table from another and cannot be blank or null. (*Note*: If more than one column is necessary to make the row unique, it is called a **composite primary key** (An identifier used when more than one column is necessary to make a row unique in a table. See also *primary key*.) .)

Primary keys usually are represented by underlining the column or columns that comprises the primary key for each table in the database. The complete representation for the JC Consulting database is as follows:

```
Employees (EmployeeID, LastName, FirstName, HireDate, Title, Salary)
Clients (ClientID, ClientName, Street, Zip, Government)
Projects (ProjectID, ProjectStartDate, ClientID, EmployeeID, ProjectNotes)
ProjectLineItems (ProjectLineItemID, ProjectID, TaskID, TaskDate,
  Quantity, Factor, ProjectLineItemNotes)
TaskMasterList (TaskID, Description, CategoryID, Per, Estimate)
```

Q&A

2-1

Why is it customary to list the primary key field as the first field in a table?

The term **foreign key** (The field used to connect a table on the "many" side of a one-to-many relationship.) refers to the field used to connect a table on the "many" side of a one-to-many relationship. In the previous example, the ClientID field is a primary key in the Clients table and a foreign key in the Projects table. The primary key in the Projects table is ProjectID. You will learn more about foreign keys in a future module.

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-2 Creating Simple Queries and Using Query-By-Example
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

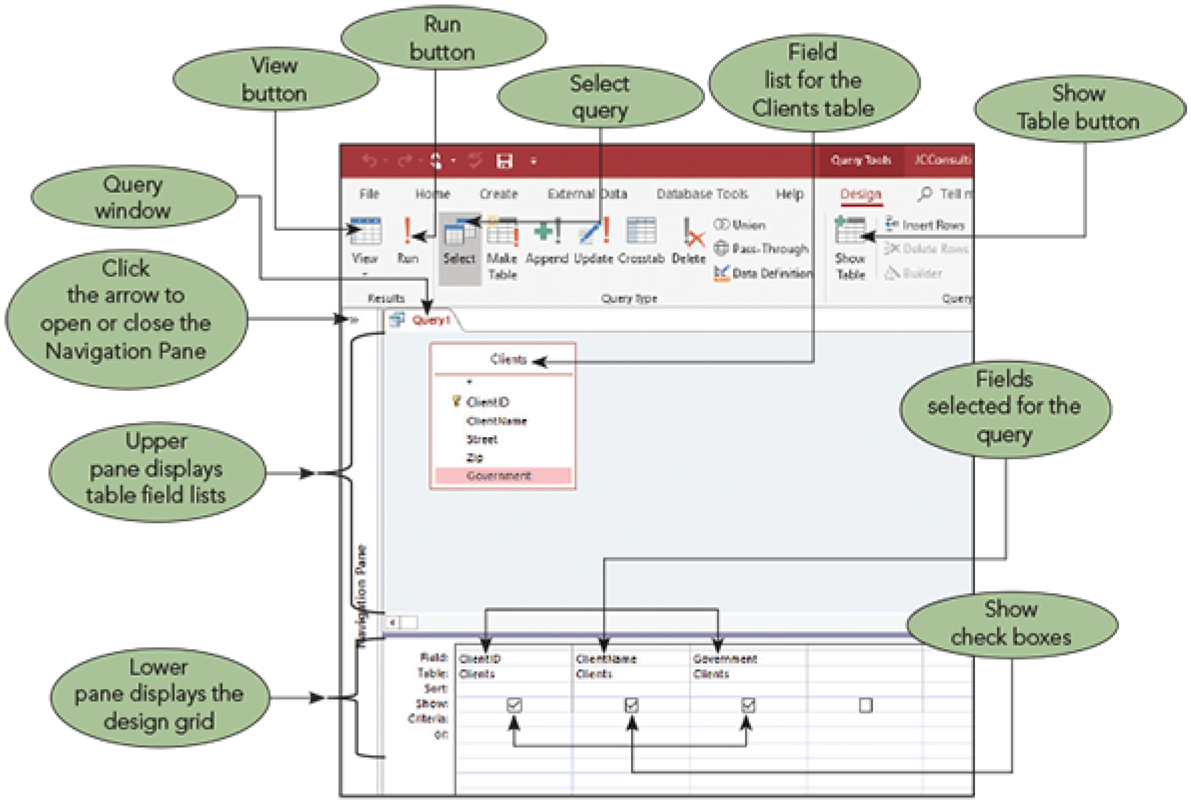
2-2 Creating Simple Queries and Using Query-By-Example

When you ask Access or any other DBMS a question about the data in a database, the question is called a query. A **query** (A question structured in a way that the DBMS can recognize and process.) is a question structured in a way that the DBMS can recognize and process. In this section, you will investigate **Query-By-Example (QBE)**, a visual **GUI (graphical user interface)** (A visual way of interacting with a computer using items such as icons and menus instead of textual commands.) approach to writing queries. For example, using a QBE system, users ask their questions by dragging column names to a grid as opposed to writing commands from a keyboard. Microsoft Access provides a QBE approach to building queries using **Query Design View** (The Access window in which you develop queries by specifying the fields, sort order, and limiting criteria that determine which fields and records are displayed in the resulting datasheet.) .

In Access, Query Design View has two panes. The upper portion of the window contains a field list for each table used in the query, as shown in Figure 2-3. The lower pane contains the **design grid** (In Access, the area in which you specify the fields to include in the query results, a sort order for the query results, any criteria, and other instructions.) , the area in which you specify the fields to include in the query results, a sort order for the query results, any criteria, and other instructions.

Figure 2-3

Fields Added to the Design Grid


Run button
View button
Select query
Field list for the Clients table
Show Table button
Query window
Click the arrow to open or close the Navigation Pane
Upper pane displays table field lists
Fields selected for the query
Show check boxes
Lower pane displays the design grid
Clients
ClientID
ClientName
Street
Zip
Government
Field:
Table:
Sort:
Show:
Criteria:
or:

To create a new, simple query in Access, perform the following steps:

- Click the Create tab on the ribbon.
- Click the Query Design button (Create tab | Queries group) to create a query using Query Design View. Access displays the Show Table dialog box and a new tab on the ribbon named Query Tools Design.
- Click the table in the Show Table dialog box that you want to use in the query.
- Click the Add button (Show Table dialog box) to add the table to the query.
- When you finish adding the tables needed for the query, click the Close button (Show Table dialog box) to close the Show Table dialog box.

A field list for the table or tables you selected appear in the Query Design View window as shown in Figure 2-3. You can resize the field list by dragging any of its borders. You create the query by adding fields and other information to the design grid in the lower portion of the window.

2-2a Selecting Fields and Running the Query

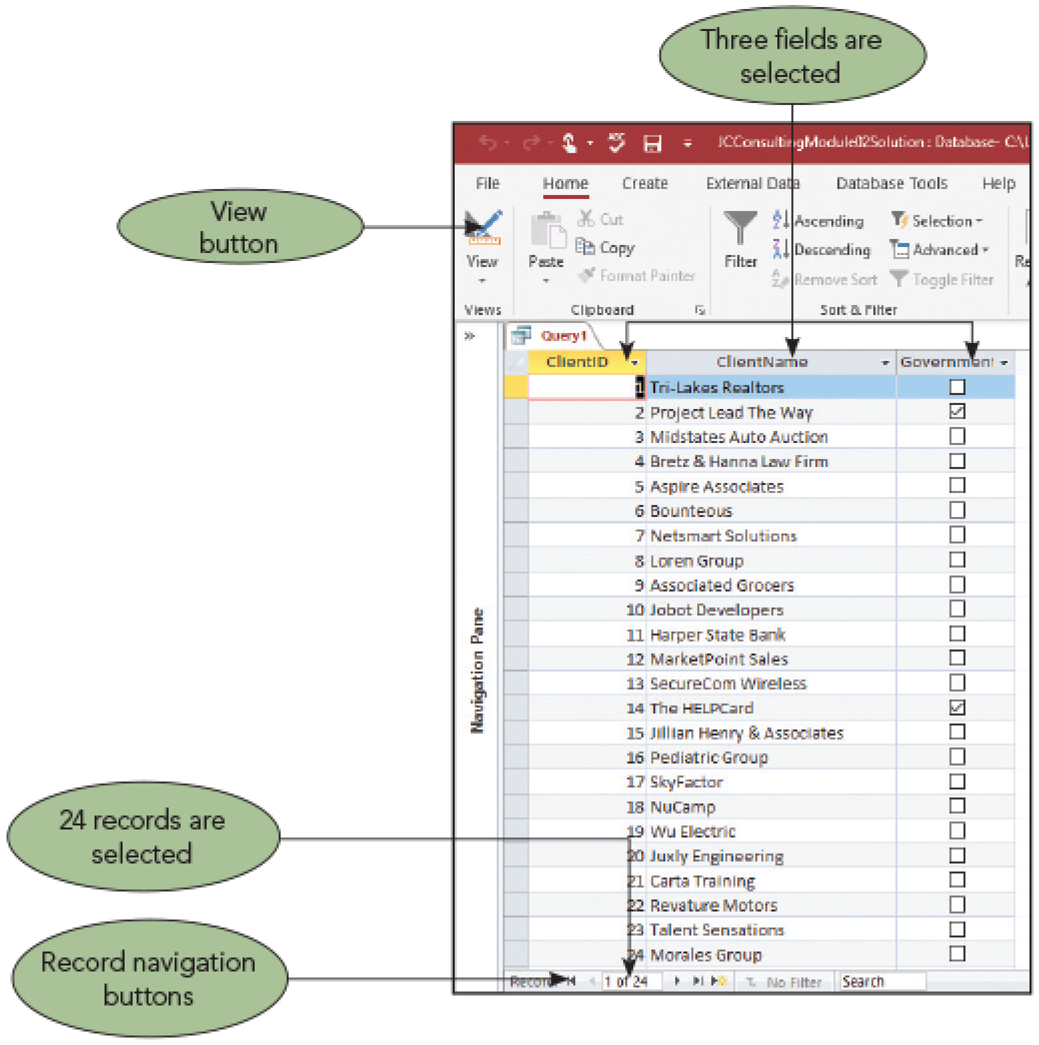
Suppose you need to list the ClientID, ClientName, and Government field values for each record in the Clients table.

To select a field in an Access query, perform the following steps:

- Add the field list for the desired table, such as the Clients table, to Query Design View.
- Double-click the field in the field list to place it in the next available column in the design grid, or drag it from the field list to the desired column in the design grid. In this case, you would double-click ClientID, then ClientName, and then Government in the Clients field list.
- The checkmarks in the Show check boxes indicate the fields that will appear in the query results. To omit a field from the query results, clear the checkmark from the field's Show check box or remove the field from the design grid.
- Click the Run button (Query Tools Design tab | Results group) to run or execute the query and display the query results in Query Datasheet View, as shown in Figure 2-4. If the query is a **select query** (A query that selects fields and records from the database. Also called a *simple query*.) , a query that selects fields and records from the database, clicking the View button (Query Tools Design tab | Results group) also runs the query to select and display the fields and records that the query has identified. A select query is also sometimes called a **simple query** (A query that selects fields and records from the database. Also called a *Select query*.) .

Figure 2-4

Clients Table Query Results



ClientID	ClientName	Government
1	Tri-Lakes Realtors	☐
2	Project Lead The Way	☑
3	Midstates Auto Auction	☐
4	Bretz & Hanna Law Firm	☐
5	Aspire Associates	☐
6	Bounteous	☐
7	Netsmart Solutions	☐
8	Loren Group	☐
9	Associated Grocers	☐
10	Jobot Developers	☐
11	Harper State Bank	☐
12	MarketPoint Sales	☐
13	SecureCom Wireless	☐
14	The HELPCard	☑
15	Jillian Henry & Associates	☐
16	Pediatric Group	☐
17	SkyFactor	☐
18	NuCamp	☐
19	Wu Electric	☐
20	Juxly Engineering	☐
21	Carta Training	☐
22	Revature Motors	☐
23	Talent Sensations	☐
24	Morales Group	☐

Note that the View button's appearance changes when you run a query. For a select query, clicking the View button in Query Datasheet View switches to Query Design View and vice versa.

As shown in Figure 2-4, the record navigation buttons indicate the current record number as well as the total number of records selected in the query.

To add or remove a table from a query, perform the following steps:

- If you add the wrong table to a query, you can remove it in Query Design View by right-clicking the field list title bar, and then clicking Remove Table on the shortcut menu.
- You can add a new table to a query by clicking the Show Table button (Query Tools Design tab | Query Setup group). Access displays the Show Table dialog box, in which you can select and add the desired table.

The two query views you will use in this module are **Datasheet View** (An Access view that shows a table as a collection of rows and columns, similar to a spreadsheet.) to see the results and **Design View** (An Access view in which the structure of an object can be manipulated.) to change the design. It's common to build a select query in incremental steps, switching multiple times between Query Design View, where you build the query, and Query Datasheet View, where you see the selected records, as the query is developed. The View button (Home tab | Views group in Query Datasheet View and Query Tools Design tab | Results group) changes icons to help you quickly flip between these two views.

Running a query in Access does not create a copy of the data, but rather, simply selects the current information from underlying tables where the data is physically stored. If new records are added or existing data is changed, those updates are automatically presented in the results of the query the next time it is run.

You can add and delete records as well as modify data in Datasheet View of a select query. Because a query does not store any data, changes you make to data in the Datasheet View of a select query are actually being stored in an underlying table. The changes are immediately reflected in the current query as well as any other query that selects or otherwise uses that data. The ability to store data in only one location, the tables, yet allow you to select, view, and analyze it in multiple queries is a powerful feature of all modern relational database systems.

2-2b Saving and Using Queries

To save a query, perform the following steps:

- Click the Save button (Quick Access Toolbar).
- Enter a name for the saved query in the Save As dialog box.
- Click the OK button (Save As dialog box). As the data in your database changes over time, the query will always select the most current, accurate data.

After you create and save a query, you can use it in a variety of ways:

- To view the results of a saved select query that is not currently open, run it (which is the same as opening it in Query Datasheet View) by double-clicking the query in the Navigation Pane.
- To change the design of a query that is already open, return to Design View by clicking the View button (Home tab | Views group), and then make the desired changes.
- To change the design of a query that is not currently open, right-click the query in the Navigation Pane, and then click Design View on the shortcut menu to open the query in Design View.
- To print the records selected in Query Datasheet View, click the File tab, click Print, and then click Quick Print.
- To print the query without first opening it, select the query in the Navigation Pane, click the File tab, click Print, and then click Quick Print.
- To save the query with a different name, right-click it in the Navigation Pane, click Rename on the shortcut menu, enter the new name, and then press the Enter key. A query must be closed in order to rename it.

Your Turn

2-1

Create a query to list the task IDs, categories, and task descriptions in the TaskMasterList table.

To display the TaskID, CategoryID, and Description fields for all records in the TaskMasterList table, perform the following steps:

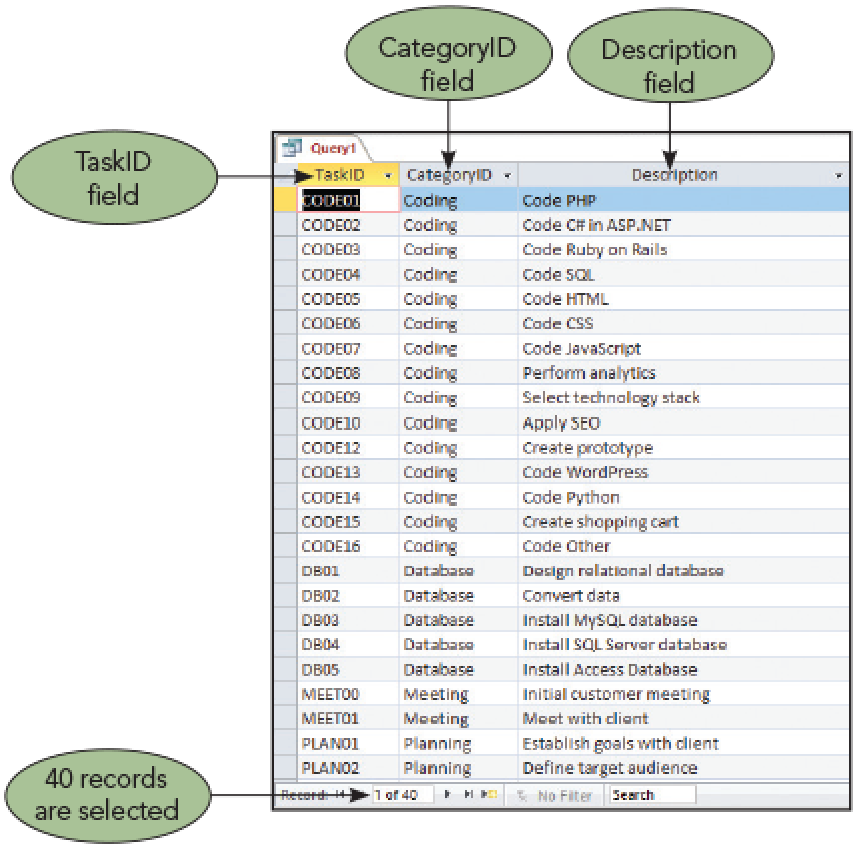
- Begin a new query using the TaskMasterList table.
- Double-click the TaskID field in the TaskMasterList field list to add it to the first column of the query grid.
- Double-click the CategoryID field in the TaskMasterList field list to add it to the first column of the query grid.
- Double-click the Description field in the TaskMasterList field list to add it to the second column of the query grid. The query design is shown in Figure 2-5.
- Click the Run button (Query Tools Design tab | Results group) to display the query results, shown in Figure 2-6.

Figure 2-5

Query to Select Three Fields from the TaskMasterList Table

Figure 2-6

TaskMasterList Table Query Results



TaskID	CategoryID	Description
CODE01	Coding	Code PHP
CODE02	Coding	Code C# in ASP.NET
CODE03	Coding	Code Ruby on Rails
CODE04	Coding	Code SQL
CODE05	Coding	Code HTML
CODE06	Coding	Code CSS
CODE07	Coding	Code JavaScript
CODE08	Coding	Perform analytics
CODE09	Coding	Select technology stack
CODE10	Coding	Apply SEO
CODE12	Coding	Create prototype
CODE13	Coding	Code WordPress
CODE14	Coding	Code Python
CODE15	Coding	Create shopping cart
CODE16	Coding	Code Other
DB01	Database	Design relational database
DB02	Database	Convert data
DB03	Database	Install MySQL database
DB04	Database	Install SQL Server database
DB05	Database	Install Access Database
MEET00	Meeting	Initial customer meeting
MEET01	Meeting	Meet with client
PLAN01	Planning	Establish goals with client
PLAN02	Planning	Define target audience

2-3 Using Simple Criteria

When the records you want must satisfy a condition, you enter that condition in the appropriate column in the query design grid. Conditions also are called **criteria** (More than one criterion or condition.) . (A single condition is called a **criterion** (A statement that can be either true or false. In queries, only records for which the statement is true will be included; also called a *condition*. The plural is *criteria*.) .) The following example illustrates the use of a criterion to select data.

Your Turn

2-2

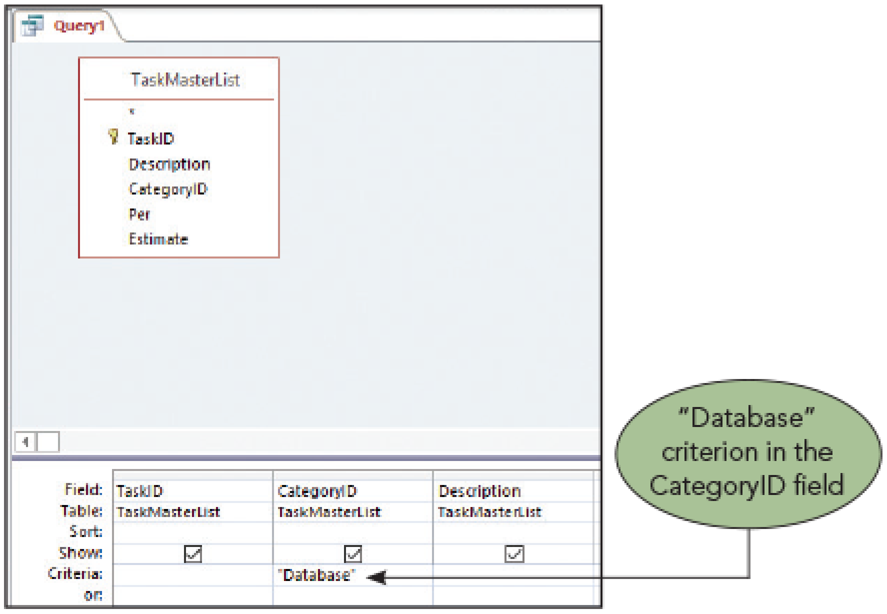
Find the tasks in the Database category.

To enter a criterion for a field, perform the following steps:

- Add the TaskMasterList table to the query.
- Include the TaskID, CategoryID, and Description fields in the design grid.
- Enter Database as the criterion in the row labeled “Criteria” for the CategoryID field, as shown in Figure 2-7.

Figure 2-7

Query to Find the Tasks in the Database Category



Field:	TaskID	CategoryID	Description
Table:	TaskMasterList	TaskMasterList	TaskMasterList
Sort:			
Show:	☑	☑	☑
Criteria:		"Database"	
or:			

When you enter a criterion for text (string) fields, Access automatically adds quotation marks around the criterion when you run the query or when you move the insertion point to another box in the design grid. Typing the quotation marks is optional. (Some database management systems use single quotation marks to identify string criteria.)

According to Figure 2-6, TaskID, Description, and CategoryID all contain text. Figure 2-1 shows the other fields in the TaskMasterList table: Per and Estimate. The Per field also contains text and the Estimate field contains numbers.

The query results shown in Figure 2-8 display those records where Database is the entry in the CategoryID field. In Access, textual criteria are not case sensitive, so “Database”, “database”, or “DATABASE” used as a criterion would all select the same records, but entering your criterion using the same case as the data often makes it easier to read.

Figure 2-8

Database Tasks Query Results

2-3a Parameter Queries

If you plan to use a query repeatedly, but with different criteria, you might want to save it as a parameter query. In Access, a **parameter query** (A query that allows you to enter criterion when you run the query, as opposed to placing it in the Access design grid.) allows you to enter criteria when you *run* the query, as opposed to placing it in the design grid. For example, if you want to search for a different CategoryID each time you run a query, you can type a prompt in the Criteria row of the design grid for the CategoryID field versus a specific criterion such as “Database”. As you run the query, Access displays a prompt that allows you to enter the desired category. Prompts must be enclosed in square brackets. For example, if you type [Enter a category] in the Criteria cell of the design grid for the CategoryID field, Access displays a dialog box when you run the query, allowing you to enter the desired category. A parameter query is easy for novice users to supply information in saved queries.

To enter parameter criteria for a field, perform the following steps:

- Add the table to the query.
- Include the field or fields in the design grid.
- Enter the prompt in [square brackets] in the row labeled “Criteria” for the desired field, as shown in Figure 2-9.

Figure 2-9

Parameter Criterion

2-3b Comparison Operators

A **comparison operator** (An operator used to compare values. Valid operators are =, <, >, <=, >=, < >, and !=. Also called a *relational operator*.) , also called a **relational operator** (An operator used to compare values. Valid operators are =, <, >, <=, >=, < >, and !=. Also called a *comparison operator*.) , can be used in criteria to compare two values in a test that returns true or false. The comparison operators are = (equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to), and <> (not equal to). They are commonly used to compare numeric and data values, and are summarized in Figure 2-10. When you enter criteria without an operator, the = (equal to) operator is assumed. You will learn about other types of **operators** (A mathematical symbol used in an expression to combine different values, resulting in a single value.) as you work through the examples in this text. For example, the familiar math operators of + (add), – (subtract), * (multiply), and / (divide) are commonly across all languages.

Figure 2-10

Comparison Operators

Operator	Name	Description	Example Criterion
=	Equal to	Select all records that exactly match this criterion	="Tanaka" (for a last name field)
>	Greater than	Select all records that are greater than the criterion	>5000 (for a salary field that stores monthly salary values)
<	Less than	Select all records that are less than the criterion	<100 (for a cost field that stores prices)

Operator	Name	Description	Example Criterion
>=	Greater than or equal to	Select all records that are greater than or equal to the criterion	>=5000 (for a salary field that stores monthly salary values)
<=	Less than or equal to	Select all records that are less than or equal to the criterion	<=100 (for a cost field that stores prices)
<>	Not equal to	Select all records that do not match the criterion	<>“MA” (for a state field)

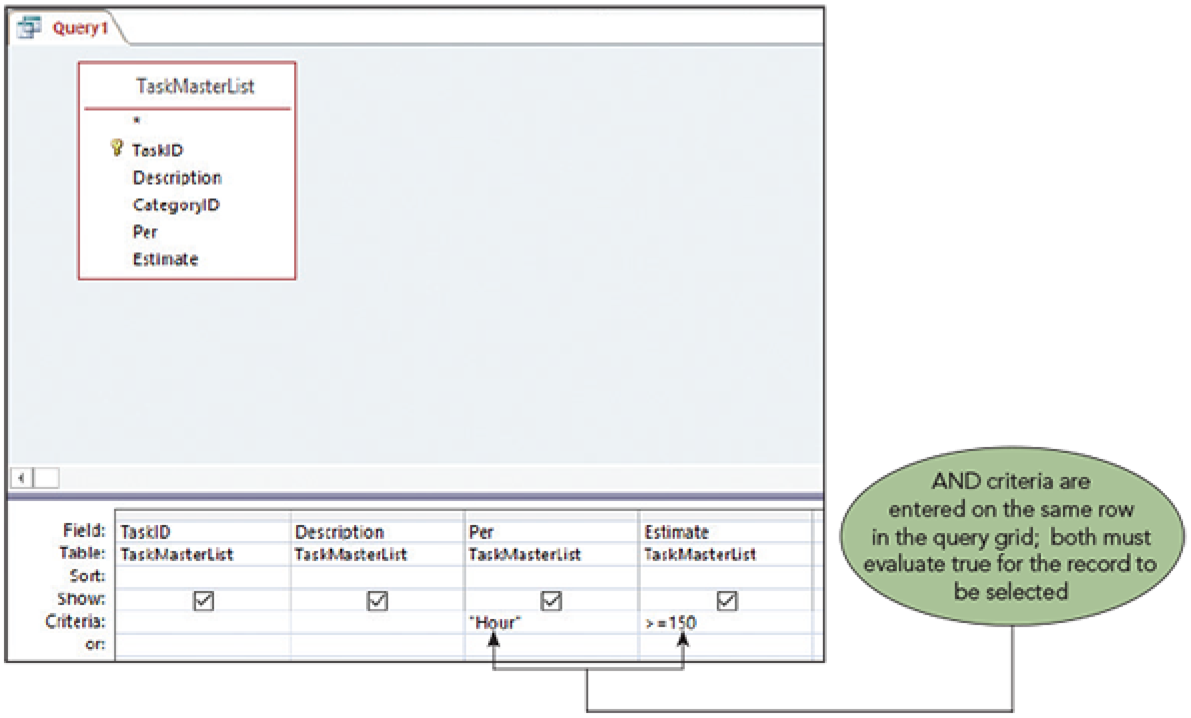
2-4 Using Compound Criteria

You also can combine more than one criterion to create **compound criteria** (Two simple criteria (conditions) in a query that are combined with the AND or OR operators.) , or **compound conditions** (Two simple conditions (criteria) in a query that are combined with the AND or OR operators.) . If you use **AND criteria** (Combination of criteria in which each criterion must be true.) , *each* criterion must be true for a record to be selected. If you use **OR criteria** (Combination of criteria in which only one criterion must be true.) , *only one* of the criterion must be true for a record to be selected. You may add as many AND or OR criteria as desired to create the combination needed for the query. The key is to remember that *all* AND criteria must be true and only *one* OR criterion must be true for the record to be selected.

In QBE, to create an AND criterion, place the conditions on the *same Criteria row* in the design grid (see Figure 2-11). To create an OR criterion, place the conditions on *different Criteria rows* in the design grid (see Figure 2-13).

Figure 2-11

AND Criteria


Query1
TaskMasterList
*
TaskID
Description
CategoryID
Per
Estimate
AND criteria are entered on the same row in the query grid; both must evaluate true for the record to be selected

Field:	TaskID	Description	Per	Estimate
Table:	TaskMasterList	TaskMasterList	TaskMasterList	TaskMasterList
Sort:				
Show:	☑	☑	☑	☑
Criteria:			"Hour"	>=150
or:				

Your Turn

2-3

Using the TaskMasterList table, list the TaskID, Description, Per, and Estimate fields to select the records with a Per field value of Hour *and* an Estimate field value greater than or equal to 150.

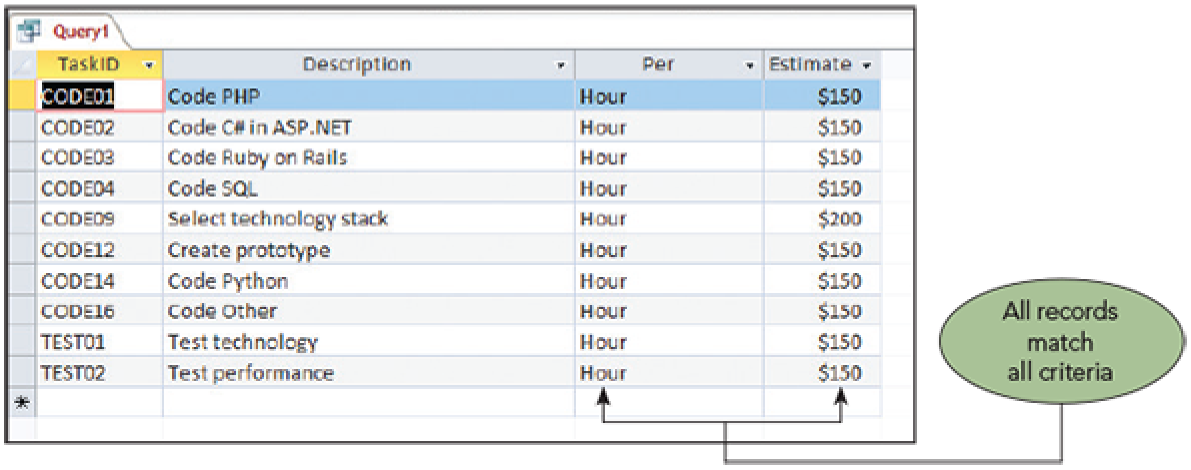
To create a query with AND criteria, perform the following steps:

- Use Query Design View to add the TaskID, Description, Per, and Estimate fields to the query grid from the TaskMasterList table.
- Add the criteria of "Hour" for the Per field and >=150 for the Estimate field to the *same Criteria row*, as shown in Figure 2-11.

The query results appear in Figure 2-12.

Figure 2-12

AND Criteria Query Results



TaskID	Description	Per	Estimate
CODE01	Code PHP	Hour	$150
CODE02	Code C# in ASP.NET	Hour	$150
CODE03	Code Ruby on Rails	Hour	$150
CODE04	Code SQL	Hour	$150
CODE09	Select technology stack	Hour	$200
CODE12	Create prototype	Hour	$150
CODE14	Code Python	Hour	$150
CODE16	Code Other	Hour	$150
TEST01	Test technology	Hour	$150
TEST02	Test performance	Hour	$150

Your Turn

2-4

Using the TaskMasterList table, list the TaskID, Description, Per, and Estimate fields to select the records with a Per field value of Hour *or* an Estimate field value greater than or equal to 150.

To create a query with OR criteria, perform the following steps:

- Use Query Design View to add the TaskID, Description, Per, and Estimate fields to the query grid from the TaskMasterList table.
- Add the criteria of "Hour" for the Per field and >=150 for the Estimate field *using different Criteria rows*, as shown in Figure 2-13.

Figure 2-13

OR Criteria

The query results appear in Figure 2-14.

Figure 2-14

OR Criteria Query Results

Given that a record is selected if it is true for only *one* OR criterion, queries with OR criteria always have the potential to select more records than using the same criteria in an AND query. In Access, this means that for each row of criteria that you add (or for each OR criterion you add), you potentially increase the number of records that are selected by that query.

Using the TaskMasterList table, suppose you need to list the TaskID, Description, Per, and Estimate values for all records that contain “Hour” in the Per field and have Estimate values between 100 and 150.

This example requires you to query a single field for a range of values to find all records that contain an Estimate value between 100 and 150. When you ask this kind of question, you are looking for all values that are greater than or equal to 100 as well as less than or equal to 150. The answer to this question requires using a compound criterion in the same field.

To place two criteria in the same field, perform the following steps:

- Use Query Design View to add the TaskID, Description, Per, and Estimate fields to the query grid from the TaskMasterList table.
- Add the criterion of “Hour” for the Per field and >=100 and <=150 for the Estimate field *using the same Criteria rows*, as shown in Figure 2-15.

Figure 2-15

Query Uses AND Condition for a Single Field to Select a Range of Values

Query criteria are not case sensitive, so AND does not need to be in all capital letters.

An alternate compound condition that tests for a range of values is the **BETWEEN operator** (An operator that allows you to specify a range of values for the criteria, including the lower number, the higher number, and all numbers in between.) . The BETWEEN operator is inclusive, which means it includes the lower number, the higher number, and all numbers in between, as shown in Figure 2-16.

Figure 2-16

Alternative Criterion to Query for a Range of Values

The criteria in Figures 2-15 and 2-16 select the same records, as shown in Figure 2-17.

Figure 2-17

BETWEEN Criterion Query Results

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-5 Creating Computed Fields
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-5 Creating Computed Fields

A **computed field** (A field whose value is computed from other fields in the database; also called a *calculated field*.) or **calculated field** (A field whose value is calculated from other fields in the database; also called a *computed field*.) is a field in a record that is the result of a calculation using data from the rest of the record. Any time you can determine the value of a field using information from existing fields in a record, you should create a computed field. For example, a record in a line item table may include a value for a discount or a tax. By calculating the discount or tax from other data in the record, you can be confident that the discount or tax values are calculated correctly as compared to asking a user to enter the discount or tax values, a process that is both error prone and unproductive.

You can also create computed fields from existing fields that contain text. For example, you may want to combine the values in a FirstName field with the values in a LastName field to create a field that contains both values. Proper database design will always break out the parts of a name into multiple fields so that you can more easily sort, filter, or find data on any individual value. For reporting purposes, however, you may want to create a computed field that displays the entire name as a single value.

Your Turn

2-5

Calculate the annual salary for each employee using the Salary field, which contains a monthly value multiplied by 12. Include the employee’s last name, title, and annual salary in the results.

To include a computed field in a query, perform the following steps:

- Use Query Design View to add the LastName and Title fields from the Employees table to the query grid.
- In the Field row, enter a name for the new computed field, followed first by a colon, and then by a mathematical expression that calculates the desired information.

To calculate the annual salary, you enter the expression *AnnualSalary: [Salary] * 12* in the next blank Field row in the design grid. An **expression** (A combination of data and operators that results in a single value.) is a combination of data and operators that results in a single value.

When entering an expression in the design grid, the default column size may prevent you from seeing the complete expression. To address this, you can either widen the column by dragging its right edge, or right-click the column in the Field cell, and then click Zoom on the shortcut menu to use the Zoom dialog box to enter the expression. Both techniques are shown in Figure 2-18.

Figure 2-18

Using the Zoom Dialog Box

Q&A

2-2

When I run the calculated field query, Access asks me for a parameter value. What should I do?

In Access, when you use a field name in an expression, it is enclosed in [square brackets]. When you are entering the expression, you do not need to include the [square brackets] for field names that do not contain spaces. However, field names that have spaces *require* this syntax, so it's a good habit to include the square brackets around field names any time you are using them in an expression.

If you open the Zoom dialog box to create a computed field, close the dialog box by clicking OK, and then click the Run button (Query Tools Design tab | Results group) to display the query results as shown in Figure 2-19.

Figure 2-19

Query Results with Computed Field

You also can use addition (+), subtraction (–), and division (/) in your expressions. You can include parentheses to indicate which computations Access should perform first. The main thing to understand about computed fields is that they create a new piece of data for *each* record using an expression to calculate that data.

Expressions may also include **functions** (Part of an expression used to calculate the number of entries, the sum or average of all the entries in a given column, or the largest or smallest of the entries in a given column; also called *aggregate function*.) , built-in shortcut formulas that can help you calculate information faster than creating the entire formula yourself. For example, if you want to calculate the number of days between a field named InvoiceDate and today's date to determine the age of that invoice, you could use the following expression: Date()–[InvoiceDate]. The Access Date function returns today's date and the entire expression returns the number of days between today's date and the InvoiceDate.

Function names are always followed by (parentheses). If you need to pass information to the function so it can operate, that information is called the function **argument** (Information a macro action needs to complete processing; also, specific information provided to a function so it can operate.) and is passed inside the parentheses. For example, to use the Int function, pass it a number to evaluate such as Int(5.5). Int is the function, the value 5.5 is the function argument, and the return value of the function is 5. When a function uses multiple arguments to operate, the arguments are separated by commas. If an argument is a field name, it is surrounded by [square brackets]. For example, to use the Sum function to add the values in a field named Salary, use the expression Sum([Salary]).

Every function evaluates to a single piece of data, whether a number, text (commonly called a **string** (Text data, including spaces, symbols, and punctuation marks.)), or a **Boolean** (A data type for fields that store only Yes or No (On or Off, True or False) values.) (true or false). That piece of data is called the function's **return value** (The data a function returns when it completes its task.) .

Common Access functions are shown in Figure 2-20. Some functions require a specific type of data as their argument value. For example, the Len function only works with textual (string) data and returns the number of characters in the string. The Month function only works with a date argument and returns a number that represents the number of the month in a given date (January is 1, February is 2, and so forth). Other functions such as Count, Format, Max, and Min may be used with different types of data including text, numbers, and dates.

Figure 2-20

Common Access Functions

Data Type for the Function Argument	Function	Description and Return Value
Text (String)	LCase	Converts and then returns a string to all lowercase characters
	Left	Returns the number of characters in a string (starting from left)
	Len	Returns the length of a string
	Right	Returns the number of characters from a string (starting from right)
	Trim	Removes both leading and trailing spaces from text (a string) and returns the string
	UCase	Converts and then returns a string to all uppercase characters
Numeric	Avg	Returns the average value of a numeric field for a group of records
	Int	Returns the integer part of a number
	Rnd	Returns a random number between 0 and 1

Data Type for the Function Argument	Function	Description and Return Value
	Round	Rounds and then returns a number to a specified number of decimal places
	Sqr	Returns the square root of a number
	StDev	Returns a number that represents the standard deviation of a numeric field for a group of records
	Sum	Adds then returns the sum of a numeric field for a group of records
	Var	Returns a number that represents the variance of a numeric field for a group of records
	Date	Returns the current system date
Date	DateDiff	Returns the difference between two dates
	Day	Returns the day of the month for a given date
	Hour	Returns the hour part of a time
	Minute	Returns the minute part of a time
	Month	Returns the month part of a given date
	MonthName	Returns the name of the month based on a number
	Now	Returns the current date and time based on the computer’s system date and time
	Second	Returns the seconds part of a time
	Time	Returns the current system time
	Weekday	Returns the weekday number for a given date
	WeekdayName	Returns the weekday name based on a number
	Year	Returns the year part of a given date
	Count	Returns the number of records in a group of records

Data Type for the Function Argument	Function	Description and Return Value
General	Format	Formats and then returns text, numbers, or dates in a specific pattern
	IsDate	Checks whether an expression can be converted to a date and returns true or false
	IsNull	Checks whether an expression contains Null (no data) and returns true or false
	IsNumeric	Checks whether an expression is a valid number and returns true or false
	Max	Returns the maximum value in a group of records
	Min	Returns the minimum value in a group of records

2-6 Summarizing with Aggregate Functions and Grouping

Some of the functions listed in Figure 2-20 are **aggregate functions** (Part of an expression used to calculate the number of entries, the sum or average of all the entries in a given column, or the largest or smallest of the entries in a given column; also called *function*.) , which make calculations on groups of records. Aggregate functions use a single field name as their only argument such as Sum([Salary]) or Count([LastName]). The following are common aggregate functions:

- Avg: Returns the average value of a numeric field in a group of records
- Count: Returns the number of records in a group
- Max: Returns the maximum field value in a group of records
- Min: Returns the minimum field value in a group of records
- Sum: Returns the summed total of a numeric field in a group of records

Before creating a query with an aggregate function, you should decide how you want to **group** (To create collections of records that share a common characteristic.) the records together. QBE programs use grouping to create groups of records that share some common characteristic. Without a **grouping field** (The field that divides records into groups based on the values in the specified field.) , *all* records are summarized for the calculation.

Secondly, decide which field contains the information you want to use for the calculation. Many of the aggregate functions such as Avg, Sum, StDev, and Var only make sense when applied to a field that contains numbers. Others such as Count, Max, and Min can be applied to any type of data.

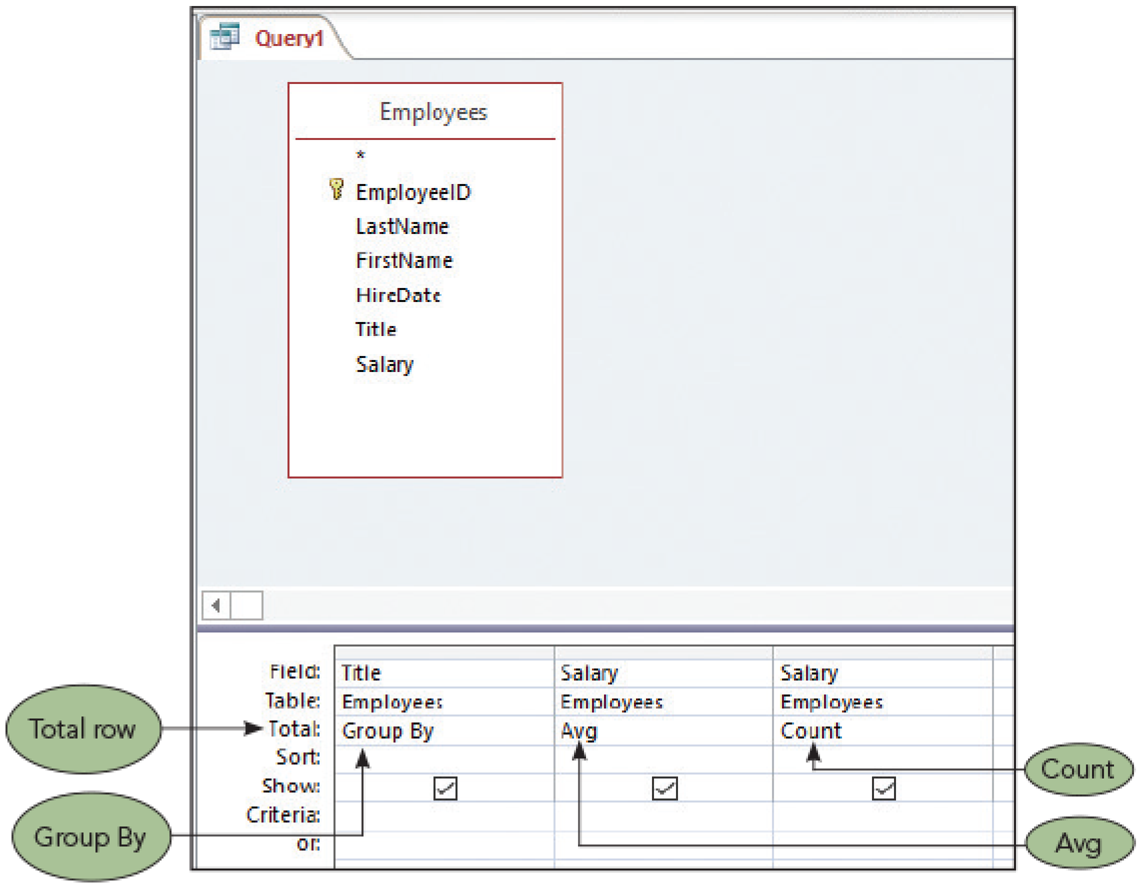
To use aggregate functions in an Access query, perform the following steps.

- Determine how you want to group the records.
- Determine which field or fields you want to use for the aggregate calculations.
- In Query Design View, add the tables containing the fields you want to use in the query, and then add those fields to the query grid.
- Click the Totals button (Query Tools Design tab | Show/Hide group). Access displays the Total row (see Figure 2-21) with Group By as the default value for each field.

- Change the Group By value to the appropriate aggregate function for the desired computation, and then run the query.

Figure 2-21

Query to Group, Average, and Count Records



Field:	Title	Salary	Salary
Table:	Employees	Employees	Employees
Total:	Group By	Avg	Count
Sort:			
Show:	✓	✓	✓
Criteria:			
or:			

You can add criteria to the query grid to further refine which records are selected for the query. If you remove the Group By field altogether, all records are grouped for the calculation.

Your Turn

2-6

What is the average salary value for each job title? Also include the total number of employees that share the same title.

To determine the average salary value as well as how many employees share the same job title, perform the following steps:

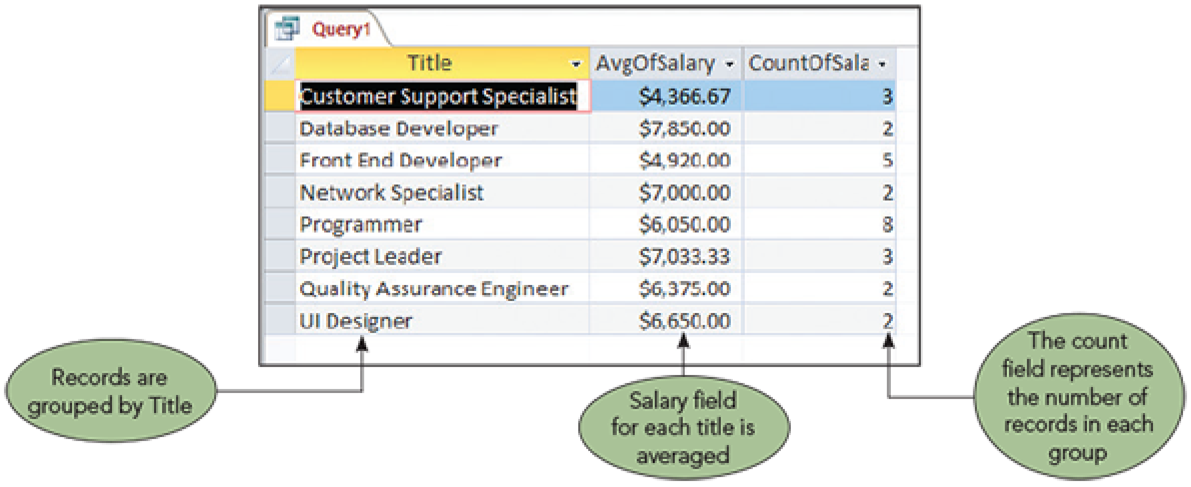
- In Query Design View, add the Title and Salary fields from the Employees table to the query grid. Add the Salary field a second time to the third column of the grid.

- Click the Totals button to add the Total row to the query grid.
- Click Group By for the first Salary field to display the list of aggregate operators, and then click Avg.
- Click Group By for the second Salary field to display the list of aggregate operators, and then click Count as shown in Figure 2-21.
- Run the query.

The query results appear in Figure 2-22. Access creates new default names for the columns with calculations, AvgOfSalary and CountOfSalary. You could rename those fields in Query Design View using *newfieldname:fieldname* syntax if desired such as *AverageSalary:Salary* or *Count:Salary.* It doesn't matter which field you count if all fields have a value, given the Count function returns the number of records in the group.

Figure 2-22

Grouped Query Results


Query1
Records are grouped by Title
Salary field for each title is averaged
The count field represents the number of records in each group

Title	AvgOfSalary	CountOfSala
Customer Support Specialist	$4,366.67	3
Database Developer	$7,850.00	2
Front End Developer	$4,920.00	5
Network Specialist	$7,000.00	2
Programmer	$6,050.00	8
Project Leader	$7,033.33	3
Quality Assurance Engineer	$6,375.00	2
UI Designer	$6,650.00	2

You also group records and use aggregate functions to find the minimum and maximum monthly salary for all employees and how many employees this represents.

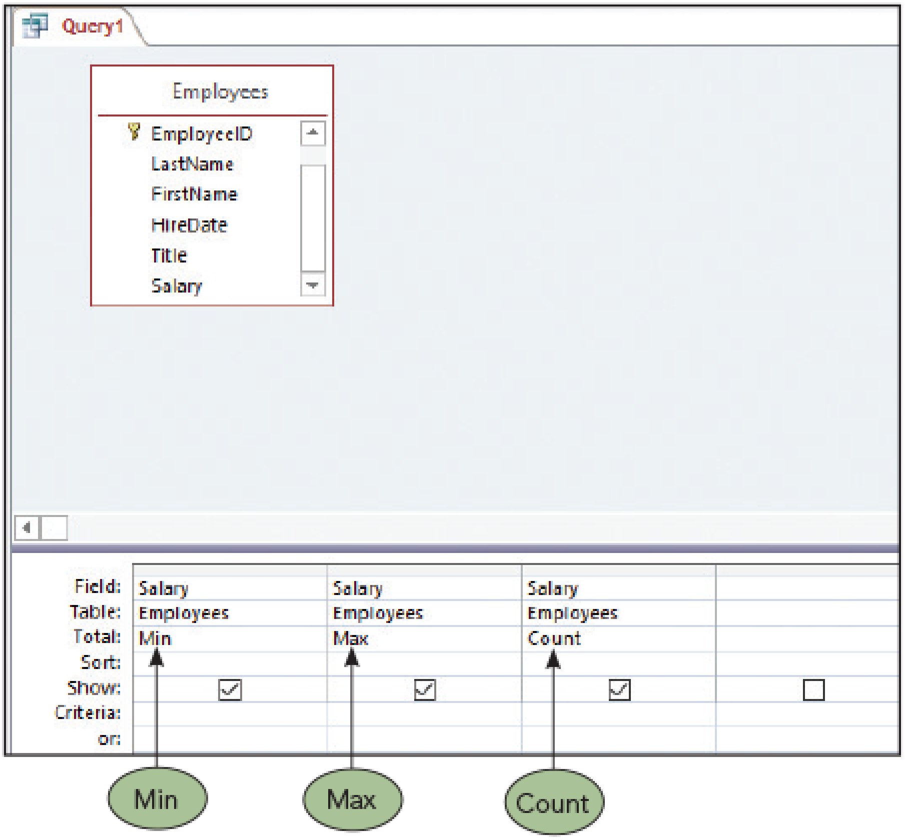
To make these calculations, perform the following steps:

- In Query Design View, add the Salary field from the Employees table to the query grid. Add the Salary field a second time to the second column of the grid and a third time to the third column of the grid.
- Click the Totals button to add the Total row to the query grid.
- Click Group By for the first Salary field to display the list of aggregate operators, and then click Min.
- Click Group By for the second Salary field to display the list of aggregate operators, and then click Max.

- Click Group By for the third Salary field to display the list of aggregate operators, and then click Count as shown in Figure 2-23.
- Run the query.

Figure 2-23

Query to Calculate the Minimum, Maximum, and Count of Salary

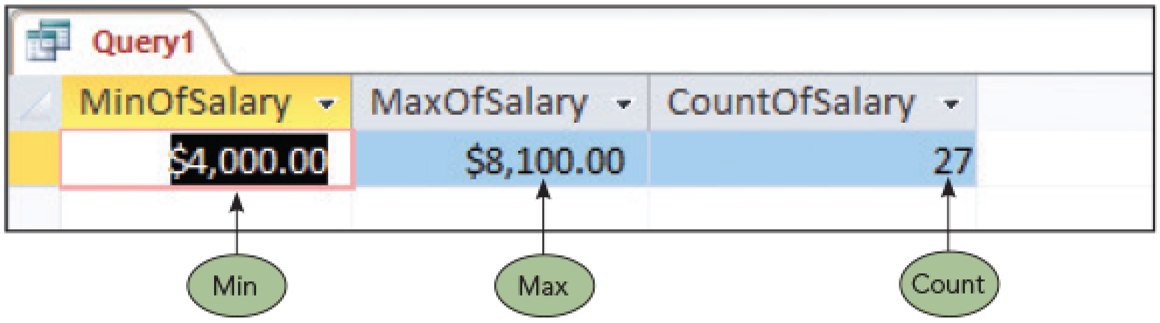


Field:	Salary	Salary	Salary	
Table:	Employees	Employees	Employees	
Total:	Min	Max	Count	
Sort:				
Show:	✓	✓	✓	☐
Criteria:				
or:				

The query results appear in Figure 2-24.

Figure 2-24

Grouped and Summarized Query Results



MinOfSalary	MaxOfSalary	CountOfSalary
$4,000.00	$8,100.00	27

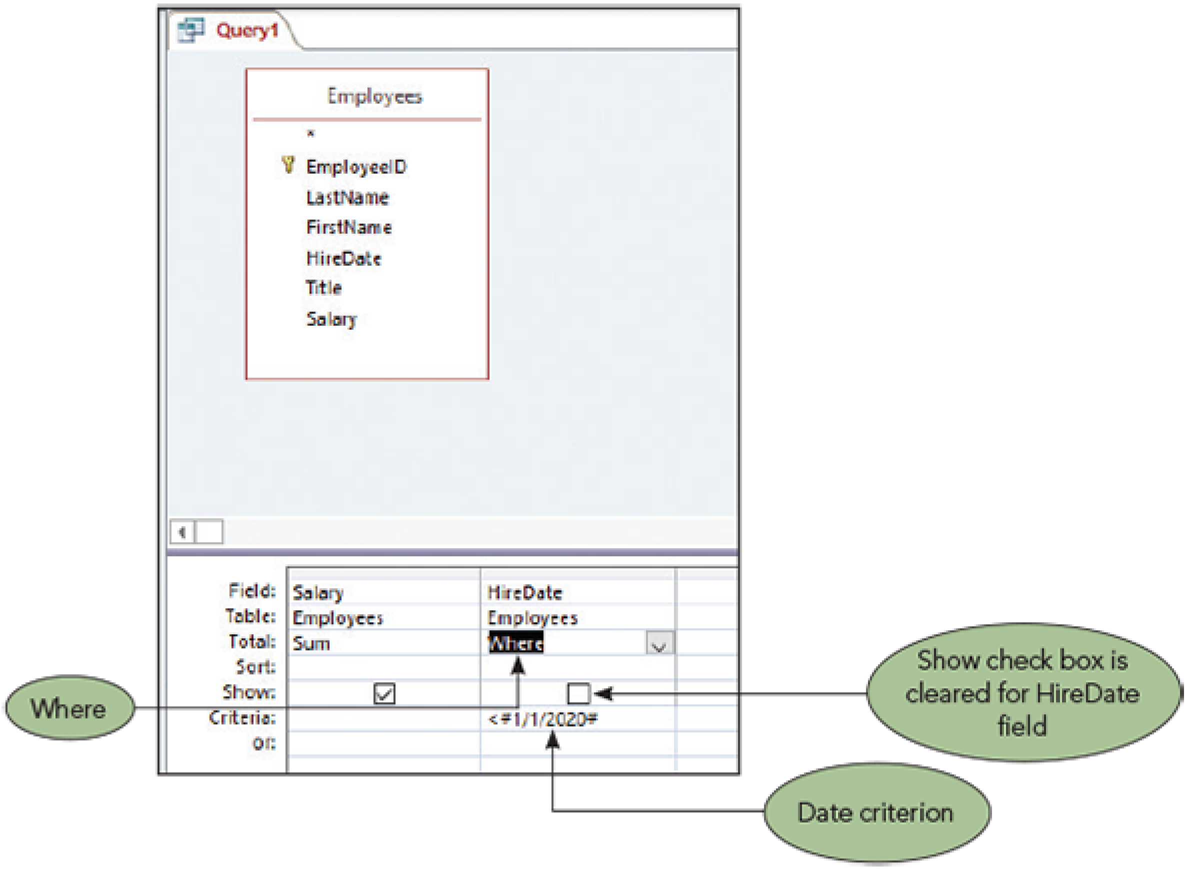
You can also add criteria to the query grid for queries that group and summarize data using the WHERE keyword.

Suppose you need to know the total monthly salary for all employees that were hired prior to 1/1/2020. Perform the following steps:

- In Query Design View, add the Salary field from the Employees table to the query grid. Add the HireDate field to the second column.
- Click the Totals button to add the Total row to the query grid.
- Click Group By for the Salary field to display the list of aggregate operators, and then click Sum.
- Click Group By for the HireDate field to display the list of aggregate operators, and then click Where.
- Click the Criteria cell for the HireDate field and enter <#1/1/2020# as shown in Figure 2-25.
- Remove the checkmark from the Show check box for the HireDate field. You can't both Sum the Salary field and show all of the dates prior to 1/1/2020 in Query Datasheet View, so the HireDate field is not displayed.
- Run the query.

Figure 2-25

Query to Group Records with Criteria

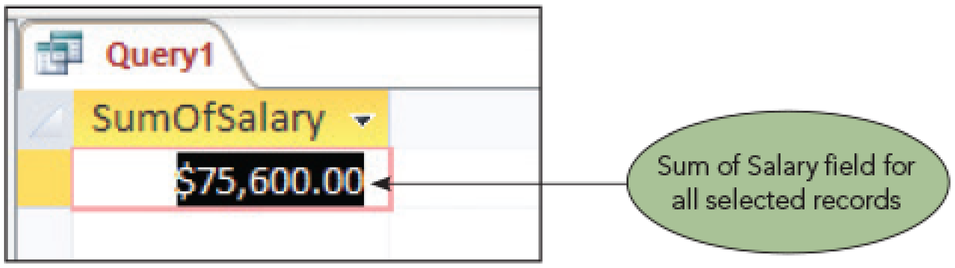


Field:	Salary	HireDate
Table:	Employees	Employees
Total:	Sum	Where
Sort:		
Show:	✓	☐
Criteria:		<#1/1/2020#
or:		

Date criteria is surrounded by # symbols (sometimes called the pound sign, hash mark, or octothorpe symbol) similarly to how text criteria is surrounded by "quotation marks". Access automatically adds the # symbols around date criteria if you do not enter those symbols yourself. The query results appear in Figure 2-26.

Figure 2-26

Grouped Results for Query Using a Criterion



Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-7 Sorting Records
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-7 Sorting Records

Specifying a **sort** (To arrange rows in a table or results of a query in a particular order.) order to determine the order of the records that are selected for a query can make the information easier to read. For example, you might want to see employees listed based on their hire date, clients listed alphabetically by client name, or tasks listed alphabetically by task name within a category.

The field on which records are sorted is called the **sort key** (The field on which records are sorted.) ; you can sort records using as many fields as desired. The first field used in the sort order determines the order of the records until two records have the same value in the first sort field. At that point, the second sort field takes over to determine the order of the records that have the same value in the first sort order, and so forth. To sort in Access, you specify the sort order (ascending or descending) in the Sort row of the design grid for the sort fields. Sort orders are evaluated left to right, so the leftmost sort order is the first sort order, and so forth.

Your Turn

2-7

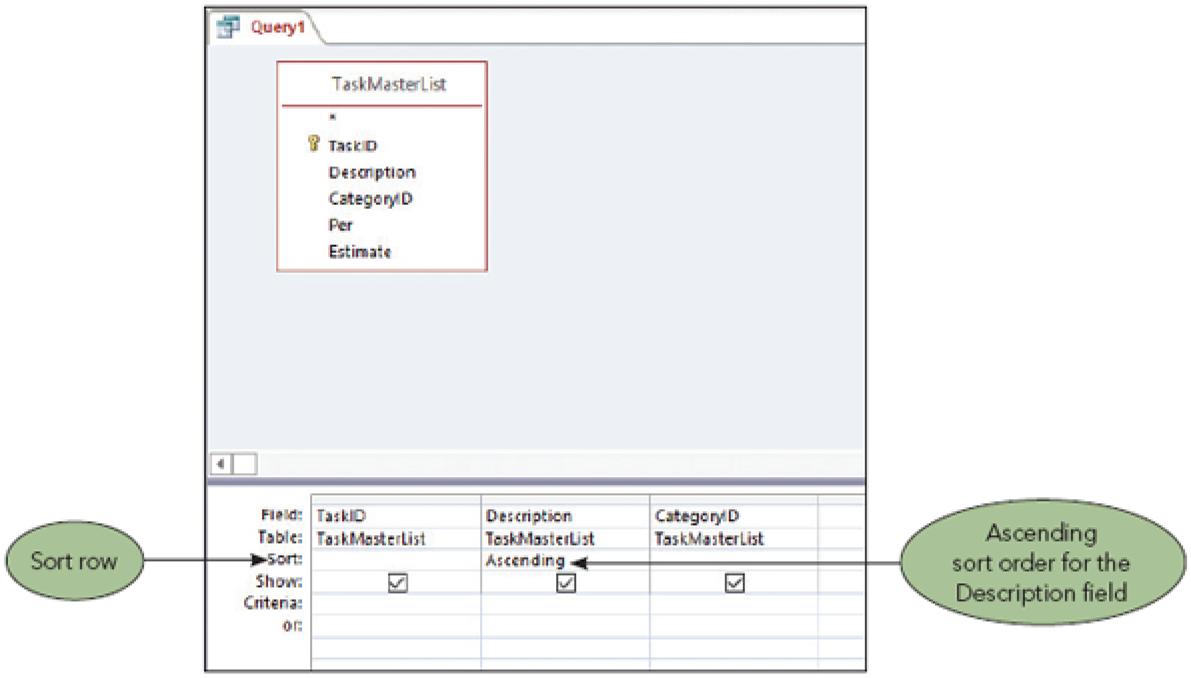
List the task ID, description, and category ID for each task in the TaskMasterList table. Sort the records in ascending order by description.

To sort the records alphabetically using the Description field, perform the following steps:

- In Query Design View, add the TaskID, Description, and CategoryID fields from the TaskMasterList table to the query grid in that order.
- Select the Ascending sort order in the Sort row for the Description column, as shown in Figure 2-27.
- You can click the Totals button (Query Tools Design tab | Show/Hide group) to turn off the total in the grid.

Figure 2-27

Query to Sort Records

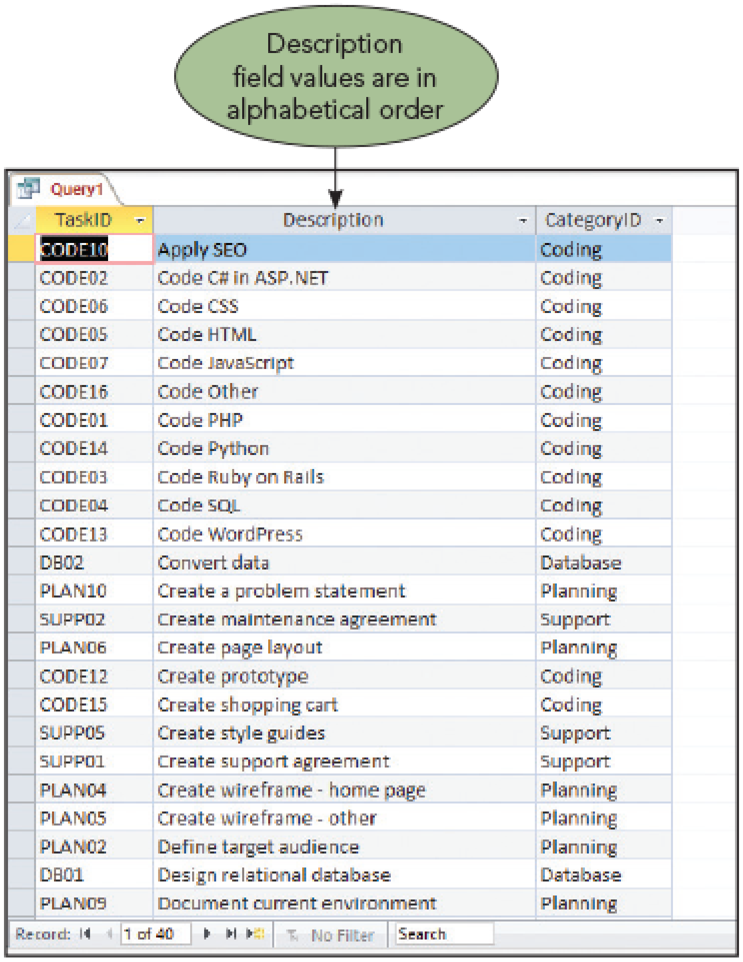


Field:	TaskID	Description	CategoryID
Table:	TaskMasterList	TaskMasterList	TaskMasterList
Sort:		Ascending	
Show:	☑	☑	☑
Criteria:			
or:			

The query results appear in Figure 2-28 with the task descriptions listed in alphabetical order.

Figure 2-28

Query Results with One Sort Key



TaskID	Description	CategoryID
CODE10	Apply SEO	Coding
CODE02	Code C# in ASP.NET	Coding
CODE06	Code CSS	Coding
CODE05	Code HTML	Coding
CODE07	Code JavaScript	Coding
CODE16	Code Other	Coding
CODE01	Code PHP	Coding
CODE14	Code Python	Coding
CODE03	Code Ruby on Rails	Coding
CODE04	Code SQL	Coding
CODE13	Code WordPress	Coding
DB02	Convert data	Database
PLAN10	Create a problem statement	Planning
SUPP02	Create maintenance agreement	Support
PLAN06	Create page layout	Planning
CODE12	Create prototype	Coding
CODE15	Create shopping cart	Coding
SUPP05	Create style guides	Support
SUPP01	Create support agreement	Support
PLAN04	Create wireframe - home page	Planning
PLAN05	Create wireframe - other	Planning
PLAN02	Define target audience	Planning
DB01	Design relational database	Database
PLAN09	Document current environment	Planning

2-7a Sorting on Multiple Keys

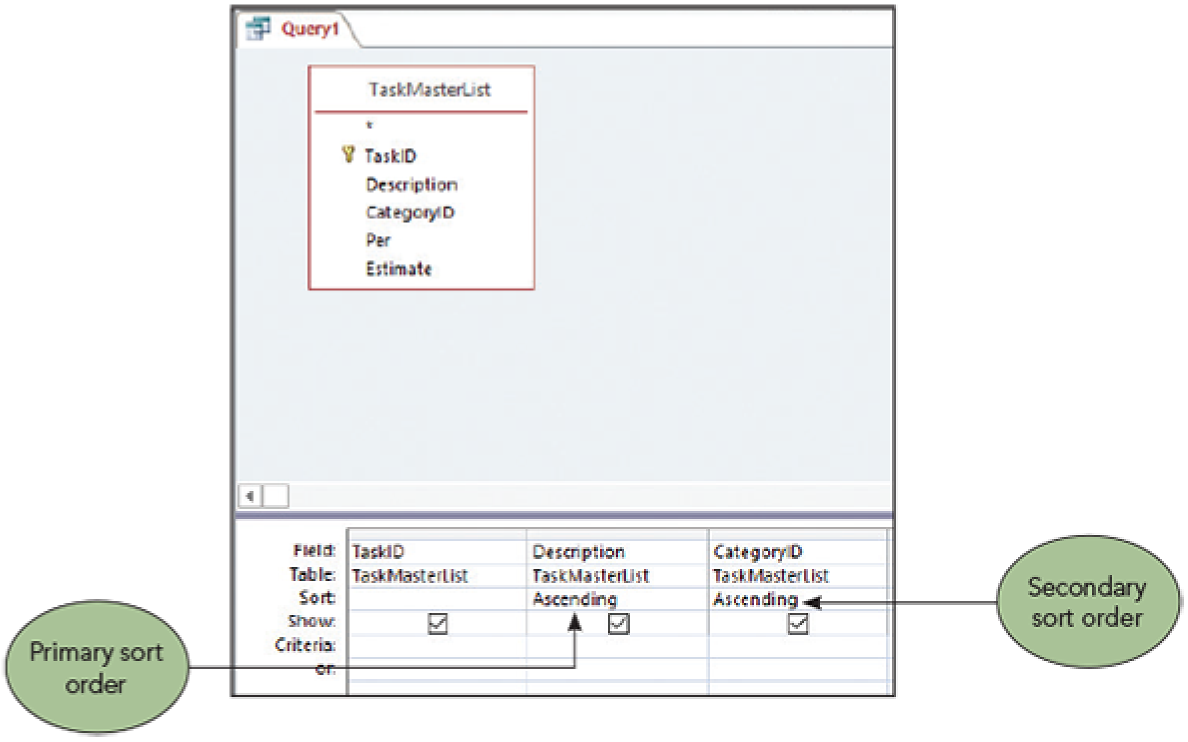
You can specify more than one sort key in a query. The leftmost sort order in the design grid is the **major** (When sorting on two fields, the more important field; also called a *primary* sort key.) **(primary) sort key** (When sorting on two fields, the more important field; also called a *major* sort key.) . Each additional sort key determines the order of the records when the previous sort field's values are the same.

For example, suppose you need to list the TaskID, Description, and CategoryID for each task in the TaskMasterList table, and sort the records in ascending order by task description within their category.

To sort records by description within category, the CategoryID should be the major sort key and Description should be the **minor** (When sorting on two fields, the less important field; also called *secondary* sort key.) **(secondary) sort key** (When sorting on two fields, the less important field; also called *minor* sort key.) . If you simply select the sort orders for these fields in the current design grid, your results would not be sorted correctly because the fields are listed in the wrong order left to right. Figure 2-29 shows an *incorrect* query design.

Figure 2-29

Sort Orders Work From Left to Right in Query Design View



Field:	TaskID	Description	CategoryID
Table:	TaskMasterList	TaskMasterList	TaskMasterList
Sort:		Ascending	Ascending
Show:	☑	☑	☑
Criteria:			
or:			

In Figure 2-29, the Description field is to the left of the CategoryID field in the design grid. With this order, Description becomes the major sort key; the data is sorted by description first, as shown in Figure 2-30. The minor sort key is not needed given there are never any values in the major sort key that are the same.

Figure 2-30

Results of Query with Incorrect Design for Sorting on Multiple Keys

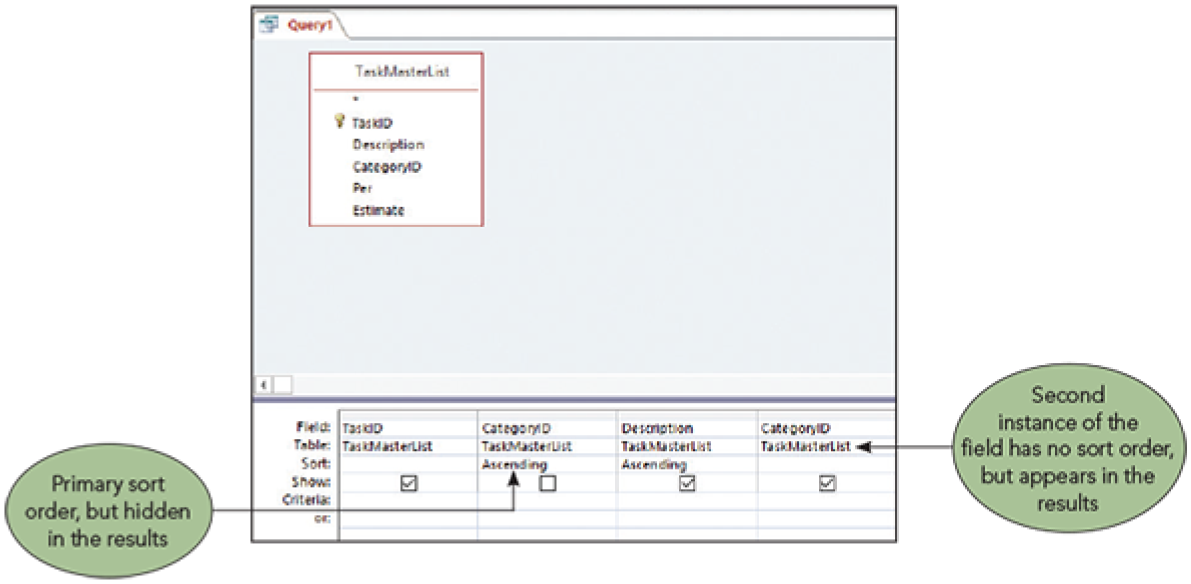
To correct this problem, the CategoryID field needs to move to the left of the Description field in the design grid. To move a field in the design grid, perform the following steps:

- Point to the top of the column.
- When the pointer changes to a down arrow, click to select the column.
- Drag the column to the new location.

Moving a column changes the output order, however, which you may not want to do. If the original order is important, you can include the secondary sort field twice—once before the primary sort field for sort order purposes, and once after for display purposes. You then can sort by the first occurrence of the field but hide it from the output, as shown in Figure 2-31. Notice the first occurrence contains the Ascending sort order but displays no checkmark in the Show check box. The second occurrence will be displayed, but it has no sort order selected.

Figure 2-31

Query Design to Sort by CategoryID Then by Description, but Display CategoryID After Description


Query1
TaskMasterList
*
TaskID
Description
CategoryID
Per
Estimate
Primary sort order, but hidden in the results
Second instance of the field has no sort order, but appears in the results

Field:	TaskID	CategoryID	Description	CategoryID
Table:	TaskMasterList	TaskMasterList	TaskMasterList	TaskMasterList
Sort:		Ascending	Ascending	
Show:	☑	☐	☑	☑
Criteria:				
or:				

In Figure 2-31, the CategoryID field is the major sort key, and the Description is the secondary, or minor, sort key given their left-to-right positions. The second CategoryID field in the design grid will display the category values in the query results in the desired position, as shown in Figure 2-32.

Figure 2-32

Results of Query with Multiple Sort Keys

2-8 Joining Tables

So far, the queries used in the examples have displayed records from a single table. In many cases, however, you will need to create queries to select data from more than one table at the same time. To do so, it is necessary to **join** (To connect two tables based on common data.) the tables based on a common field. You can join two tables in the upper pane of Query Design View, but it is more productive to join the tables ahead of time, in the Relationships window, so that they are automatically joined for every query that uses them.

To view the existing relationships in a database, perform the following steps:

- Click the Database Tools tab on the ribbon.
- Click the Relationships button (Database Tools tab | Relationships group).
- The relationships for the database appear in the Relationships window, but to be sure that none have been temporarily hidden, click the All Relationships button (Relationship Tools Design tab | Relationships group).

The Relationships window is where you add, delete, or modify relationships between tables. The one symbol on the **join line** (In an Access query, the line drawn between tables to indicate how they are related.) between the two tables identifies the field in the "one" table (often called the **parent table** (The table on the "one" side of a "one-to-many" relationship.)) of a one-to-many relationship, and the infinity symbol identifies the field in the "many" table (often called the **child table** (The table on the "many" side of a "one-to-many" relationship.)).

To delete an existing relationship between two tables, perform the following steps:

- View the existing relationships in the Relationships window.
- Right-click a join line, and then click Delete.

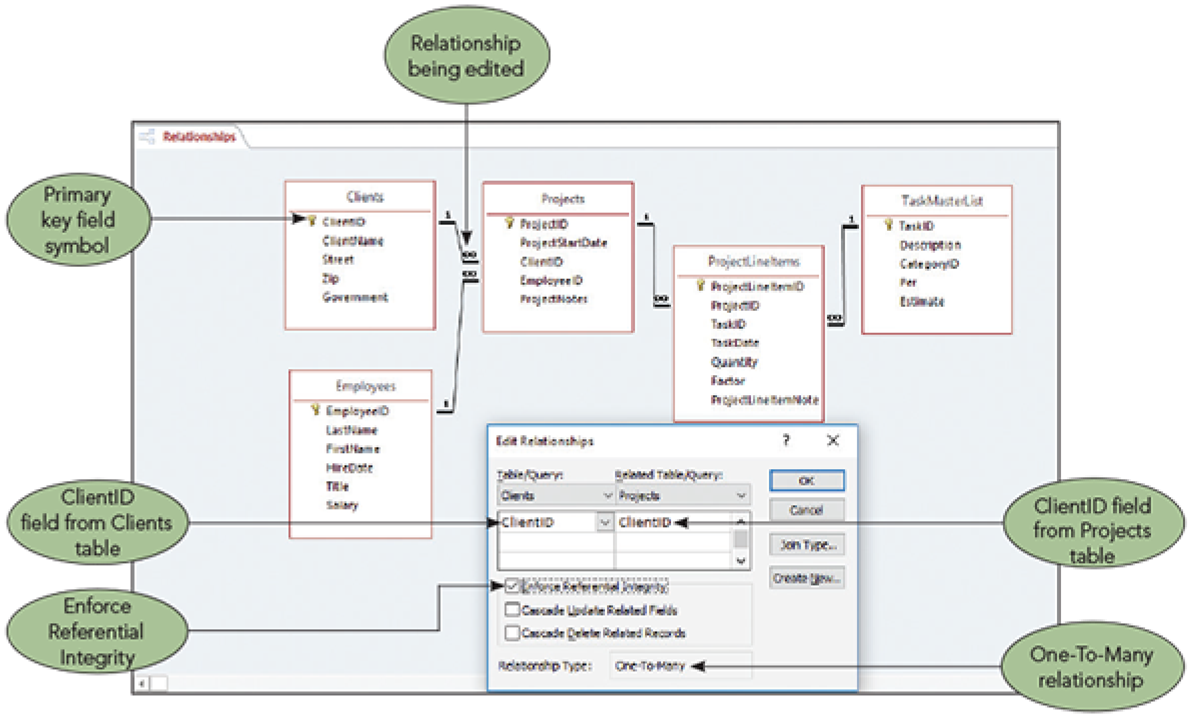
To edit an existing relationship between two tables, perform the following steps:

- View the existing relationships in the Relationships window.
- Right-click a join line, and then click Edit Relationship.

The Edit Relationships dialog box appears, as shown in Figure 2-33, identifying which fields are used in each table to create the relationship. The Edit Relationships dialog box also provides options to enforce referential integrity as well as other relationship options.

Figure 2-33

Editing the Relationship Between the Clients and Projects Table in the Relationships Window



Notice that the field on the “one” side of a one-to-many relationship is always a **primary key field** (A field that contains unique information for each record. A primary key field cannot contain a null entry.) as indicated by the **key symbol** (In Access, the symbol that identifies the primary key field in each table.) to the left of the field in the field list. The field on the “many” side of a one-to-many relationship is called the **foreign key field** (In a one-to-many relationship between two tables, the field in the “many” table that links the table to the primary key field in the “one” table.) .

Recall from Module 1 that **referential integrity** is a set of rules applied to the relationship that prevents the creation of orphan records. An **orphan record** is a record in the “many” (child) table that has no match in the “one” (parent) table. In the relationship shown in Figure 2-33, referential integrity prevents orphan records from being created in the Projects table. In other words, you cannot enter a record in the Projects table with a ClientID value that doesn’t already exist in the Clients table. You also cannot delete a record in the Clients table that has existing related records in the Projects table. When referential integrity is enforced on a one-to-many relationship, the link line displays “one” and “infinity” symbols to identify the “one” (parent) table and “many” (child) table.

To create a new existing relationship between two tables in the Relationships window, perform the following steps:

- View the existing relationships in the Relationships window.

- Drag the field used to create the relationship from the table on the “one” side of the relationship (which is always a primary key field) to the field used to create the relationship in the table on the “many” side of the relationship (which is never a primary key field. Rather, it is called the foreign key field).

  Technically, it doesn’t matter if you drag from the primary key field to the foreign key field or vice versa to make the relationship, but logically, it makes more sense to drag from the “one” (parent) table that contains the primary key field to the “many” (child) table that contains the foreign key field.

- The Edit Relationships dialog box opens, in which you can further modify the relationship by choosing options such as the Enforce Referential Integrity check box.

- Click OK in the Edit Relationships dialog box to complete the new relationship.

Note that creating relationships in the Relationships window requires that the table on the “one” side of the relationship have a primary key field, given the primary key field *always* becomes the linking field on the “one” side of a one-to-many relationship. If you do not see a key symbol to the left of a field in the field list, you first need to establish a primary key field in that table before it can participate on the “one” (parent) side of a one-to-many relationship.

To establish a primary key field for a table in the Relationships window, perform the following steps:

- Right-click the field list that represents the table for which you want to establish a primary key field.

- Click the field that you want to establish as the primary key field.

- Click the Primary Key button (Table Tools Design tab | Tools group).

In order to serve as a primary key field, a field must contain unique information in each record. If several records are entered into a table before the primary key field is established for the table, you may have to find and correct any duplicate entries in the field before it can be established as the primary key field, a process that is sometimes referred to as **scrubbing** (To remove and fix orphan records in a relational database.) the data. Some tables have natural primary key fields such as an employee ID or number, a student number, product ID, or an invoice number. Others, however, do not have an obvious candidate for the primary key field. In that case, an AutoNumber field, which automatically increments to the next integer value for each new record entered into the table, often serves as the primary key field for that table.

If table relationships have not been established in the Relationships window, you can still join tables in the upper pane of Query Design View. While not as productive as setting up the relationships ahead of time in the Relationships window, and also more computingintensive, you may occasionally need to establish a specific relationship for a specific query in Query Design View.

To join two tables in Query Design View, perform the following steps:

- Drag the field used to create the relationship from the table on the “one” side of the relationship to the field used to create the relationship in the table on the “many” side of the relationship.

  Technically, it doesn’t matter which field you start with to make the relationship, but logically, it makes more sense to drag from the “one” (parent) table to the “many” (child) table.

When you create a relationship between two tables in Query Design View, Access draws a join line between matching fields in the two tables, indicating that the tables are related. (If no relationships have been established in the Relationships window, Query Design View will try to automatically join two tables that have corresponding fields with the same field name if the field is a primary key field in one of the tables.)

A relationship created in Query Design View will not allow you to enforce referential integrity. A join line between two tables that does not have referential integrity is displayed as a thin line without the “one” and “infinity” symbols. Although the symbols are not displayed on the join line, the relationship is still a one-to-many relationship. Without enforcing referential integrity, however, orphan records may be easily created in the table on the “many” side of the relationship. More information on enforcing referential integrity as well as the other options in the Edit Relationships dialog box are covered in Module 4.

Once the tables are properly related in Query Design View, you can select fields from either or both tables for the query.

Your Turn

2-8

List each client ID and name, along with their related project IDs and project notes.

You cannot create this query using a single table—the client name is in the Clients table and the project information is in the Projects table. To select the fields needed for this query, perform the following steps:

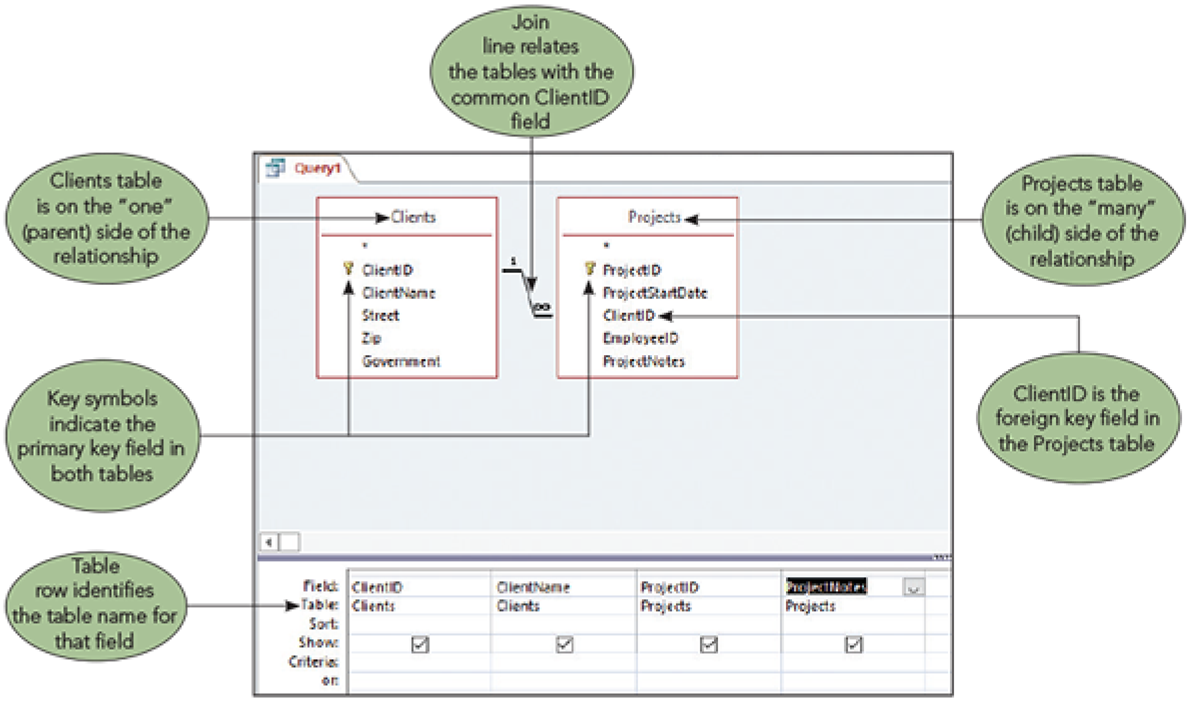
- In Query Design view, use the Show Table dialog box to add the Clients and Projects tables.

  A join line appears, indicating how the tables are related. Given the join line has the “one” and “infinity” symbols, you know that the relationship was previously created in the Relationships window and that referential integrity was enforced.

- Add the ClientID and ClientName fields from the Clients table to the design grid, and then add the ProjectID and ProjectNotes fields from the Projects table as shown in Figure 2-34.

Figure 2-34

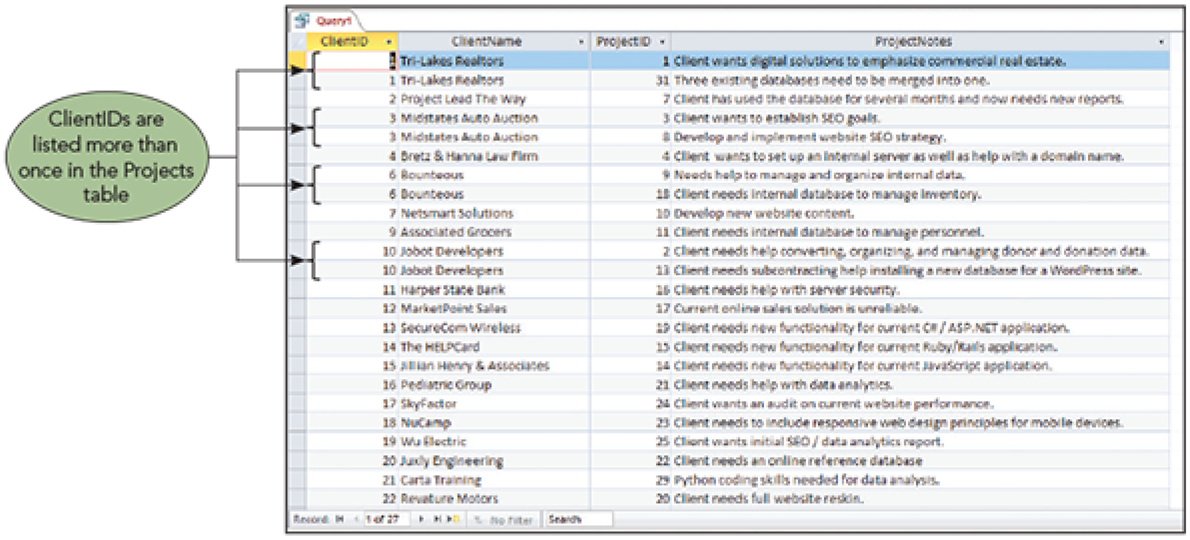
Query Design View with Related Tables


Join line relates the tables with the common ClientID field
Clients table is on the "one" (parent) side of the relationship
Projects table is on the "many" (child) side of the relationship
Key symbols indicate the primary key field in both tables
ClientID is the foreign key field in the Projects table
Table row identifies the table name for that field
Query1
Clients
ClientID
ClientName
Street
Zip
Government
Projects
ProjectID
ProjectStartDate
ClientID
EmployeeID
ProjectNotes
Field:
Table:
Sort:
Show:
Criteria:
or:
ClientID
Clients
ClientName
Clients
ProjectID
Projects
ProjectNotes
Projects

The Table row in the design grid indicates the table from which each field is selected. The query results appear in Figure 2-35. Notice that some clients are listed more than once. Those clients have more than one related record in the Projects table. Their ClientID value is listed more than once in the ClientID field of the Projects table, which links each record in the Projects table to the corresponding client in the Clients table.

Figure 2-35

Results of Query with Related Tables



ClientID	ClientName	ProjectID	ProjectNotes
1	Tri-Lakes Realtors	1	Client wants digital solutions to emphasize commercial real estate.
1	Tri-Lakes Realtors	31	Three existing databases need to be merged into one.
2	Project Lead The Way	7	Client has used the database for several months and now needs new reports.
3	Midstates Auto Auction	3	Client wants to establish SEO goals.
3	Midstates Auto Auction	8	Develop and implement website SEO strategy.
4	Bretz & Hanna Law Firm	4	Client wants to set up an internal server as well as help with a domain name.
6	Bounteous	9	Needs help to manage and organize internal data.
6	Bounteous	18	Client needs internal database to manage inventory.
7	Netsmart Solutions	10	Develop new website content.
9	Associated Grocers	11	Client needs internal database to manage personnel.
10	Jobot Developers	2	Client needs help converting, organizing, and managing donor and donation data.
10	Jobot Developers	13	Client needs subcontracting help installing a new database for a WordPress site.
11	Harper State Bank	16	Client needs help with server security.
12	MarketPoint Sales	17	Current online sales solution is unreliable.
13	SecureCom Wireless	19	Client needs new functionality for current C# / ASP.NET application.
14	The HELPCard	15	Client needs new functionality for current Ruby/Rails application.
15	Jillian Henry & Associates	14	Client needs new functionality for current JavaScript application.
16	Pediatric Group	21	Client needs help with data analytics.
17	SkyFactor	24	Client wants an audit on current website performance.
18	NuCamp	23	Client needs to include responsive web design principles for mobile devices.
19	Wu Electric	25	Client wants initial SEO / data analytics report.
20	Juxly Engineering	22	Client needs an online reference database
21	Carta Training	29	Python coding skills needed for data analysis.
22	Revature Motors	20	Client needs full website reskin.

For each project, suppose you need to list the project ID, project start date, employee ID, and last name for the employee assigned to that project. You must also select only those projects that start after 1/1/2021, and sort the records by the project start date within each employee's last name.

You cannot create this query using a single table—the client name is in the Clients table and the employee information is in the Employees table. To select the fields needed for this query, perform the following steps:

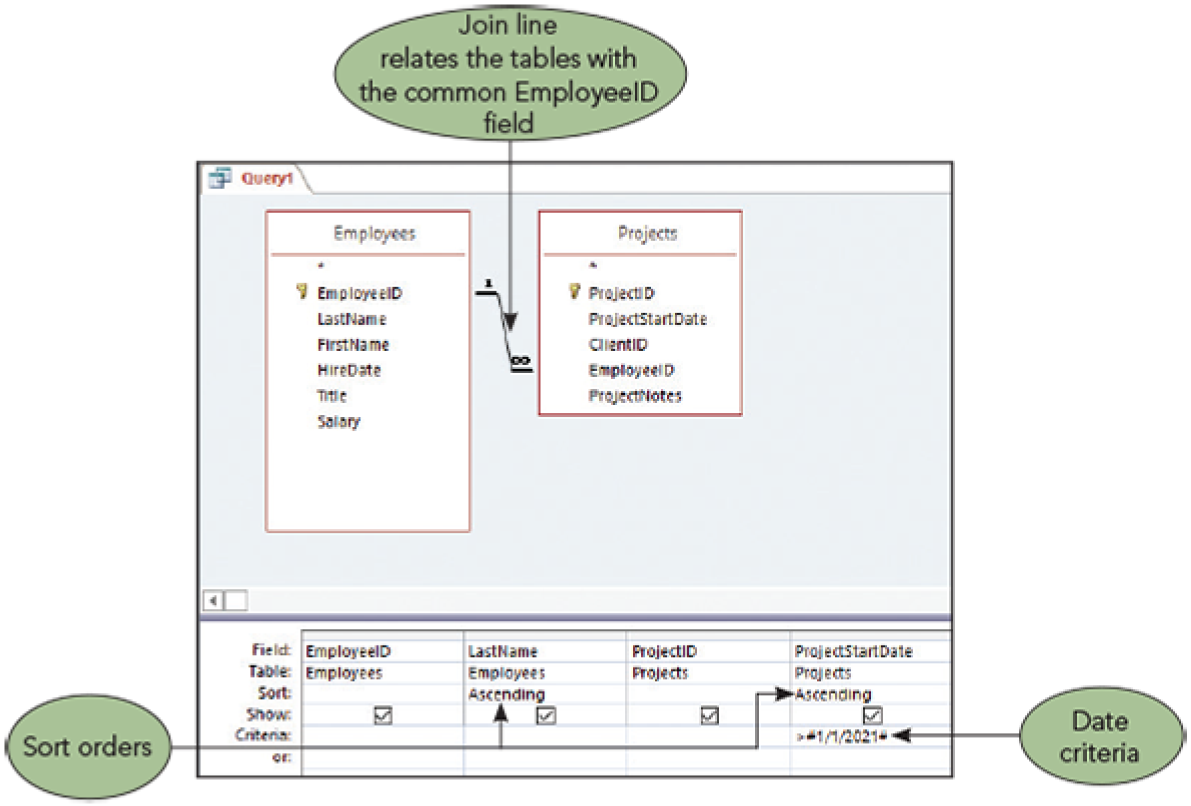
- In Query Design view, use the Show Table dialog box to add the Employees and Projects tables.

  A join line appears, indicating how the tables are related. Given the join line has the "one" and "infinity" symbols, you know that the relationship was previously created in the Relationships window and that referential integrity was enforced.

- Add the EmployeeID and LastName fields from the Employees table to the design grid, and then add the ProjectID and ProjectStartDate fields from the Projects table.
- Add an Ascending sort order to the LastName and ProjectStartDate fields.
- Add >#1/1/2021# to the Criteria row for the ProjectStartDate field as shown in Figure 2-36.

Figure 2-36

Query to Sort Joined Records



Field:	EmployeeID	LastName	ProjectID	ProjectStartDate
Table:	Employees	Employees	Projects	Projects
Sort:		Ascending		Ascending
Show:	☑	☑	☑	☑
Criteria:				>#1/1/2021#
or:				

Only records with a project start date after 1/1/2021 are selected. Records are sorted in alphabetical order on the LastName field, and records with the same LastName value are further sorted in ascending order on the ProjectStartDate field, as shown in Figure 2-37.

Figure 2-37

Results of Query with Joined and Sorted Records

2-8a Joining Multiple Tables

If you want to include fields from additional tables in your query, you can continue adding tables to the upper pane of Query Design View. All tables must be related to another table in a proper one-to-many relationship in order to select the desired records.

For example, suppose that for each client, you want to list the project ID, task ID, and task description for each project. You also want to sort the records by client name, then by project ID, and then by task description.

This query requires data from four tables: Clients, Projects, ProjectLineItems, and TaskMasterList. To only view the data, the ProjectID field can be selected from either the Projects or ProjectLineItems tables. The TaskID field can be selected from either the TaskMasterList or ProjectLineItems tables. Even if you do not use any fields from the ProjectLineItems table, however, you must include it in the upper pane of Query Design View so that the tables follow their relationships to select the correct data. Figure 2-38 shows the query design.

Figure 2-38

Query with Four Related Tables and Three Sort Orders

The query results appear in Figure 2-39.

Figure 2-39

Results of Query with Four Related Tables and Three Sort Orders

2-9 Using an Update Query

In addition to selecting fields and records from multiple tables, calculating new data for every record, and summarizing data for groups of records, you can use a query to modify data in a single process. Modifying several records in a single process is sometimes called a **batch update** (To modify several records in a single process.) because you are using one action to update data in many records, also called a batch, as opposed to a single record. A query that changes data in a batch process is called an **action query** (A query that changes data in a batch process.) . Examples include an update query, make table query, and delete query.

An **update query** (A query that makes a specified change to all records satisfying the criteria in the query.) makes a specified change to all records satisfying the criteria in the query. To change a query to an update query, click the Update button (Query Tools Design tab | Query Type group) in Query Design View. When you create an update query, a new row, called the Update To row, is added to the design grid. You use this row to indicate how to update the data selected by the query.

There is no undo feature to reverse the changes made by an update query. Therefore, it's a good practice to build your update query by first creating a select query and viewing the records that are selected to make sure you have selected the correct records. Once you are satisfied that you have selected the correct fields and records to update, change the query into an update query and run the update process.

Your Turn

2-9

Change employee records with the job title of "Project Leader" to "Project Manager."

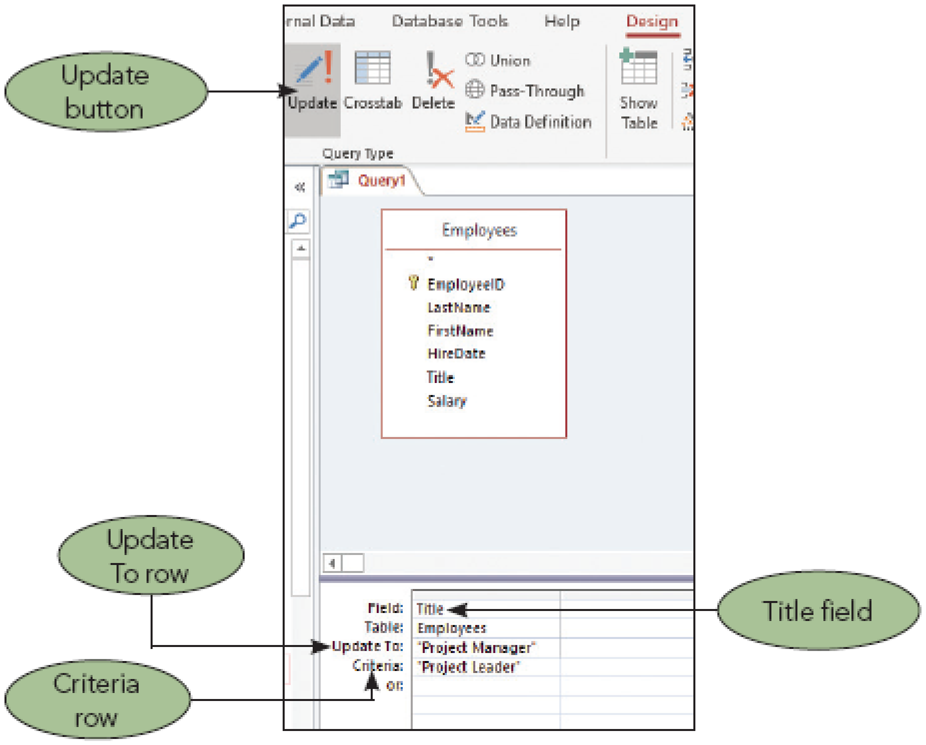
To update the employee records with a title of "Project Leader" to "Project Manager," perform the following steps:

- In Query Design View, add the Employees field list and drag the Title field to the query grid.
- Enter "Project Leader" in the Criteria cell, and then run the query to check the records that are selected. Three records should be selected.

- Return to Query Design View, and then click the Update button (Query Tools Design tab | Query Type group) to change the query from a select query to an update query.
- Enter “Project Manager” in the Update To cell for the Title field, as shown in Figure 2-40.
- Click the Run button (Query Tools Design tab | Results group). Access indicates how many rows the query will change and gives you a chance to cancel the update, if desired. Click Yes.

Figure 2-40

Query Design View for an Update Query



Field: Title
Table: Employees
Update To: "Project Manager"
Criteria: "Project Leader"
or:

When you click the Yes button, the query is executed and updates the data specified in the query design. Because the result of an update query is to change data in the records instead of selecting them to view, running the query does not produce a query datasheet.

2-10 Using a Delete Query

A **delete query** (A query that permanently deletes all the records satisfying the criteria entered in the query.) permanently deletes all the records satisfying the criteria entered in the query. For example, you might want to delete all the tasks associated with a certain project in ProjectLineItems table if the project has changed significantly and you want to start over with new tasks to create the project estimate.

Your Turn

2-10

Delete all ProjectLineItems records for the project with a project ID of 20.

To delete all ProjectLineItems records for the project with a project ID of 20, perform the following steps:

- In Query Design View, add the ProjectLineItems field list to the upper pane and drag the ProjectID field to the query grid.
- Enter 20 in the Criteria cell for the ProjectID field, click the Select button (Query Tools Design tab | Query Type group), and then run the query to check the records that are selected. Thirteen records should be selected.
- Return to Query Design View, and then click the Delete button (Query Tools Design tab | Query Type group) to change the query from a select query to a delete query.

  A new row, the Delete row, is added to the design grid as shown in Figure 2-41.
- Click the Run button. Access indicates how many rows will be deleted and gives you a chance to cancel the deletions, if desired. Click Yes.

Figure 2-41

Query Design View for a Delete Query

Running a delete query permanently deletes the records and cannot be undone.

If you run a delete query with no criteria, *all* records are selected. Running a delete query in that situation would delete all records from the table with fields in the query grid. Be careful!

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-11 Using a Make-Table Query
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-11 Using a Make-Table Query

A **make-table query** (A query that creates a new table in the current or another database with the records selected by the query.) creates a new table in the current or another database with the records selected by the query. The new table is a *copy* of the selected data, so make-table queries are only needed for specific reasons such as preparing data to be used by programs outside of the database or creating archived copies of data to capture information as of a particular point in time.

Your Turn

2-11

Create a new table containing the client name, as well as the project IDs and project start dates prior to 1/1/2020. Name the table ClientProjectsPriorTo2020.

To create a query that creates a new table with the ClientName, ProjectID, and ProjectStartDate fields prior to 1/1/2020, perform the following steps:

- In Query Design View, add the Clients and Projects field lists to the upper pane.
- Drag the ClientName field from the Clients table, and the ProjectID and ProjectStartDate fields from the Projects table to the grid.
- Enter <1/1/2020 in the Criteria cell for the ProjectStartDate field, click the Select button (Query Tools Design tab | Query Type group), and then run the query to check the records that are selected. Four records should be selected.
- Return to Query Design View, and then click the Make Table button (Query Tools Design tab | Query Type group) to change the query from a select query to a make-table query. The Make Table dialog box opens as shown in Figure 2-42.
- Enter the new table's name, ClientProjectsPriorTo2020 (Make Table dialog box).
- Click the OK button (Make Table dialog box).
- Click the Run button. Access indicates that four rows will be pasted into the new table and gives you a chance to cancel the process, if desired. Click Yes.

Figure 2-42

Query Design View for a Make-Table Query

After running the make-table query, the records are added to a new table named ClientProjectsPriorTo2020 in the current database. Figure 2-43 shows the new table created by the make-table query. In the figure, the columns in the table have been resized by dragging the right edge of the column heading. You can also specify sort orders within the make-table query if you want to order the records in a specific way in the new table.

Figure 2-43

ClientProjectsPriorTo2020 Table Created by the Make-Table Query

2-12 Optimizing Queries

A **query optimizer** (A DBMS component that analyzes a query and attempts to determine the most efficient way to execute it.) is a DBMS component that analyzes a query and attempts to determine the most efficient way to execute it. Generally, the query optimizer is a process that occurs behind the scenes without direct access to those who use the database.

In addition to using a query optimizer, you can use other techniques to increase the speed of your queries. The larger the database and the more simultaneous users, the more important these techniques become.

- Establish as many relationships in the Relationships window as possible.
- Enforce referential integrity on as many relationships as possible.
- In Query Design View, double-click fields to add them to the query grid and use list arrows to choose query options whenever possible to avoid typing errors.
- Include only the minimum number of fields necessary, especially if the query also groups records or uses aggregate functions.
- Delete any field lists that are not needed for the query or do not supply needed relationships between tables.
- Include meaningful criteria to select only those records needed for the question at hand.

Q&A

2-3

If your database is performing well, why is it still important to concern yourself with techniques that improve the performance of queries?

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-13 Examining Relational Algebra
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-13 Examining Relational Algebra

Relational algebra (A query language that takes instances of relations as input and yields instances of relations as output.) is a query language that takes instances of relations as input and yields instances of relations as output. Given the popularity of QBE approaches to relational database queries, relational algebra is rarely directly used with modern database management systems today. Still, relational algebra provides the foundation for **SQL** (See *Structured Query Language (SQL).*) , **Structured Query Language** (A very popular relational data definition and manipulation language that is used in many relational DBMSs.) , which is vitally important to programmers seeking to select, enter, update, and delete data stored in relational databases.

Figure 2-44 lists some of the common commands and operators used with relational algebra.

Figure 2-44

Relational Algebra Operators

Commands/Operators	Purpose
SELECT	Chooses a subset of rows that satisfies a condition
PROJECT	Reorders, selects, or deletes attributes during a query
JOIN	Compounds similar rows from two tables into single longer rows, as every row of the first table is joined to every row of the second table
UNION	Includes all rows from two tables, eliminating duplicates
INTERSECTION	Displays only rows that are common to two tables
SUBTRACT	Includes rows from one table that do not exist in another table (also called SET DIFFERENT, DIFFERENCE, or MINUS operation)
PRODUCT	Creates a table that has all the attributes of two tables including all rows from both tables (also referred to as the CARTESIAN PRODUCT)

Commands/Operators	Purpose
RENAME	Assigns a name to the results of queries for future reference

As in the following examples, each command ends with a GIVING clause, followed by a table name. This clause requests that the result of the command be placed in a temporary table with the specified name.

2-13a Selection

In relational algebra, the **SELECT** (SQL keyword to select fields and records from a relational database; also, the relational algebra command to select rows from a table.) command retrieves certain rows from an existing table (based on some user-specified criteria) and saves them as a new table. The SELECT command includes the word *WHERE* followed by a condition. The rows retrieved are the rows in which the condition is satisfied.

Your Turn

2-12

List all information about ClientID 19 from the Clients table.

This command creates a new table named Answer that contains only one row in which the ClientID value is 19, because that is the only row in which the condition is true. All the columns from the Clients table are included in the new Answer table.

To list all information from the Clients table about all clients with a Zip value of 02113, you would use the following command:

This command creates a new table named Answer that contains all the columns from the Clients table, but only those rows in which the Zip field value is ‘02113’. Note that a Zip field should be created as a text (string or character) field given the numbers do not represent quantities and should not be used in mathematical calculations. If the Zip field was created as a Number field, entries such as 02113 would be shortened to 2113 because leading zeros are insignificant to numbers, and are therefore deleted from the entry. As a text field, all characters are treated as significant pieces of information and saved with the entry.

2-13b Projection

In relational algebra, the **PROJECT** (The relational algebra command used to select columns from a table.) command takes a vertical subset of a table; that is, it causes only certain columns to be included in the new table. The PROJECT command includes the word *OVER* followed by a list of the columns to be included.

```
PROJECT Clients OVER (ClientID, ClientName) GIVING Answer
```

This command creates a new table named Answer that contains the ClientID and ClientName columns for all the rows in the Clients table.

Suppose you want to list the client ID and name of all clients with a Zip field value of 02447.

This example requires a two-step process. You first use a SELECT command to create a new table (named Temp) that contains only those clients with a Zip field value of 02447. Next, you project the new table to restrict the result to only the indicated columns.

```
SELECT Clients WHERE Zip='02447'
GIVING Temp
PROJECT Temp OVER (ClientID, ClientName)
GIVING Answer
```

Chapter 2: The Relational Model: Introduction, QBE, and Relational Algebra: 2-13c Joining
Book Title: Concepts of Database Management
Printed By: Michael Ammerman (mammerman0005@kctcs.edu)

2-13c Joining

The **JOIN** (The relational algebra command that connects two tables on the basis of common data.) command is a core operation of relational algebra because it allows you to extract data from more than one table. In the most common form of the join, two tables are combined based on the values in matching columns, creating a new table containing the columns in both tables. Rows in this new table are the **concatenation** (The combination of two or more rows in an operation, such as a join, or the combination of two or more columns for a primary key field to uniquely identify a given row in the table; also, joining two or more strings of textual data into one piece of data.) (combination) of a row from the first table and a row from the second table that match on the common column (often called the **join column** (The column on which two tables are joined. Also see *join*.)). In other words, two tables are joined *on* the join column.

For example, suppose you want to join the two tables shown in Figure 2-45 on the ClientID field (the join column), creating a new table named Temp.

Figure 2-45

Clients and Projects Tables

Clients

ClientID	ClientName	Street	Zip	Government
1	Tri-Lakes Realtors	135 E Jefferson St	02447	FALSE
2	Project Lead The Way	762 Saratoga Blvd	02446	TRUE
3	Midstates Auto Auction	9787 S Campbell Ln	01355	FALSE
4	Bretz & Hanna Law Firm	8101 N Olive Dr	01431	FALSE
5	Aspire Associates	5673 South Ave	01431	FALSE
6	Bounteous	9898 Ohio Ave	02770	FALSE
7	Netsmart Solutions	4091 Brentwood Ln	01354	FALSE
8	Loren Group	9565 Ridge Rd	02466	FALSE
9	Associated Grocers	231 Tecumsa Rd	02532	FALSE
10	Jobot Developers	1368 E 1000 St	02330	FALSE
11	Harper State Bank	1865 Forrest Dr	01571	FALSE
12	MarketPoint Sales	826 Hosta St	01983	FALSE

ClientID	ClientName	Street	Zip	Government
13	SecureCom Wireless	5280 Industrial Dr	01852	FALSE
14	The HELPCard	840 Boonville Ave	02466	TRUE
15	Jillian Henry & Associates	815 E California St	02113	FALSE
16	Pediatric Group	4940 W Farm Rd	02113	FALSE
17	SkyFactor	1736 Sunshine Dr	02726	FALSE
18	NuCamp	2500 E Kearny St	01431	FALSE
19	Wu Electric	5520 S Michigan	02447	FALSE
20	Juxly Engineering	4238 Rumsfield Rd	02148	FALSE
21	Carta Training	2445 N Airport Dr	02446	FALSE

Projects

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes
1	06-Feb-19	1	52	Client wants digital solutions to emphasize commercial real estate.
2	07-Feb-19	10	63	Client needs help converting, organizing, and managing donor and donation data.
3	11-Mar-19	3	52	Client wants to establish SEO goals.
4	10-Apr-20	4	52	Client wants to set up an internal server as well as help with a domain name.
7	02-Sep-19	2	63	Client has used the database for several months and now needs new reports.
8	06-Jan-20	3	52	Develop and implement website SEO strategy.
9	10-Feb-20	6	63	Needs help to manage and organize internal data.
10	31-Mar-21	7	19	Develop new website content.

ProjectID	ProjectStartDate	ClientID	EmployeeID	ProjectNotes
11	30-Apr-20	9	19	Client needs internal database to manage personnel.
13	30-Nov-20	10	64	Client needs subcontracting help installing a new database for a WordPress site.
14	09-Dec-20	15	19	Client needs new functionality for current JavaScript application.
15	21-Dec-20	14	19	Client needs new functionality for current Ruby/Rails application.
16	04-Jan-21	11	52	Client needs help with server security.
17	15-Feb-21	12	52	Current online sales solution is unreliable.
18	14-Apr-21	6	63	Client needs internal database to manage inventory.
19	04-Jun-21	13	52	Client needs new functionality for current C# / ASP.NET application.
20	30-Jul-21	22	63	Client needs full website reskin.
21	31-Aug-21	16	19	Client needs help with data analytics.
22	30-Sep-21	20	19	Client needs an online reference database
23	12-Nov-21	18	63	Client needs to include responsive web design principles for mobile devices.
24	14-Jan-22	17	63	Client wants an audit on current website performance.

The result of joining the Clients and Projects tables creates the table shown in Figure 2-46. The column that joins the tables (ClientID) appears only once. Other than that, all columns from both tables appear in the result.

Figure 2-46

Table Produced by Joining the Clients and Projects Tables

ClientID	ClientName	Street	Zip	Government	ProjectID	ProjectStartDate	EmployeeID	Project
1	Tri-Lakes Realtors	135 E Jefferson St	02447	FALSE	1	06-Feb-19	52	Client w digital solution emphas comme real est
1	Tri-Lakes Realtors	135 E Jefferson St	02447	FALSE	31	01-Mar-20	19	Three e databas need to merged one.
2	Project Lead The Way	762 Saratoga Blvd	02446	TRUE	7	02-Sep-19	63	Client h used th databas several months now ne new rep
3	Midstates Auto Auction	9787 S Campbell Ln	01355	FALSE	3	11-Mar-19	52	Client w to estab SEO go
3	Midstates Auto Auction	9787 S Campbell Ln	01355	FALSE	8	06-Jan-20	52	Develop implem website strategy
4	Bretz & Hanna Law Firm	8101 N Olive Dr	01431	FALSE	4	10-Apr-20	52	Client w to set u internal as well help wi domain
6	Bounteous	9898 Ohio Ave	02770	FALSE	9	10-Feb-20	63	Needs l manage organiz internal

ClientID	ClientName	Street	Zip	Government	ProjectID	ProjectStartDate	EmployeeID	Project
6	Bounteous	9898 Ohio Ave	02770	FALSE	18	14-Apr-21	63	Client n internal databas manage invento
7	Netsmart Solutions	4091 Brentwood Ln	01354	FALSE	10	31-Mar-21	19	Develo website content
9	Associated Grocers	231 Tecumsa Rd	02532	FALSE	11	30-Apr-20	19	Client n internal databas manage personn
10	Jobot Developers	1368 E 1000 St	02330	FALSE	2	07-Feb-19	63	Client n help convert organiz and ma donor a donatio
10	Jobot Developers	1368 E 1000 St	02330	FALSE	13	30-Nov-20	64	Client n subcon help ins a new databas WordPr site.
11	Harper State Bank	1865 Forrest Dr	01571	FALSE	16	04-Jan-21	52	Client n help wit server security
12	MarketPoint Sales	826 Hosta St	01983	FALSE	17	15-Feb-21	52	Current sales so is unrel
13	SecureCom Wireless	5280 Industrial Dr	01852	FALSE	19	04-Jun-21	52	Client n new functior for curr / ASP.N applicat

If a row in one table does not match any row in the other table, that row will not appear in the result of the join.

You can restrict the output from the join to include only certain columns by using the PROJECT command. For example, to list the ClientID, ClientName, ProjectID, and ProjectNotes for each client, use the following command:

```
JOIN Clients Projects
WHERE Clients.ClientID=Projects.ClientID
GIVING Temp
PROJECT Temp OVER (ClientID, ClientName, ProjectID, ProjectNotes)
GIVING Answer
```

In the WHERE clause of the JOIN command, the matching fields are both named ClientID—the field in the Projects table named ClientID is supposed to match the field in the Clients table named ClientID. Because two fields are named ClientID, you must qualify the field names. Just as in QBE, the ClientID field in the Projects table is written as Projects.ClientID and the ClientID field in the Clients table is written as Clients.ClientID.

In this example, the JOIN command joins the Clients and Projects tables to create a new table, named Temp. The PROJECT command creates a new table named Answer that contains all the rows from the Temp table, but only the ClientID, ClientName, ProjectID, and ProjectNotes columns.

The type of join used in the preceding example is called a **natural join** (The most common form of a join.) . Although this type of join is the most common, there is another possibility. The other type of join, the **outer join** (The form of a join in which all records appear, even if they don’t match.) , is similar to the natural join, except that it also includes records from each original table that are not common in both tables. In a natural join, these unmatched records do not appear in the new table. In the outer join, unmatched records are included and the values of the fields are vacant, or **null** (An intentional “nothing” value meaning “unknown” or “not applicable.”) , for the records that do not have data common in both tables. Performing an outer join for the preceding example would produce a table with all client records even if they did not have any related projects.

2-13d Union

The **union** (A combination of two tables consisting of all records that are in either table.) of tables A and B is a table containing all rows that are in either table A or table B or in both table A and table B. The union operation is performed by the **UNION** (The relational algebra command that combines two tables consisting of all records that are in either table.) command in relational algebra; however, there is a restriction on the operation. It does not make sense, for example, to talk about the union of the Clients table and the Projects table because the tables do not contain the same columns. The two tables *must* have the same structure for a union to be appropriate; the formal term is *union compatible.* Two tables are **union compatible** (Two tables are union compatible if they have the same number of fields and if their corresponding fields have identical data types.) when they have the same number of columns and when their corresponding columns represent the same type of data. For example, if the first column in table A contains ClientIDs, the first column in table B must also contain ClientIDs.

For example, suppose you need to list the employee ID and last name of those employees that are related to projects *or* have a title of Programmer or both.

You can create a table containing the employee ID and last and name of all employees that are related to projects by joining the Employees table and the Projects table (Temp1 in the following example) and then projecting the result over EmployeeID and LastName (Temp2). You can also create a table containing the EmployeeID and LastName of all employees with a Title value of Programmer by selecting them from the Employees table (Temp3) and then projecting the result (Temp4). The two tables ultimately created by this process (Temp2 and Temp4) have the same structure. They each have two fields: EmployeeID and LastName. Because these two tables are union compatible, it is appropriate to take the union of these two tables. This process is accomplished in relational algebra using the following code:

```
JOIN Employees, Projects
WHERE Employees.EmployeeID=Projects.EmployeeID
GIVING Temp1
PROJECT Temp1 OVER EmployeeID, LastName
GIVING Temp2
SELECT Employees WHERE Title='Programmer'
GIVING Temp3
PROJECT Temp3 OVER EmployeeID, LastName
GIVING Temp4
UNION Temp2 WITH Temp4 GIVING Answer
```

2-13e Intersection

The **intersection** (When comparing tables, an intersection is a new table containing all rows that are in both original tables.) of two tables is a table containing all rows that are *common* in both table A and table B. As you would expect, using the intersection operation also requires that the two tables have the same columns (also called union compatible) for the intersection to work. Compared to the UNION command, you replace the UNION command with the **INTERSECT** (The relational algebra command for performing the intersection of two tables.) command.

For example, suppose you want to list the employee ID and last name of employees that are related to projects *and* that have a title of Project Leader.

In this example, you need to join the Employees and Projects tables first to find which employees are related to projects. You then need to find the employees that have a title of Project Leader. Finally, you need to determine which records are common to both tables. The code to accomplish this is as follows:

```
JOIN Employees, Projects
WHERE Employees.EmployeeID=Projects.EmployeeID
GIVING Temp1
PROJECT Temp1 OVER EmployeeID, LastName
GIVING Temp2
SELECT Employees WHERE Title='Project Leader'
GIVING Temp3
PROJECT Temp3 OVER EmployeeID, LastName
GIVING Temp4
INTERSECT Temp2 WITH Temp4 GIVING Answer
```

2-13f Difference

The **difference** (When comparing tables, the set of all rows that are in the first table but that are not in the second table.) of two tables A and B (referred to as "A minus B") is the set of all rows in table A that are not in table B. As with intersection, the two tables must have the same columns, or be union compatible, for the difference to work. The difference operation is performed by the **SUBTRACT** (The relational algebra command for performing the difference of two tables.) command in relational algebra.

Suppose you want to list the employee ID and last name of those employees that are related to projects but that do *not* have a title of Project Leader.

This process is virtually identical to the one in the union and intersection examples, but in this case, you subtract one table from the other instead of taking their union or intersection. This process is accomplished in relational algebra using the following commands:

2-13g Product

The **product** (The table obtained by concatenating every row in the first table with every row in the second table.) of two tables (mathematically known as the **Cartesian product** (The table obtained by concatenating every row in the first table with every row in the second table.)) is the table obtained by combining every row in the first table with every row in the second table. If the first table has *m* rows and the second table has *n* rows, the product would have *m times n* rows. When creating a query in Query Design View, if you include two tables that are not joined directly or by intermediary table(s), the query completes a Cartesian join, and the resulting datasheet shows the number of records in the first table multiplied (joined to) each record of the second table. In some instances, Cartesian joins can be used to produce intentional results, but more often they are the result of tables not being properly joined in Query Design View.

2-13h Division

The **division** (The relational algebra command that combines tables and searches for rows in the first table that match all rows in the second table.) process also is used infrequently. It is best illustrated by considering the division of a table with two columns by a table with a single column. Consider the first two tables shown in Figure 2-47. The first table contains two columns: ProjectID and TaskID. The second table contains only a single column, TaskID.

Figure 2-47

Dividing One Table by Another

The quotient (the result of the division) is a new table with a single column named ProjectID (the column from the first table that is *not* in the second). The rows in this new table contain those ProjectIDs from the first table that have TaskIDs that "match" *all* the TaskIDs in the second table. Figure 2-47 shows the table that results from dividing the first table by the second.