

PETER ROB • CARLOS CORONEL • KEELEY CROCKETT



DATABASE SYSTEMS

DESIGN, IMPLEMENTATION & MANAGEMENT

INTERNATIONAL EDITION

MS Access Tutorial

A supplement to: *Database Systems: Design, Implementation and Management*
(International Edition)

Rob, Coronel & Crockett (ISBN: 9781844807321)

Table of Contents

Section	Title	Page
	Introduction	3
1.1	Create the Database	3
1.2	Creating the Table Structures	7
1.2.1	Data Entry	15
1.3	Importing Components from Other MS Access Databases	18
1.3.1	Editing the Imported Tables	22
1.4	Tracing a Transaction	27
1.5	Creating Relationships Among Tables	30
2.1	Queries	34
2.1.1	Editing the Query Output	42
2.1.2	Parameter Queries	47
2.1.3	Multiple Table Queries	52
2.1.4	Querying a Query	53
2.1.5	Update Queries	55
	Using Update Queries to Manage Transactions	62
2.1.6	Make Table Queries	66
2.1.7	Append Queries	71
3.1	Forms	77
3.1.1	Editing the Form	80
3.1.2	Forms Based on Queries	93
3.1.3	Forms with Subforms	96
3.1.4	Controlling Input with Combo Boxes	103
3.1.5	Controlling Input via List Boxes	106
3.1.6	Menus	108
4.1	Reports	118
4.1.1	Special Reports: Labels	124
5.1	Macro Groups and the Macros within Them	131
5.1.1	Attaching the Macros	133
5.1.2	A More Complex Set of Macros	134
5.1.3	Payment Options Organization	139
Conclusion		147

MS Access Tutorial

Introduction

After a database has been designed properly it must be implemented and the applications that make the database useful to the end users must be developed. Microsoft Access is a great tool for prototyping a database's implementation and its applications. Therefore, we will show you how to

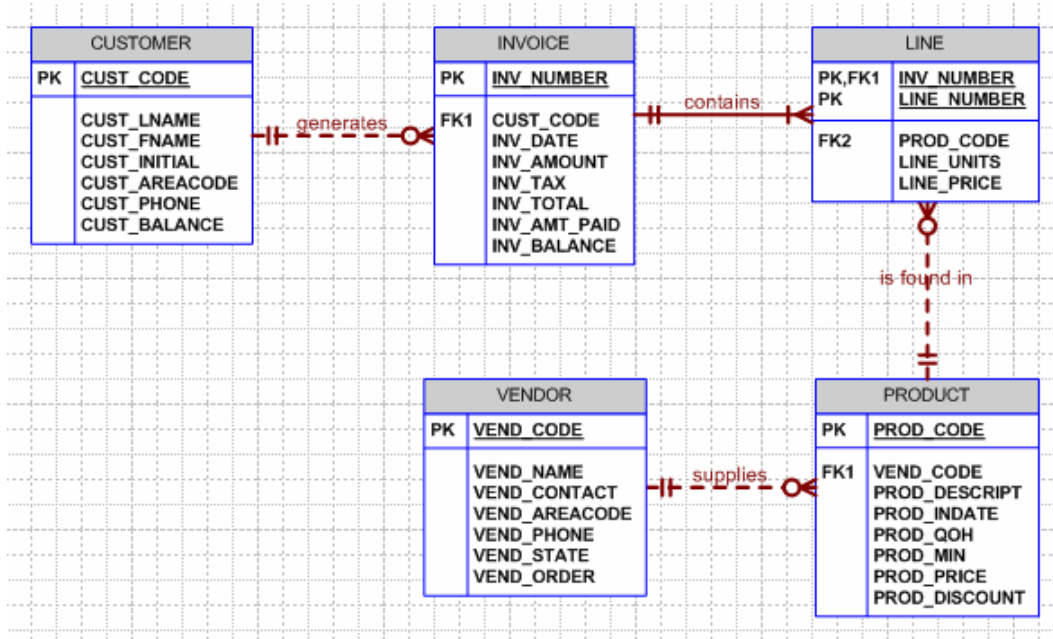
- Create the database.
- Create and modify tables.
- Enter data into the tables.
- Import tables.
- Define the relationship(s) between the tables.
- Create queries.
- Create forms.
- Create reports.
- Link application components with macros.

This tutorial assumes that you know how to perform basic operations in the Microsoft Windows environment. Therefore, you should know what a folder is, how to maximize or minimize a folder, how to create a folder, how to select a file, how you maximize and minimize windows, what clicking and double-clicking indicate, how you create a folder, how you drag, how to use drag and drop, how you save a file, and so on.

1.1 Create the Database

If you have studied our **Database Systems: Design, Implementation, and Management** text, you already know the important ground rule: The database design is the crucial first step in the journey that lets you successfully implement and manage a database. The database design shown in Figure 1 in the form of an Entity Relationship Diagram (ERD) will be the basis for this tutorial.

Figure 1 The SaleCo Database ERD

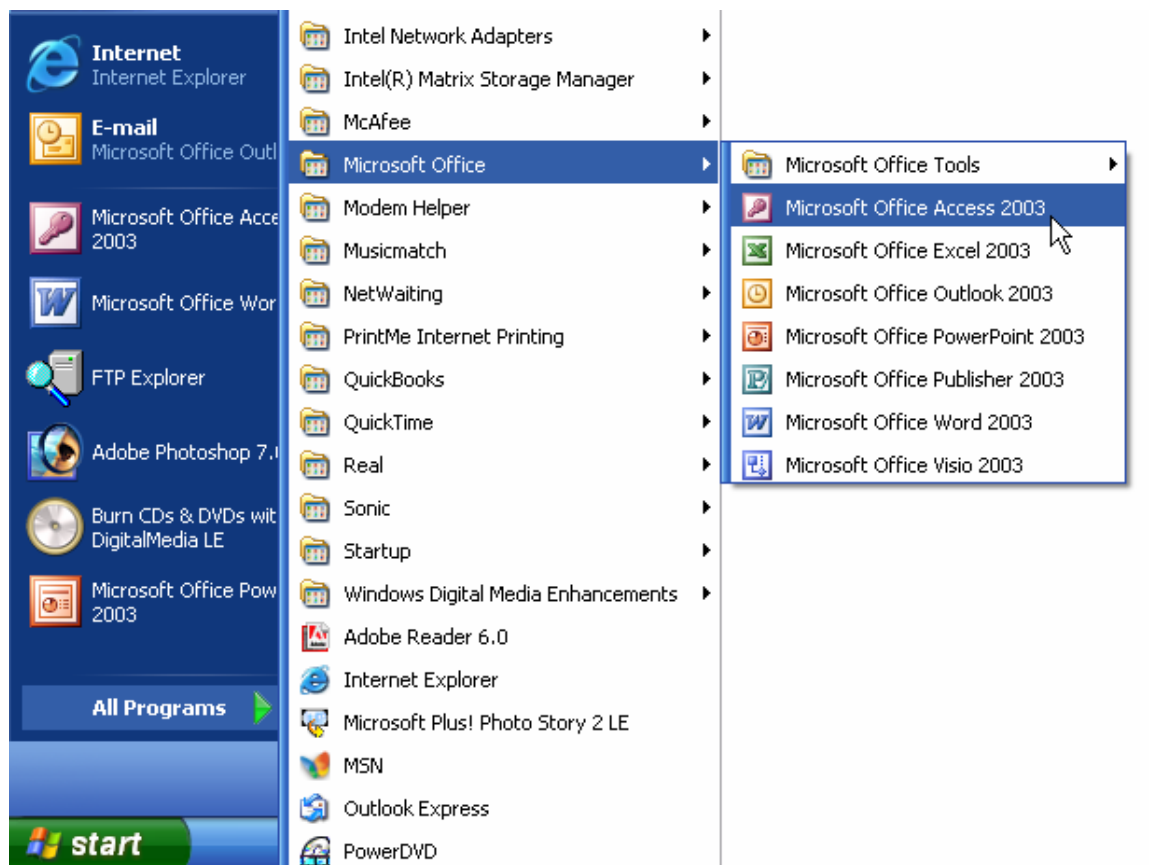


NOTE

If necessary, review the text’s Chapters 4 through 6 to ensure that you have a sound basis for database design work. You can study Appendix A, “Designing Databases with Visio Professional: A Tutorial,” if you want to create the database design shown in Figure 1.

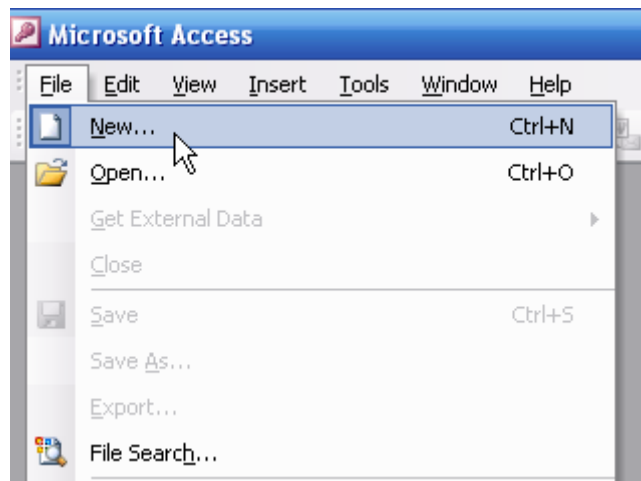
Given the database design shown in Figure 1, you are ready to implement it in Microsoft Access. The first step is to start the MS Access DBMS software, which is part of the Microsoft Office *Professional* suite. Use the same routine you use in all Microsoft office components. In other words, to start Microsoft Access, follow the sequence **Start/All Programs/Microsoft Office/Microsoft Access**. The *sequence* is the same for all MS Office versions. However, you will see slight variations in the labeling of the MS Office components. For example, Figure 2 shows Office 2003 labels. (And, naturally, your screen is likely to differ -- in detail -- from the one you see in Figure 2. After all, you may be running a different version of Windows and you may have a different version of the Windows Office suite. Nevertheless, the basics remain the same.)

Figure 2 Starting Microsoft Access



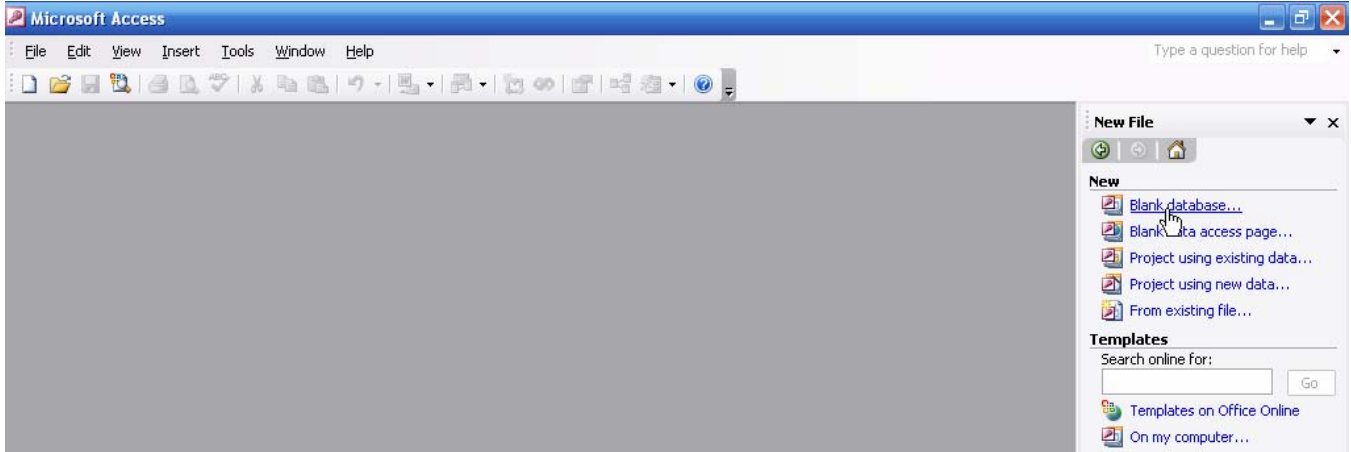
After you have completed the just-described sequence, you will see the screen in Figure 3. (Only a small portion of the screen is shown.)

Figure 3 Create a New Database



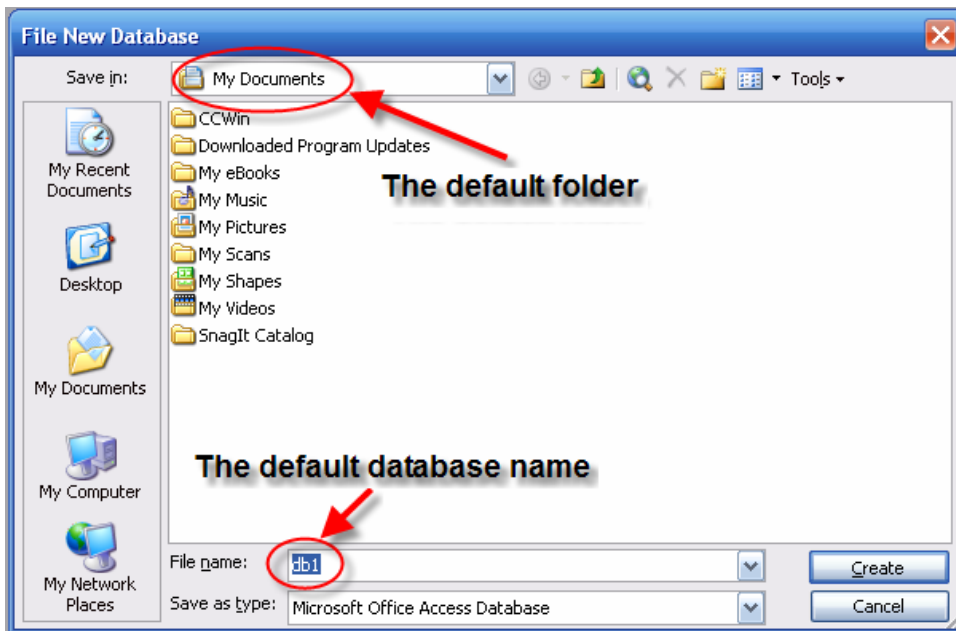
After you select the [New...](#) option shown in Figure 3 the screen will change to show Figure 4. Place the cursor over the [Blank database...](#) option. (Note that the cursor changes to the shape of a hand.)

Figure 4 Selecting a Blank Template



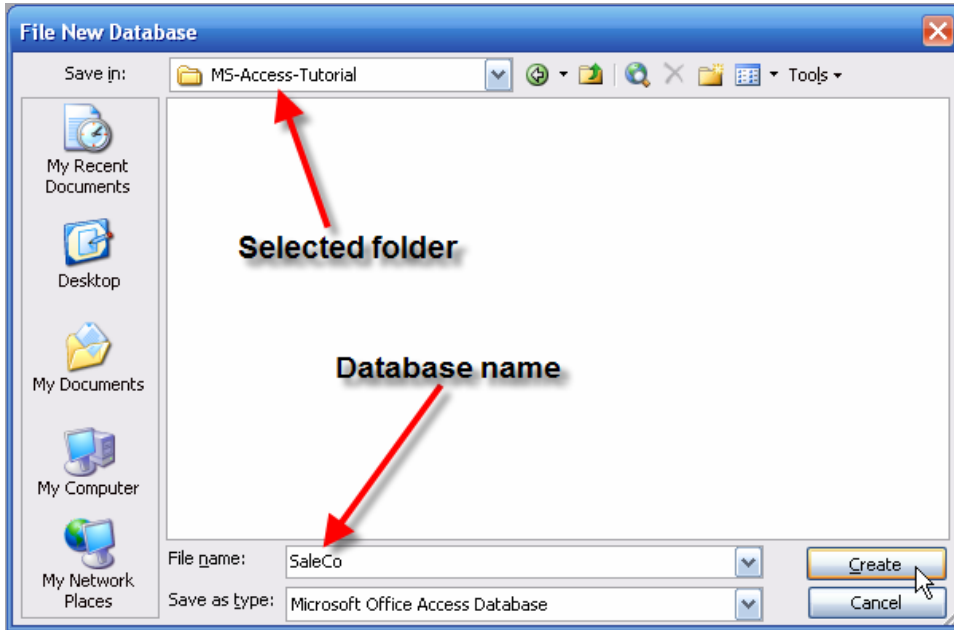
When you select the [Blank database...](#) option shown in Figure 4, Figure 5 will appear. Note that the default folder is **My Documents** and that the default database name is **db1**.

Figure 5 Default Folder and Database Name



Select the folder in which you want to store the database. In the example you see in Figure 6, the folder is named **MS-Access-Tutorial** and the database name is **SaleCo**.

Figure 6 Select the Database Folder and Name

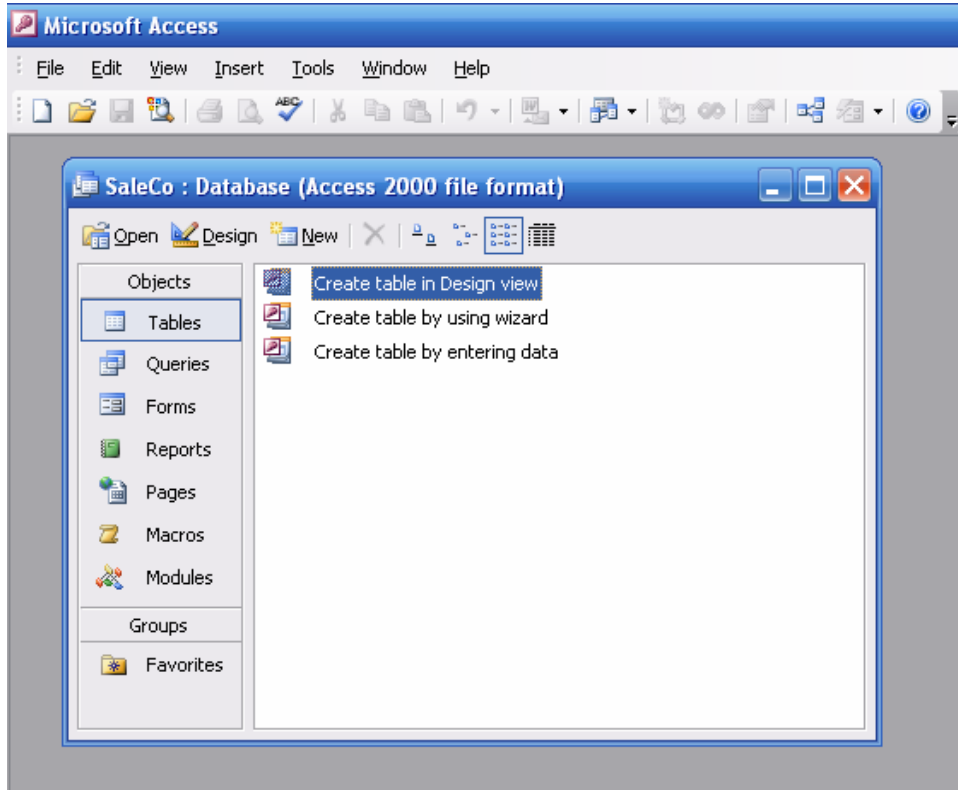


Note

The **SaleCo** database is also found on the Course Technology website for the text. However, *that* database includes all the components you will learn about in this tutorial. If you want to copy the **SaleCo** database from the website, make sure you place it in a different folder to avoid over-writing the database components that you will create in this tutorial.

After selecting the folder and database name as shown in Figure 6, click on the **Create** button to generate the **SaleCo** database you see in Figure 7.

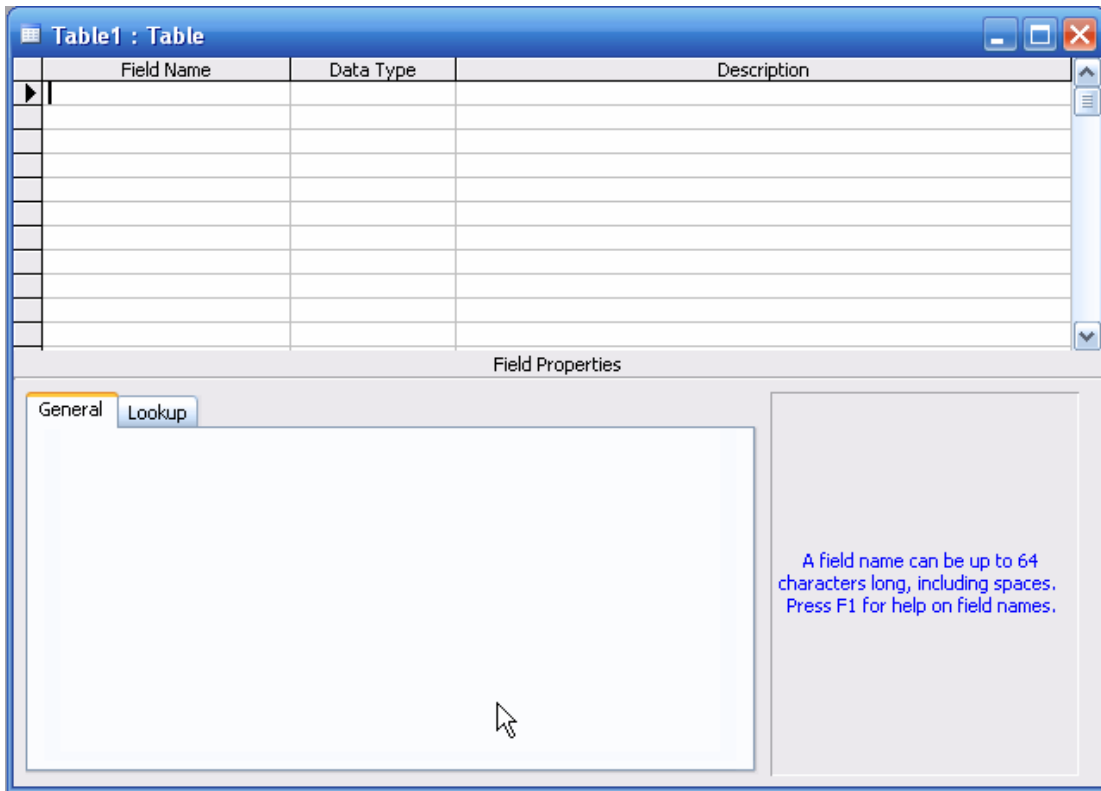
Figure 7 The SaleCo Database



1.2 Creating the Table Structures

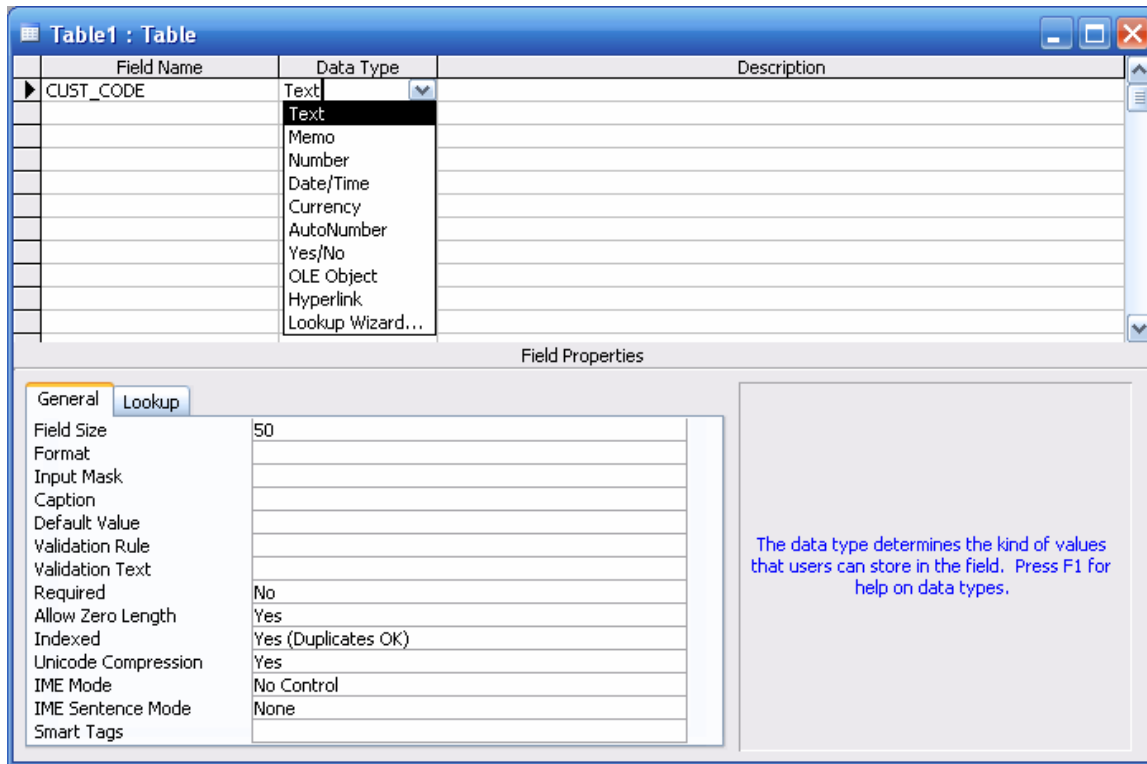
As you examine Figure 7, note that the default **Objects** selection is **Tables** and that the default table format is **Create table in Design view**. To create the first table structure, double-click on the **Create table in Design view** option or click on the **Design** option you see in the command bar to generate the screen you see in Figure 8. (You can click on the maximize button to maximize the screen or you can simply drag the sides to suit your needs.)

Figure 8 The Initial Table in Design Format



You are now ready to type in the first attribute name. (See Figure 1 again to see what attributes are included in the CUSTOMER table.) Figure 9 shows the entry of the CUST_CODE attribute.

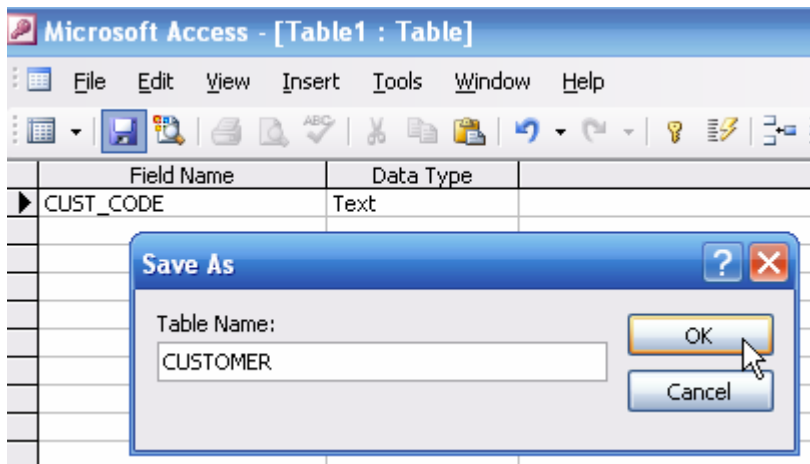
Figure 9 The Initial CUSTOMER Table Field Entry



The CUST_CODE values will all consist of a five-character text string, so the **Field Properties** box default **Field Size** of 50 should be reset to 5. (Just mark the “50” and type in the value “5” to get the job done.)

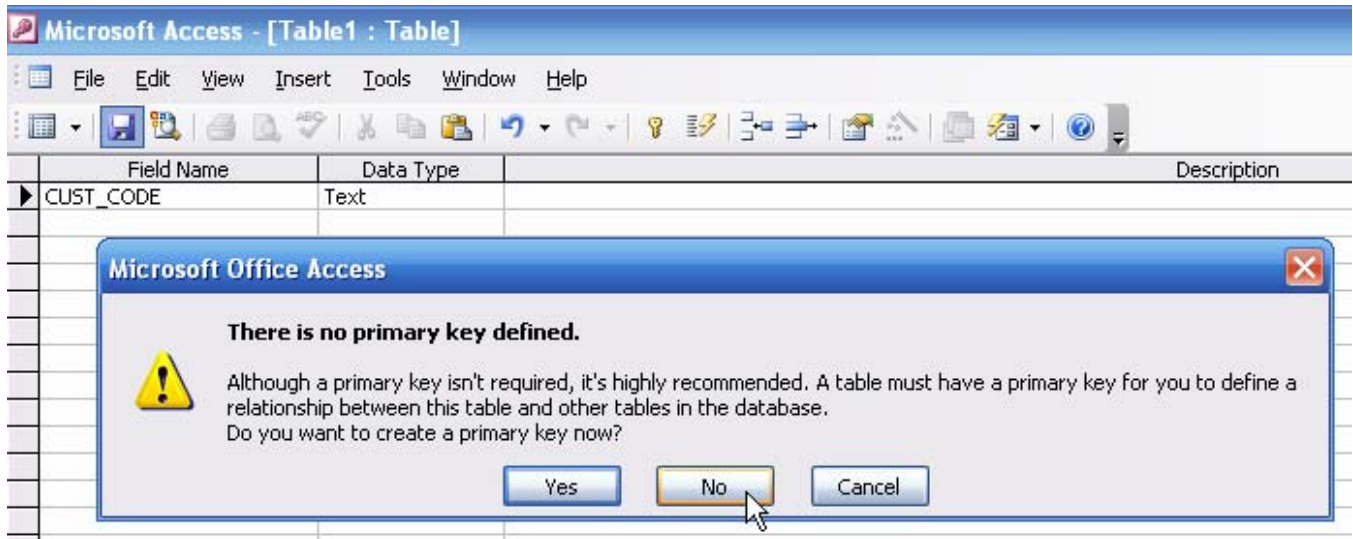
We suggest that you save your work frequently, so go ahead and save the CUSTOMER table structure. You will be prompted to enter a table name, so type the CUSTOMER name in the **Save As** dialog box, as shown in Figure 10.

Figure 10 Save the CUSTOMER Table



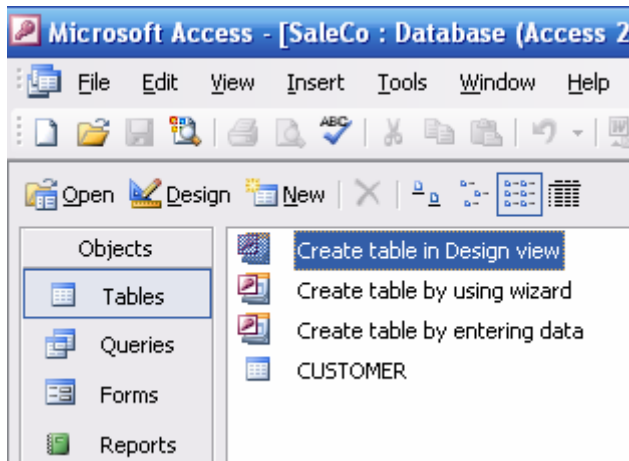
However, note that your save routine pops up the message box shown in Figure 11.

Figure 11 Primary Key Reminder



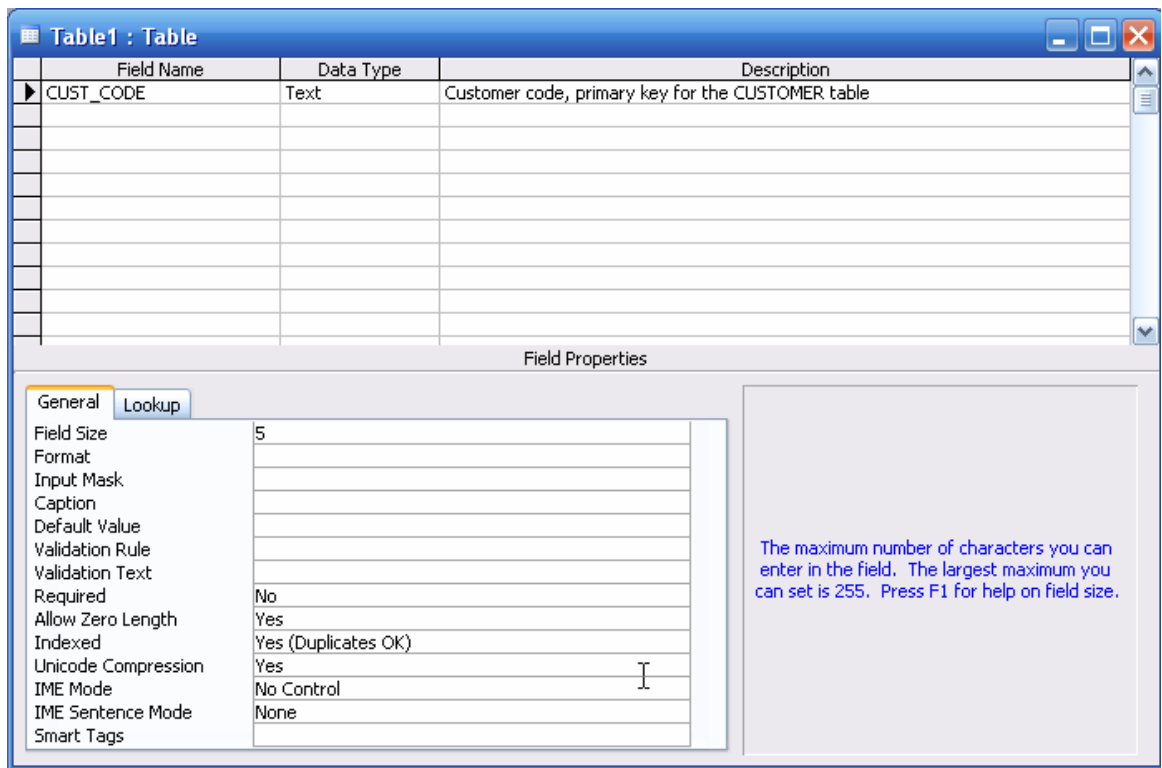
If you select the **Yes** button in the PK reminder shown in Figure 11, MS Access will automatically create a PK attribute for you ... and that is not what you want. (Remember, you want the CUST_CODE to be your PK.) If you select **Cancel**, you will be returned to the table design format without first saving the table. Since you want to save the table, select **No** to ensure that the table will be saved, albeit without a PK. (We will show you how to set the PK later.) After you click on the **No** button, MS Access will return you to the screen you first saw in Figure 7, but the new CUSTOMER table will be listed as shown in Figure 12.

Figure 12 The Saved CUSTOMER Table



To continue working on the CUSTOMER table, click on the CUSTOMER table name and then select the **Design** option to return to the table field entry screen. If you take a look at Figure 13, you'll see that we have entered a CUST_CODE description and that we have decreased the **Field Size** to 5.

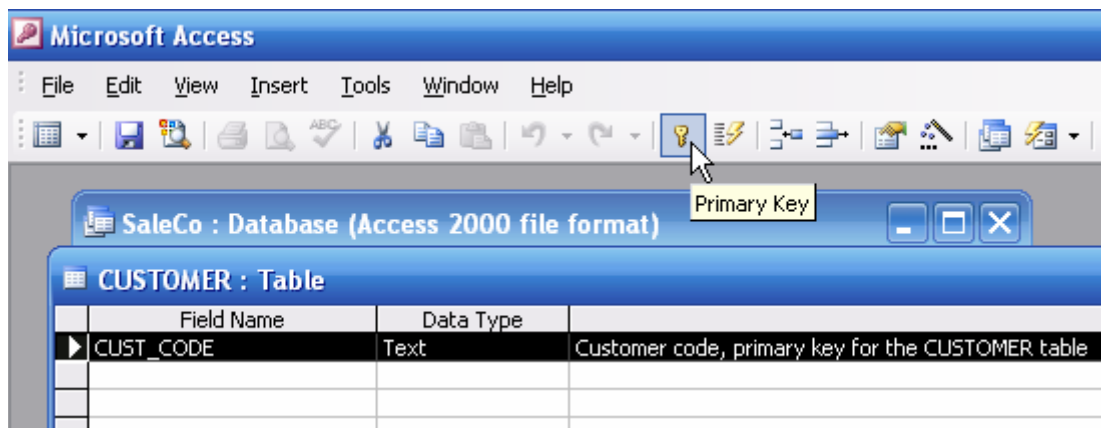
Figure 13 Additional CUSTOMER Table Field Entries



As you examine Figure 13, note that the **Field Properties** box entry for the CUST_CODE indicate that the **Indexed** default value is set to **Yes (Duplicates OK)**. Clearly, that is not appropriate if the CUST_CODE is to be the PK, so let's set the PK. That job is done in two steps:

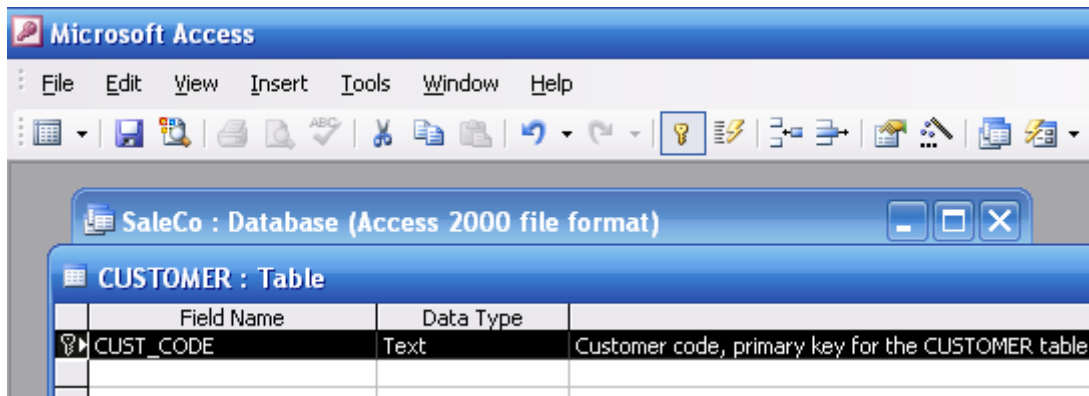
1. Click in the narrow column just to the left of the CUST_CODE attribute in the **Field Name** column. Note that this action inverts the screen for the first field name row. (Figure 14 shows that the background is black and the lettering is white.)
2. Click on the key symbol on the button bar along the top of the screen to set the PK.

Figure 14 Setting the Primary Key



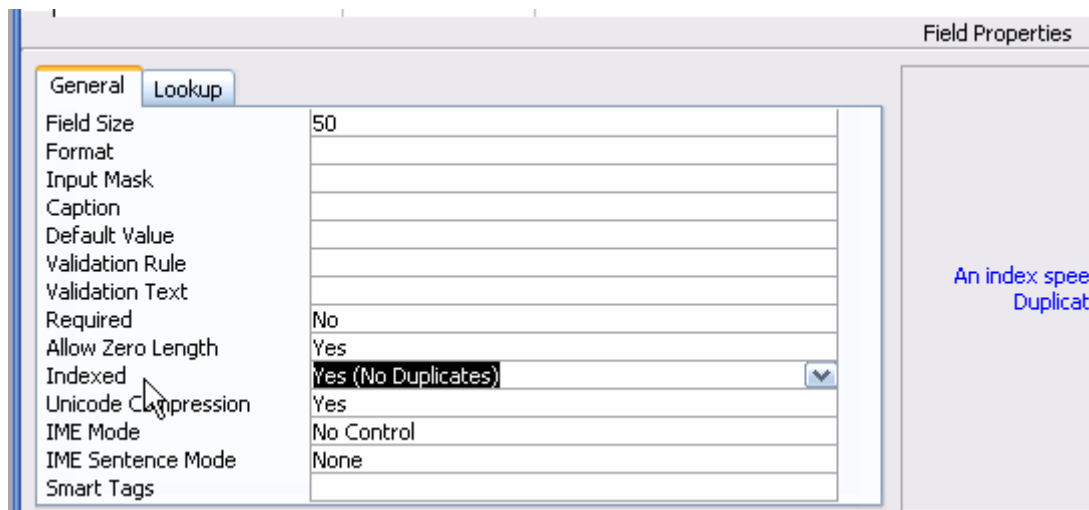
The completion of step 2 generates the screen you see in Figure 15. Note that the narrow column to the left of the **Field Name** column now shows the key symbol to indicate that the CUST_CODE attribute is now the PK. (We suggest that you save the table again.)

Figure 15 Primary Key is Set



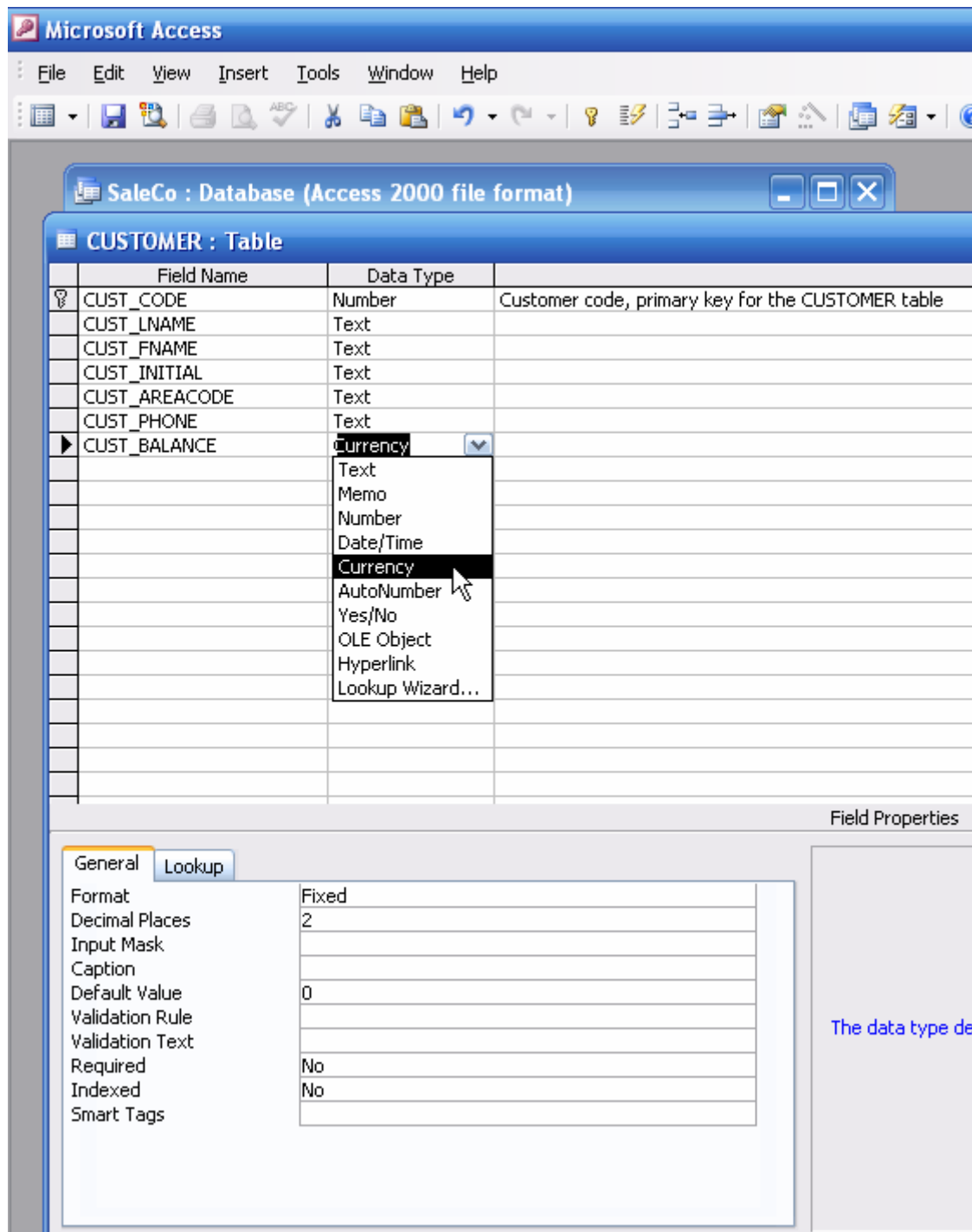
If you look at the CUST_CODE **Field Properties** box again, you will see – check Figure 16 – that the CUST_CODE **Indexed** property has been reset to **Yes (No Duplicates)**. Note that this new property enforces entity integrity, because no duplicate values will be permitted.

Figure 16 The CUST_CODE Field Properties



Using the procedures you have just learned, go ahead and create the remaining attributes for the **SaleCo** database’s CUSTOMER table. (Use the ERD in Figure 1 as your guide.) Note that the CUST_BALANCE is a **currency** value in Figure 17. Make sure that you use a **Text** format for all of the remaining attributes and that you limit the field lengths for CUST_LNAME, CUST_FNAME, CUST_INITIAL, CUST_AREACODE, and CUST_PHONE to 20, 20, 1, 3, and 8, respectively. (Phone numbers will be entered with the format 999-9999.) Given the self-describing field names, there is little need to continue the entry of more detailed descriptions. For example, the CUST_LNAME is clearly a customer’s last name, so no further description is necessary.

Figure 17 The Completed CUSTOMER Table



Make sure that you save the table again. You are now ready to enter some data. Incidentally, note that the CUST_BALANCE default value in Figure 17 is 0. Therefore, the CUST_BALANCE (currency) value will be shown as 0.00 until you change it.)

1.2.1 Data Entry

Close the CUSTOMER table’s design view to return to the table options screen, then select the CUSTOMER table and click on the **Open** table option shown in Figure 18 to generate Figure 19. (You can also double-click on the CUSTOMER table to see Figure 19.)

Figure 18 Opening the CUSTOMER Table for Data Entry

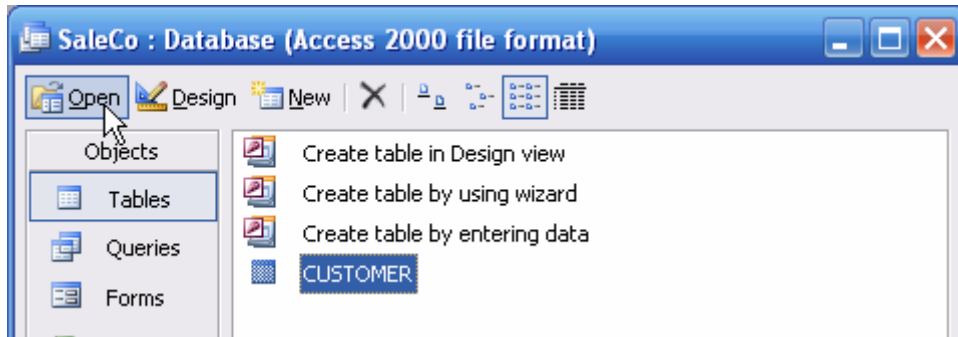
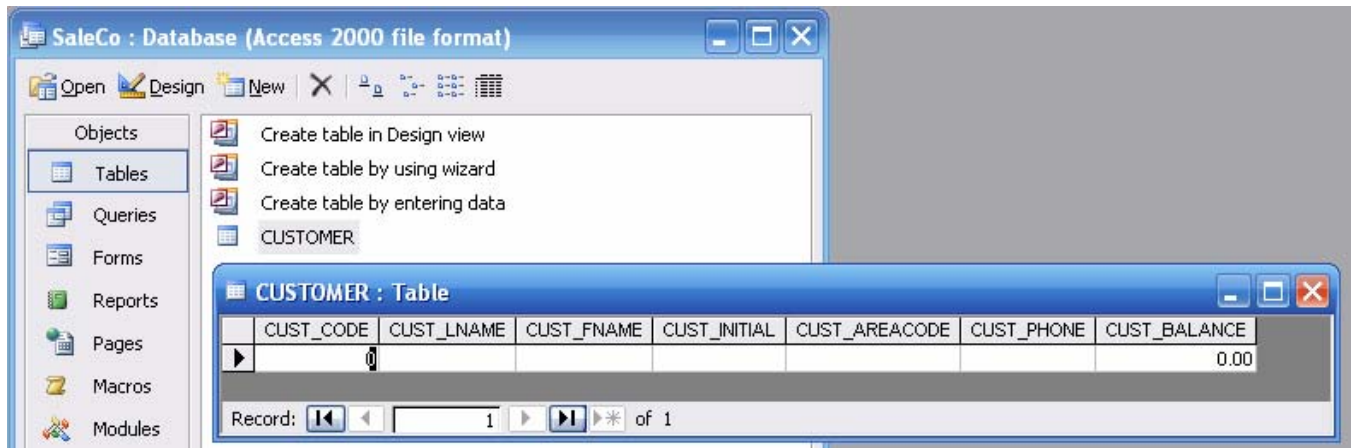


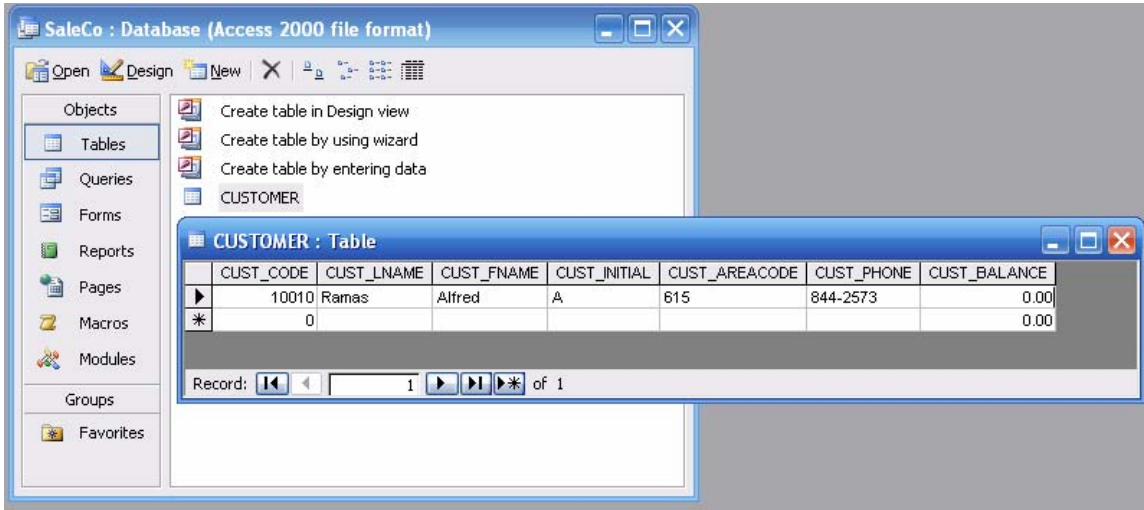
Figure 19 The CUSTOMER Data Select View



NOTE

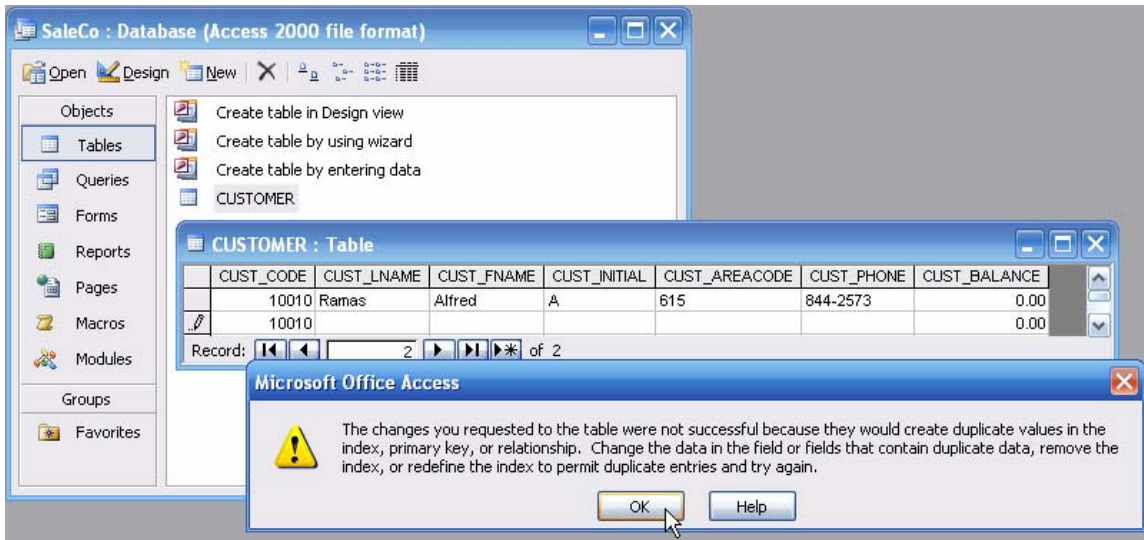
Before you enter any data, you should remember that the “parent” table entries always precede the “child” data entries. In other words, in a 1:M relationship, the “1” side’s data entry precedes that of the “M” side data entry. For example, if you try to make a data entry into the INVOICE table for a customer who does not yet exist, you will commit a data integrity violation. You will learn how to create relationships between tables in Section 1.3 – and you will then discover that you will have the option to have MS Access enforce referential integrity.

Figure 20 Entering the First CUSTOMER Record



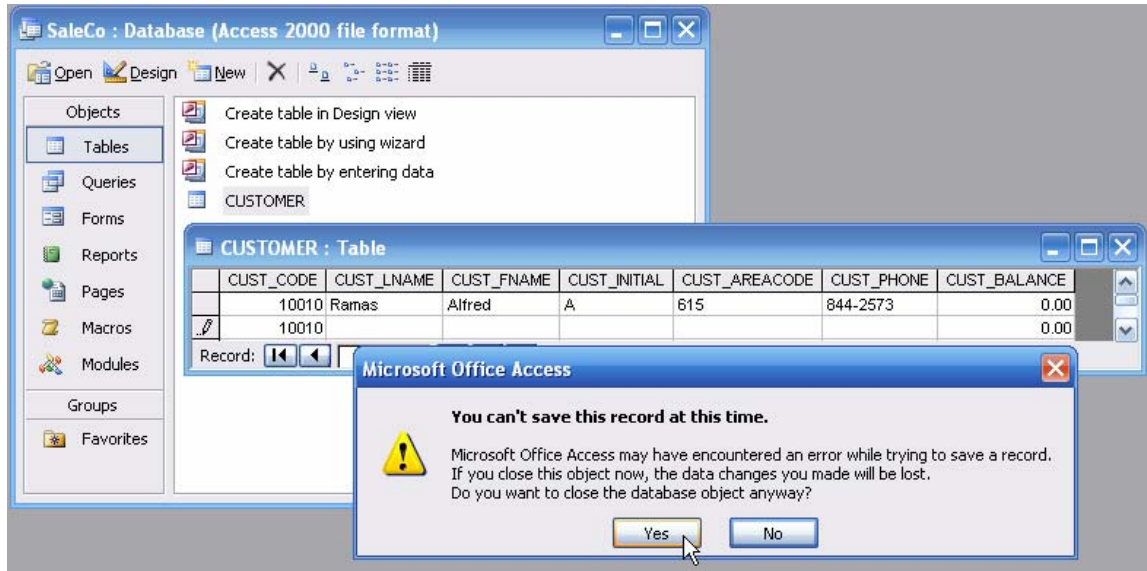
Suppose you now try to enter the second record. After accepting the 0.00 CUST_BALANCE value, you tap the **Enter** key to move the cursor to the second record. Now type in the same 10010 CUST_CODE value you used for the first record ... and then try to close the CUSTOMER table. Your reward will be the error message shown in Figure 21, because you violated entity integrity requirements.

Figure 21 Enforcement of Entity Integrity



Click **OK** to acknowledge the error message. This action will generate the result you see in Figure 22. (As you can tell, the MS Access window selection is “restore down” to ensure that all the components you see in Figure 22 can be shown. If you had selected the “maximize window” option, you would be limited to seeing the table and the dialog box.)

Figure 22 Enforcement of Entity



If you click on the **Yes** button shown in Figure 22, Access will return you to the table option screen you saw in Figure 18. In this case, the second record will not be saved and the table simple retains the first record “as is.” If you click on the **No** button, the cursor returns to the data entry screen. This action lets you make the necessary correction by typing in a CUST_CODE value other than 10010.

The remaining records are handled just as the first record was. When you are done, the CUSTOMER table contents should match Figure 23.

Figure 23 Completed CUSTOMER Table Entries

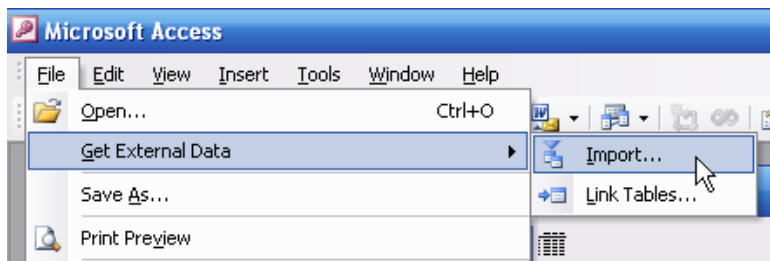
	CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
▶	10010	Ramas	Alfred	A	615	844-2573	0.00
+	10011	Dunne	Leona	K	713	894-1238	0.00
+	10012	Smith	Kathy	vW	615	894-2285	345.86
+	10013	Olowski	Paul	F	615	894-2180	536.75
+	10014	Orlando	Myron		615	222-1672	0.00
+	10015	O'Brian	Amy	B	713	442-3381	0.00
+	10016	Brown	James	G	615	297-1228	221.19
+	10017	vWilliams	George		615	290-2556	768.93
+	10018	Farriss	Anne	G	713	382-7185	216.55
+	10019	Smith	Olette	K	615	297-3809	0.00
*	0						0.00

You are now ready to create the remaining tables (INVOICE, LINE, PRODUCT, and VENDOR) and their contents. However, rather than wasting your time on such repetitive tasks, we will show you how to import these items from a database you already have. That database, named **Ch07_SaleCo**, is used in the text's Chapter 7, Advanced SQL.” This database is available online, so go ahead and download it now into a folder of your choice. You will learn how to import database components from the **Ch07_SaleCo** database to your **SaleCo** database in the next section 1.2.2.

1.3 Importing Components from other MS Access Databases

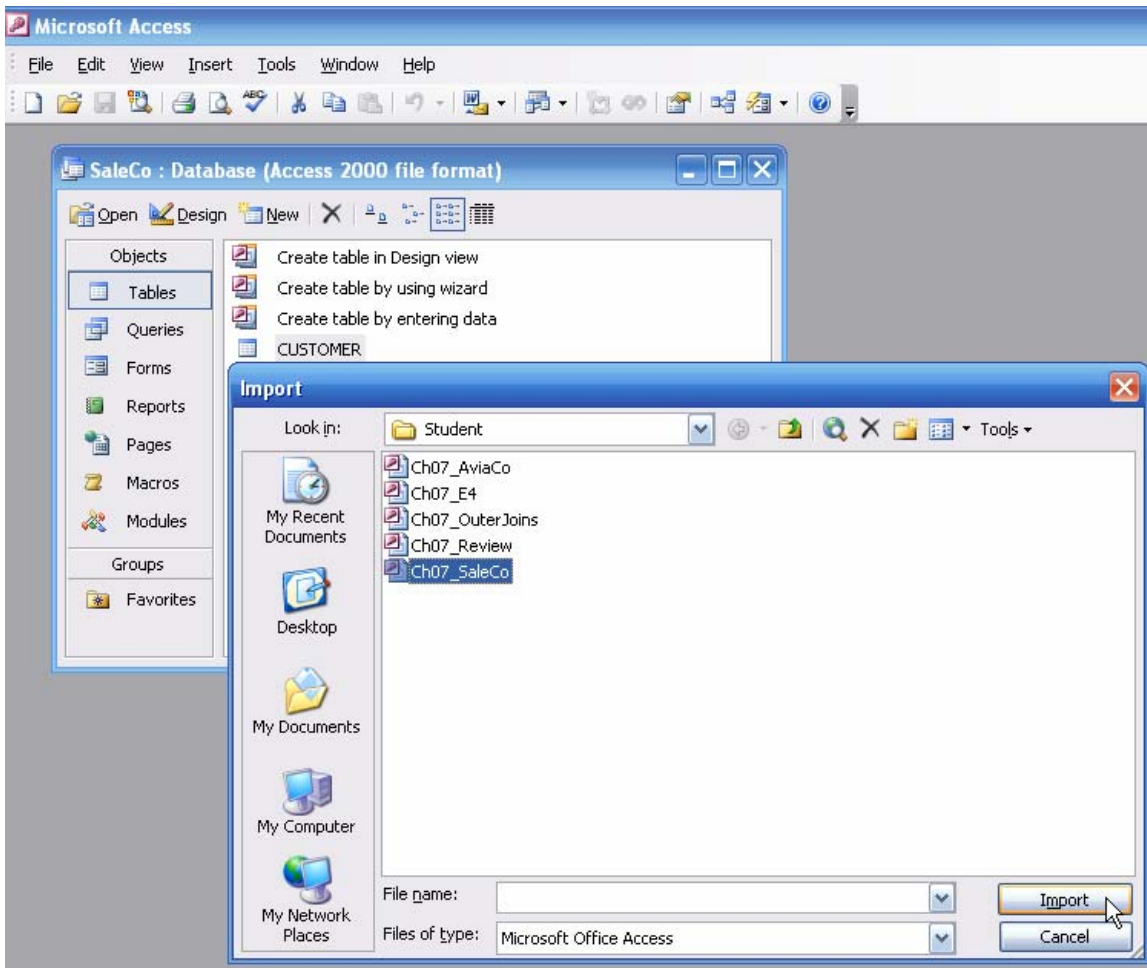
To import a table – or, for that matter, any other Access object – start by selecting the **File/Get External Data/Import** sequence shown in Figure 24.

Figure 24 Starting the Import Sequence



Next, select the data source. In this example, the **Ch07_SaleCo** database stored in the student folder will be used, so move to the folder that contains this database and select the **Ch07_SaleCo** database as shown in Figure 25.

Figure 25 Selecting the Data Source



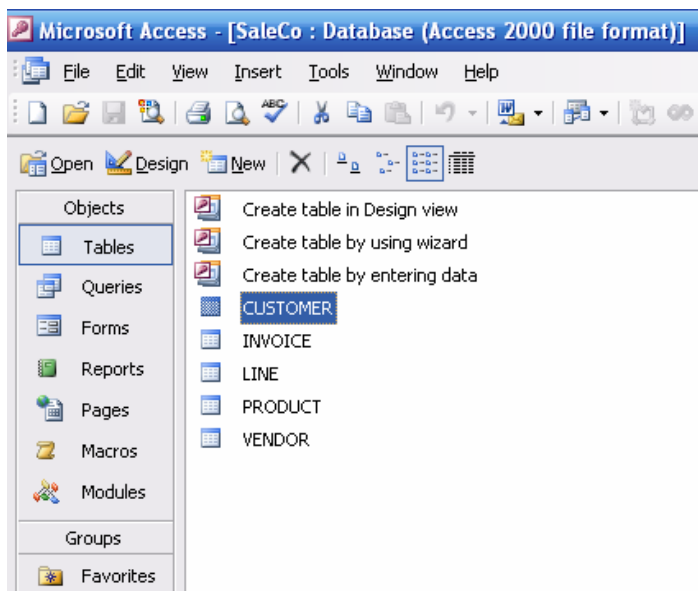
Now select what objects you want to import and in what format you want them imported. Figure 26 shows that **Tables** are to be imported. Note that the **Options >>** have been selected to give you a chance to make the selections shown in Figure 27.

As you examine Figure 27, note the following selections:

- The INVOICE, LINE, PRODUCT and VENDOR tables have been marked.
- The **Relationships** option has been de-selected – no checkmark is shown in the box. (If the box contains a checkmark, click on it to remove the checkmark. The reason for not including the relationships is that you should learn how to create such relationships later.)
- The **Definition and Data** option has been selected to ensure that the table structure and all the data contained in each selected table will be imported.

After you have made the selections shown in Figure 27, click **OK** to import the tables. The results of this action are shown in Figure 28.

Figure 28 The Imported Tables



1.3.1 Editing the Imported Tables

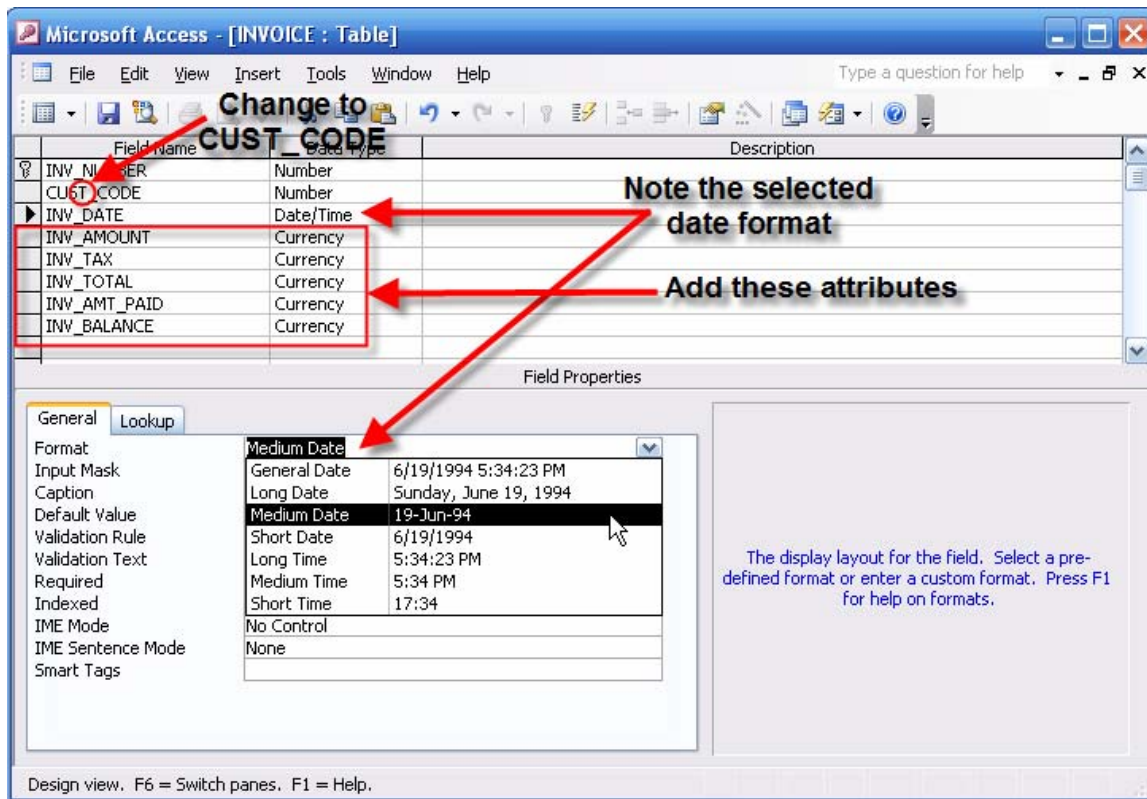
As you can tell by looking at Figure 28, the imported tables are all there ... but you will discover that their attribute names do not precisely conform to the ones used in Figure 1's ERD. For example, note that in Figure 29 the INVOICE table's FK is CUS_CODE, rather than CUST_CODE. Therefore, you should edit the attribute names in each of the tables.

Figure 29 Imported INVOICE Table Structure

INVOICE : Table	
Field Name	Data Type
INV_NUMBER	Number
CUS_CODE	Number
INV_DATE	Date/Time

Make the changes illustrated in Figure 30. Note the addition of attributes and their field types. (All changes in the table structures are made via the **Design View**.)

Figure 30 The Edited INVOICE Table Structure



When you open the INVOICE table to see its contents, you will see that the new attributes are there, but they do not (yet) contain values. (See Figure 31.)

Figure 31 The INVOICE Table Contents

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
▶	1001	10014	16-Mar-06					
	1002	10011	16-Mar-06					
	1003	10012	16-Mar-06					
	1004	10011	17-Mar-06					
	1005	10018	17-Mar-06					
	1006	10014	17-Mar-06					
	1007	10015	17-Mar-06					
	1008	10011	17-Mar-06					
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

If you wonder why the new attributes in Figure 31’s INVOICE table have \$0.00 default values, open the INVOICE table in its design format and note the selection of the currency format and default value in the **Field Properties** box shown in Figure 32.

Figure 32 INVOICE Table Field Properties

Field Name	Data Type	Description
INV_NUMBER	Number	
CUST_CODE	Number	
INV_DATE	Date/Time	
▶ INV_AMOUNT	Currency	
INV_TAX	Currency	
INV_TOTAL	Currency	
INV_AMT_PAID	Currency	
INV_BALANCE	Currency	

Field Properties

General | Lookup

Format: Currency

Decimal Places: Auto

Input Mask:

Caption:

Default Value: 0

Validation Rule:

Validation Text:

Required: No

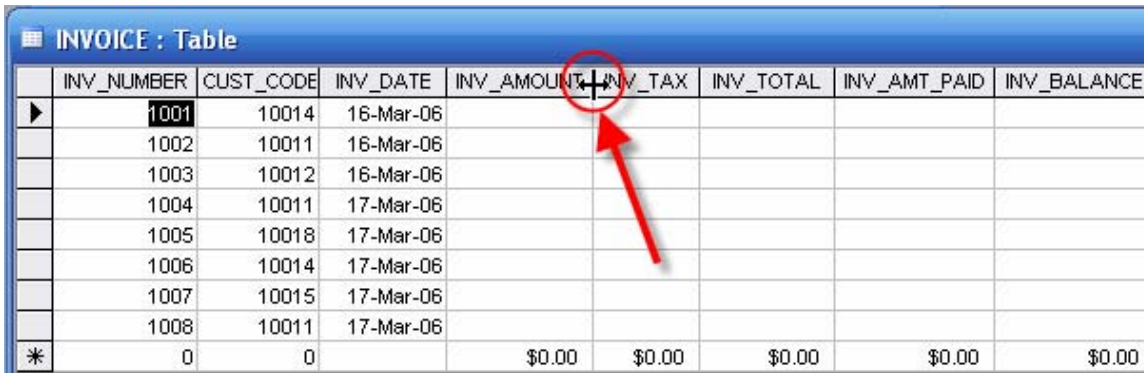
Indexed: No

Smart Tags:

The display layout for the field. Select a pre-defined format or enter a custom format. Press F1 for help on formats.

While you have the INVOICE table open, you can change the spacing between the attributes. When you put the cursor on a boundary between attributes, note that the cursor changes to a two-side arrow as shown in Figure 33. You can drag the boundary to increase or decrease the column display width, or you can double-click on any column boundary to change the column display width to whatever size is required to show the column values.

Figure 33 Changing the Column Spacing

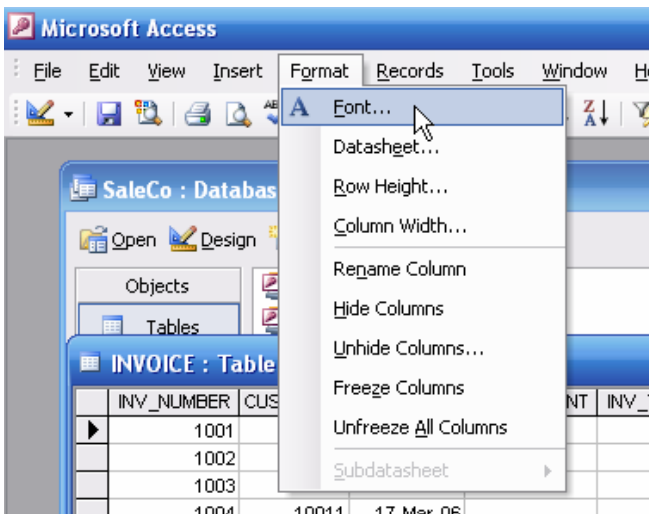


The screenshot shows the 'INVOICE : Table' view in Microsoft Access. The table has the following columns: INV_NUMBER, CUST_CODE, INV_DATE, INV_AMOUNT, INV_TAX, INV_TOTAL, INV_AMT_PAID, and INV_BALANCE. A red circle highlights the vertical line between the INV_AMOUNT and INV_TAX columns, with a red arrow pointing to it from below. The data rows show values for INV_NUMBER (1001-1008) and INV_DATE (16-Mar-06 and 17-Mar-06). The bottom row shows summary values: 0 for INV_NUMBER and CUST_CODE, and \$0.00 for INV_AMOUNT, INV_TAX, INV_TOTAL, INV_AMT_PAID, and INV_BALANCE.

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
▶	1001	10014	16-Mar-06					
	1002	10011	16-Mar-06					
	1003	10012	16-Mar-06					
	1004	10011	17-Mar-06					
	1005	10018	17-Mar-06					
	1006	10014	17-Mar-06					
	1007	10015	17-Mar-06					
	1008	10011	17-Mar-06					
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

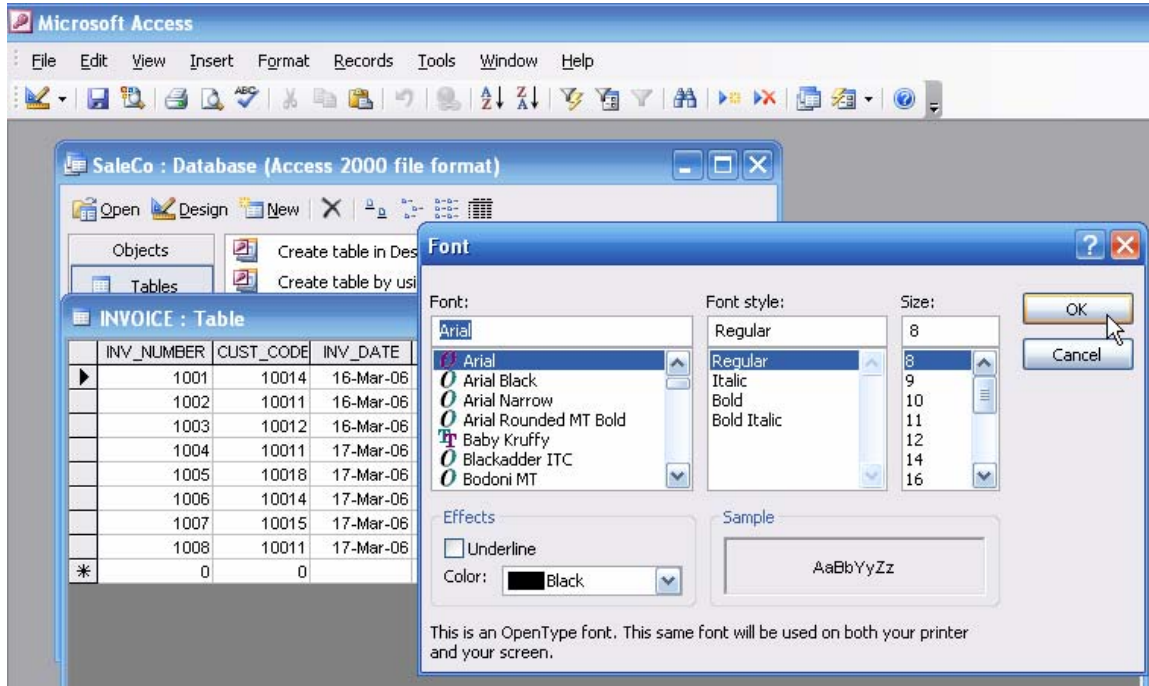
You can also change the presentation font by selecting the **Format/Font...** option shown in Figure 34.

Figure 34 Selecting the Font Format



After making the **Format/Font...** selection shown in Figure 34, note the selection of the font, style, and size in Figure 35. Click **OK** when you are done with the selection process.

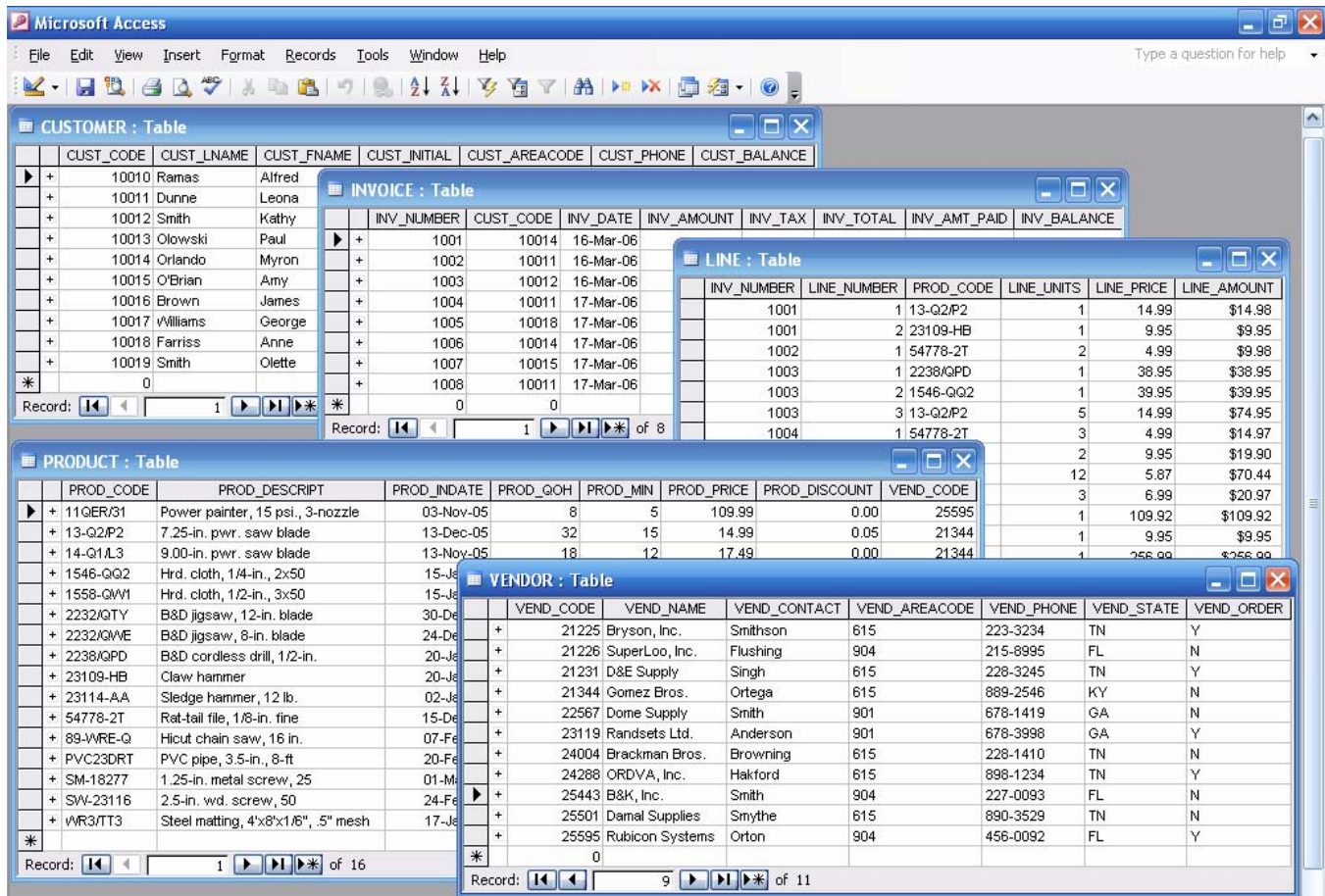
Figure 35 Selecting the Font Format – Part 2



Go ahead and experiment with different formatting options. A good rule to follow is this: If you don't know what something is or what it does, right-click on it.

Now go ahead and edit the table structures of the remaining tables. Use the database tables shown in Figure 36 as your guide. Remember, you imported most of the tables from the **Ch07_SaleCo** database, so the table values match those in that database *at this point*. However, while the *values* match, a careful examination of Figure 36 shows that we have already made a few changes in the **SaleCo** database.

Figure 36 The SaleCo Database Tables



For example, if you look at the LINE table in Figure 36, note that:

- The LINE table now has an additional currency field, **LINE_AMOUNT**, which is the product of the **LINE_UNITS** and **LINE_PRICE**. (The third record in the LINE table shows that the **LINE_AMOUNT** = 2 x \$4.99 = \$9.98.)
- The **LINE_AMOUNT** data type and field property were both set to **currency**, thus causing the dollar signs to precede the amount. You can change all the other fields that store dollar values to a currency data type at any time.
- There is an entry error in the first record of the LINE table – the **LINE_PRICE** is \$14.99 and the **LINE_UNITS** value is 1, so the **LINE_AMOUNT** should be 1 x 14.99 = \$14.99. (Go ahead and open the LINE table in its datasheet view and make the correction.)

1.4 Tracing a Transaction

How do you enter the values for the new attributes in the INVOICE table? You can, of course, hand-calculate the invoice amount, tax, total, and then enter the amount paid and calculate the balance. To get that job done, you'll have to do some major work. Let's do just one invoice entry at this point, using the "activity trace" shown in Figure 37. (Note that the LINE_AMOUNT in the LINE table's first record has been corrected.)

Figure 37 Tracing a Purchase

The screenshot shows the Microsoft Access interface with three tables open:

- CUSTOMER : Table**

	CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
+	10010	Ramas	Alfred	A	615	844-2573	0.00
+	10011	Dunne	Leona	K	713	894-1238	0.00
+	10012	Smith	Kathy	vW	615	894-2285	345.86
+	10013	Olowski	Paul	F	615	894-2180	536.75
+	10014	Orlando	Myron		615	222-1672	6.94
+	10015	O'Brian	Amy	B	713	442-3381	0.00
	615				297-1228		221.19
	615				290-2556		768.93
	713				382-7185		216.55
	615				297-3809		0.00
- INVOICE : Table**

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
+	1001	10014	16-Mar-06					
- LINE : Table**

	INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE	LINE_AMOUNT
+	1001	1	13-Q2/P2	1	14.99	\$14.99
+	1001	2	23109-HB	1	9.95	\$9.95
- PRODUCT : Table**

	PROD_CODE	PROD_DESCRIPTOR	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_CODE
+	11QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-05	8	5	109.99	0.00	25595
+	13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-05	32	15	14.99	0.05	21344
+	14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-05	18	12	17.49	0.00	21344
+	1546-QQ2	Hrd. cloth, 14-in., 2x50	15-Jan-06	15	8	39.95	0.00	23119
+	1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-06	23	5	43.99	0.00	23119
+	2232/GTY	B&D jigsaw, 12-in. blade	30-Dec-05	8	5	109.92	0.05	24288
+	2232/GW6	B&D jigsaw, 8-in. blade	24-Dec-05	6	5	99.87	0.05	24288
+	2238/GPD	B&D cordless drill, 1/2-in.	20-Jan-06	12	5	38.95	0.05	25595
+	23109-HB	Claw hammer	20-Jan-06	23	10	9.95	0.10	21225

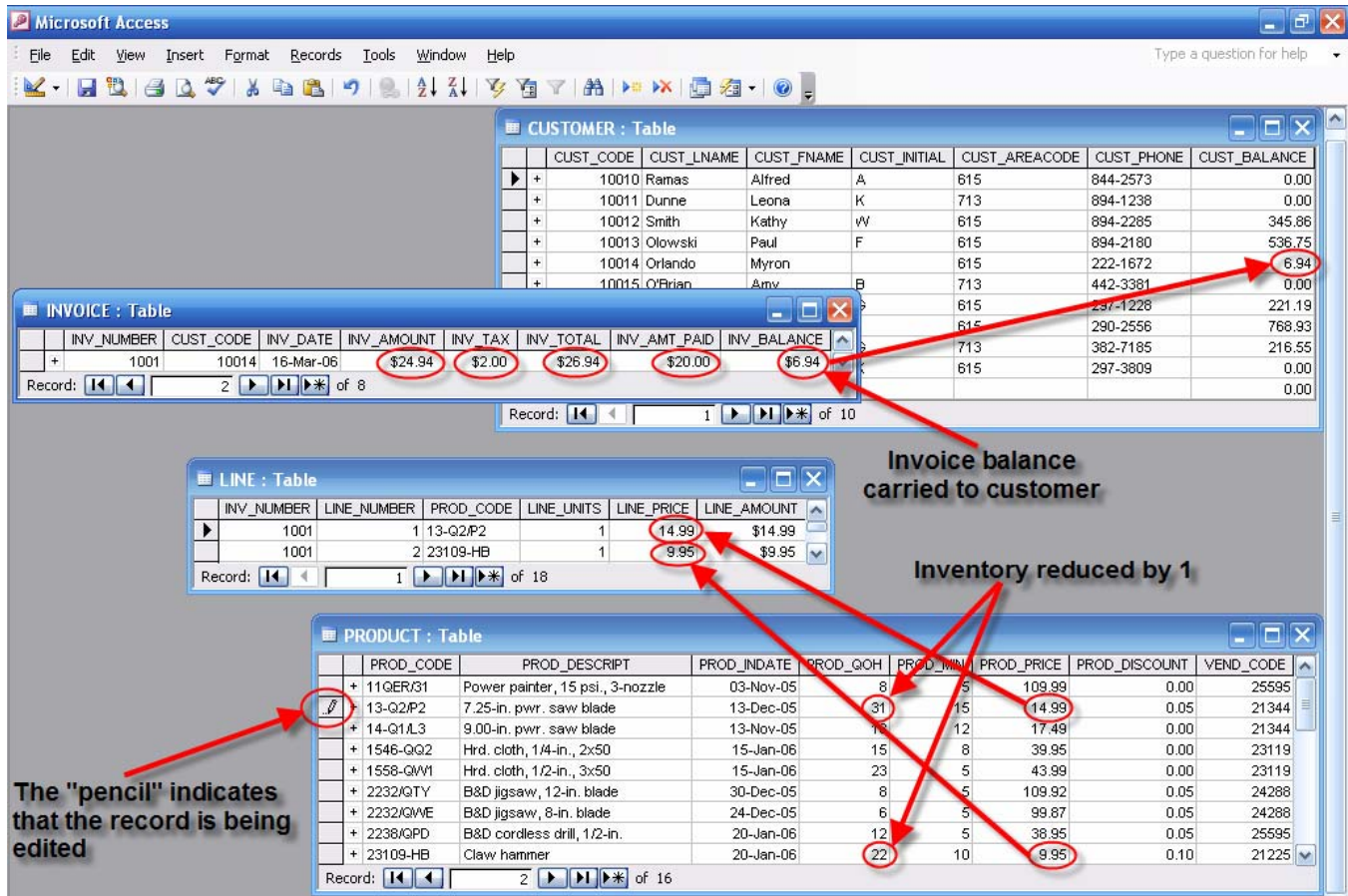
As you can tell by looking at Figure 37, the following activities take place:

1. The customer 10014 (Myron Orlando) made the purchase(s) that produced invoice number 1001.
2. Invoice number 1001 records two line items, for a purchase of one \$14.99 saw blade and one \$9.95 claw hammer. Let's assume for simplicity's sake that discounts are not offered unless the purchase is made by a commercial customer ... and none of the customers in the CUSTOMER table are commercial purchasers. Therefore, type the PRODUCT table's PROD_PRICE values into the LINE table as LINE_PRICE values.
3. The total invoice amount is $\$14.99 + \$9.95 = \$24.94$.
4. Assuming a tax percentage of 8%, the tax is $\$24.94 \times 0.08 = \1.9952 , rounded to \$2.00.
5. The invoice total is thus $\$24.94 + \$2.00 = \$26.94$.
6. Let's assume that the customer pays \$20.00, leaving a \$6.94 balance.

7. Customer 10014 now has a customer balance of $\$0.00 + \$6.94 = \$6.94$.
8. The product table must be updated, too. The PROD_QOH values for each of the two products must be decreased by one each to reflect the sale activity.

The completed transaction is shown in Figure 38. (The table limits were “dragged” to ensure that all the tables would fit in the same window in Figure 38.)

Figure 38 The Completed Transaction



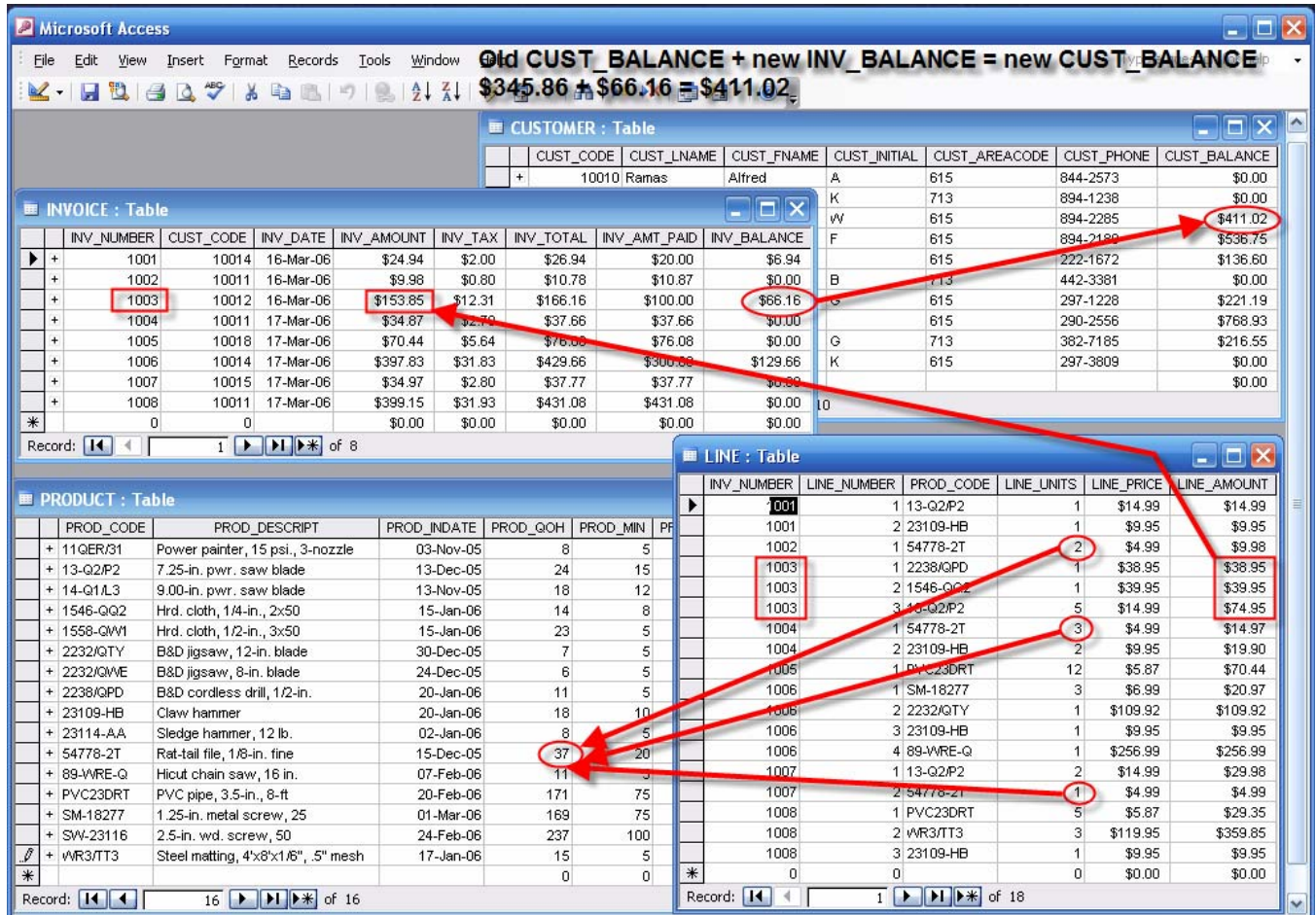
NOTE

Remember that the product price is copied to the LINE table to maintain the historical accuracy of the transaction. It is likely that, over time, the product prices will change ... but the original product prices will be maintained in the LINE table to reflect the product price that was correct *at the time of the transaction*. If necessary, review the text's Chapter 3, "The Relational Database Model," Section 3.7, "Data Redundancy Revisited."

Go ahead and complete the remaining transactions to practice the just illustrated tracing process. Note that the data entry is awkward, because you have to actually enter the product price in the LINE table, multiply the product price by the number of items (LINE_UNITS) to get the line total values. After that

job is done, you'll have to calculate the subtotals, the sales tax, and the total in the INVOICE table, and then update the inventory -- PROD_QOH in the PRODUCT table -- and the customer balance -- CUST_BALANCE in the CUSTOMER table. When you are done, your completed tables should contain the values shown in Figure 39.

Figure 39 All Completed Transactions



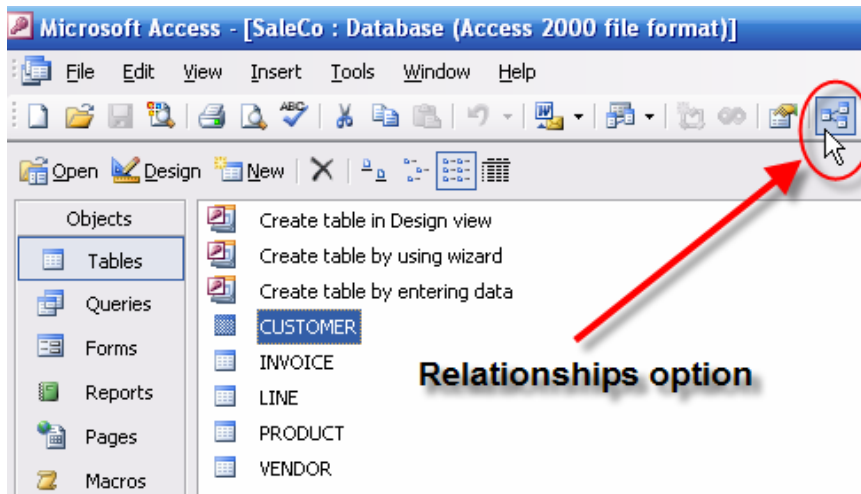
If you have recorded all the transactions shown in Figure 39 manually, you realize that such time consuming and exacting tasks are not a good idea in the real world. You will learn in Section 2.1 that MS Access includes a query type known as an **Update Query** that, as its name suggests, can make the necessary transaction updates. In Section 5.1 you will learn that macros can be used to automate the transaction update process.

1.5 Setting the Relationships between Tables

Thus far, you have learned to create, populate, and import tables. Actually, if you were to create a real world relational database, you would first create the table structures and then create the relationships among them. The reason for doing so is simple – you want to make sure that referential integrity is enforced. (If you don't quite remember what referential integrity is and why it is important, review the text's Chapter 3, "The Relational Database Model," Section 3.2, "Keys.")

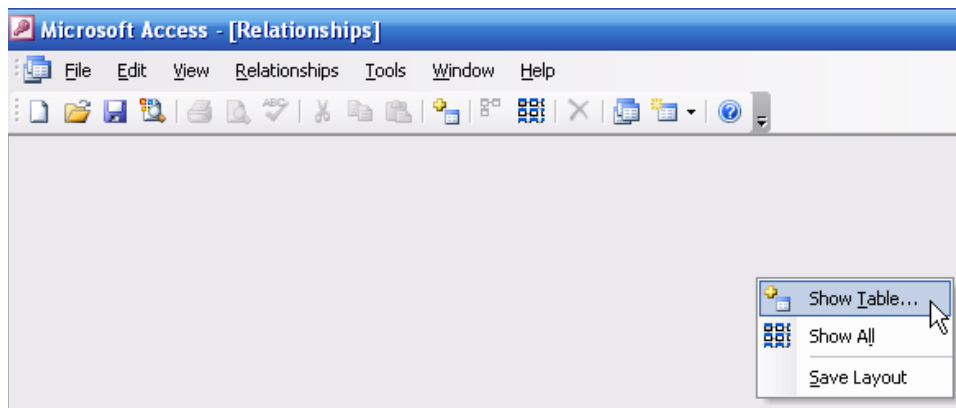
To create a relationship, start by selecting the relationships option as shown in Figure 40.

Figure 40 Select the Relationships Option



After you click on the relationships button shown in Figure 40, a blank relationships screen is shown. (That is, the relationships screen will be blank if you have not yet established relationships or you have not imported the relationships with the tables when you learned how to import tables.) Right-click anywhere on the blank relationships screen to pop up the options – **Show Table ...**, **Show All**, **Save Layout** -- you see in Figure 41.

Figure 41 Show Tables



Click on the **Show Table ...** option shown in Figure 40 to generate the **Show Table** window you see in Figure 42. Now select the two tables as shown in Figure 42 and the click on the **Add** button. (You can also double-click on each table to move it from the **Show Table** window to the relationships screen.) This action will generate the screen shown in Figure 43.

Figure 42 Select Tables

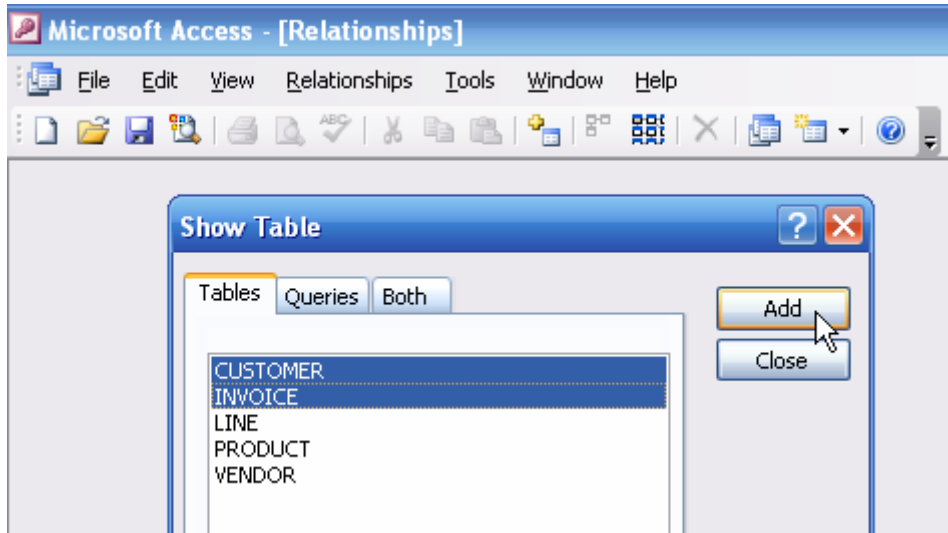
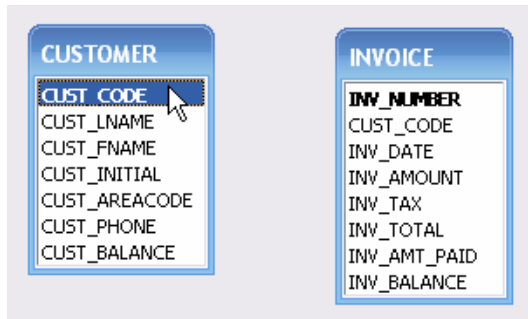
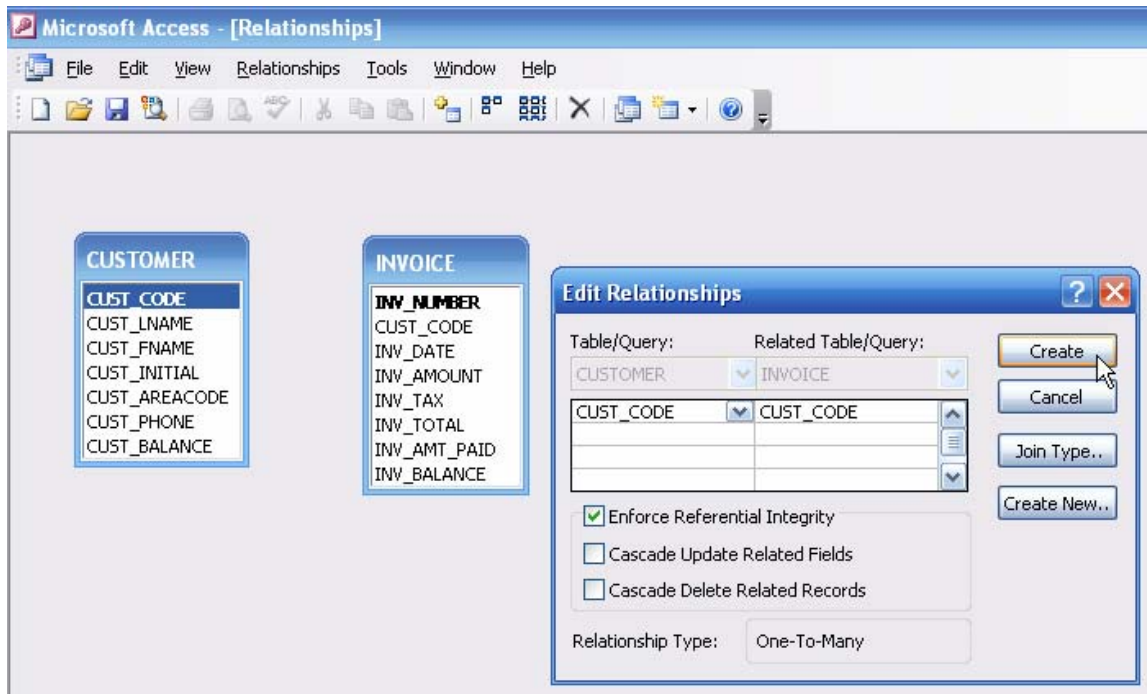


Figure 43 Selected Tables



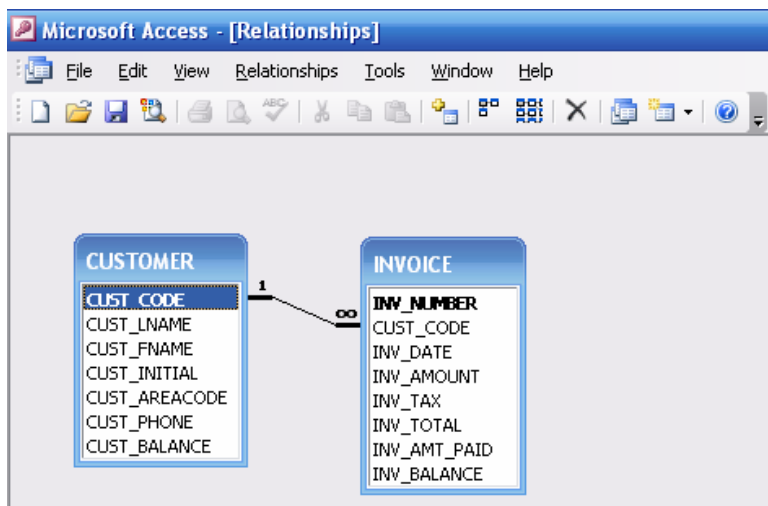
To create the relationship between the CUSTOMER and INVOICE tables, select the CUST_CODE in the CUSTOMER table and drag it to the CUST_CODE in the INVOICE table, and then drop it on the CUST_CODE in the INVOICE table. This action will produce the **Edit Relationships** dialog box you see in Figure 44.

Figure 44 Create Relationship



Next, click on the **Enforce Referential Integrity** option you see in Figure 44's **Edit Relationships** window to place a checkmark in the square shown on the left side of that option and then click on the **Create** button to create the relationship. This action will produce the results shown in Figure 45. Note that the "1" side of the relationship is marked by the boldfaced **1**, while the many (M) side of the relationship is marked by the infinity symbol ∞ .

Figure 45 The Relationship Between CUSTOMER and INVOICE

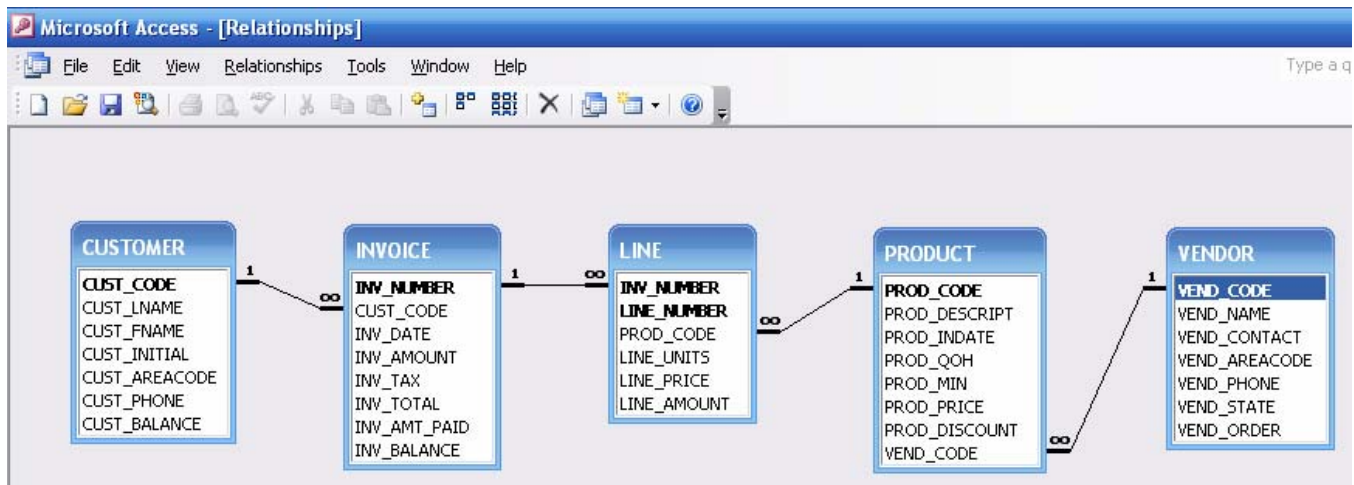


Note

Always drag and drop from the 1 side to the M side in a one-to-many (1:M) relationship. If you are creating a relationship between tables in a 1:1 relationship, drag from the parent entity to the dependent entity.

Now go ahead and create the relationships between all the tables. When you are done, your results will resemble Figure 46. (We have dragged and sized the tables to enhance the presentation.)

Figure 46 The Relationships for all the SaleCo Database Tables

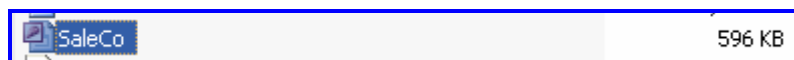


1.6 Repairing and Compacting the Database

A word of caution is in order. When you do a lot of work on your database, it tends to grow rapidly and its data dictionary fills up with all sorts of objects that you have edited or objects that do not exist. Sooner or later, this situation will degrade the performance of your database. Therefore, clean up the debris frequently.

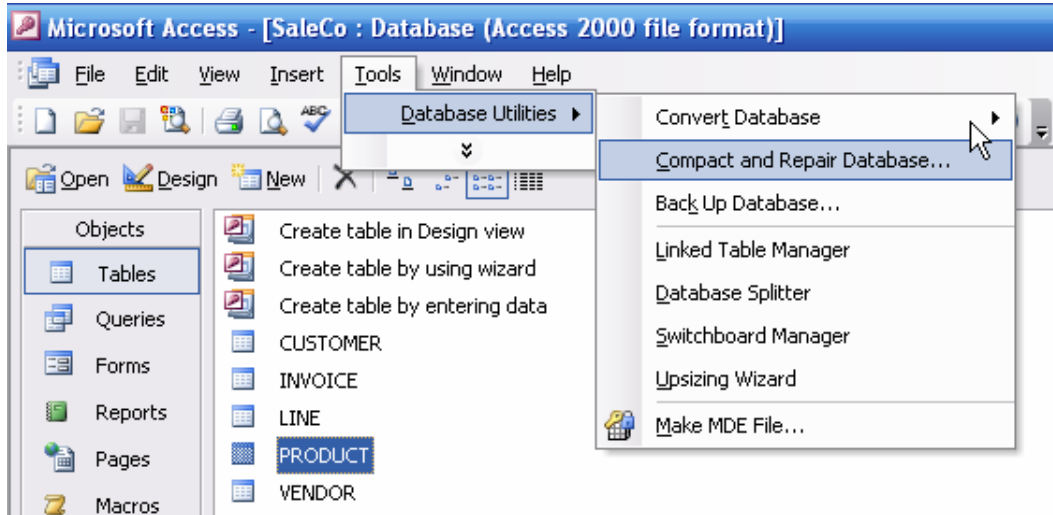
In the following sequence, note that the **SaleCo** database size is 596 KB at this point. (See Figure 47. Depending on the number of changes you made, your database size is likely to differ from the 596 KB you see here.)

Figure 47 The SaleCo Database Size Before the Cleanup



The cleanup is accomplished by selecting the sequence shown in Figure 48 – [Tools/Database Utilities/Compact and Repair Database ...](#)

Figure 48 Compact and Repair the Database



When the procedure is completed, look how much smaller the [SaleCo](#) database has become. Figure 49 shows that the [SaleCo](#) database now uses only 260 KB.

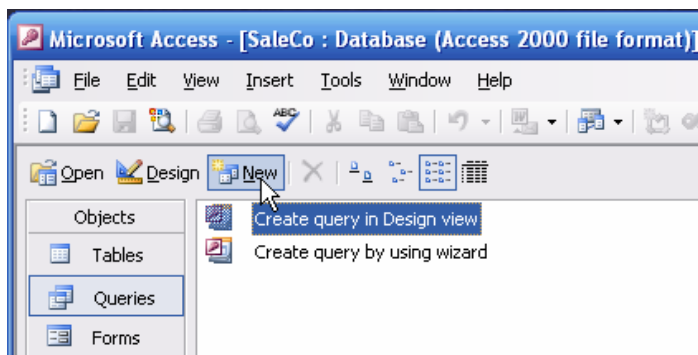
Figure 49 The SaleCo Database Size After the Cleanup



2.1 Queries

Queries are used to extract data from the database and to help transform data into information. To create a query, follow the procedure shown in Figure 50. That is, click on the [Queries](#) option in the [Objects](#) column and then click on the [New](#) option.

Figure 50 Creating the First Query



Clicking on the **New** option shown in Figure 50 will produce the **New Query** dialog box in Figure 51. Select the **Design View** option from the list and then click **OK** to generate the screen you see in Figure 52. (Many of the wizards are useful, but you are better served by using the **Design View** option if you want to learn how to create and manipulate queries. All the queries in this tutorial will be created with the help of the MS Access Graphical User Interface, or GUI.)

Figure 51 Selecting the Design View

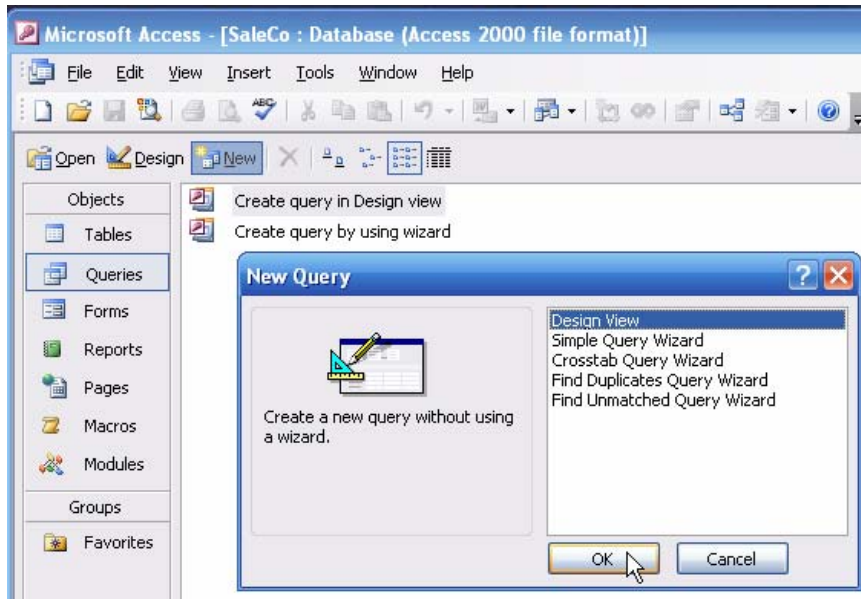


Figure 52 Selecting the CUSTOMER Table

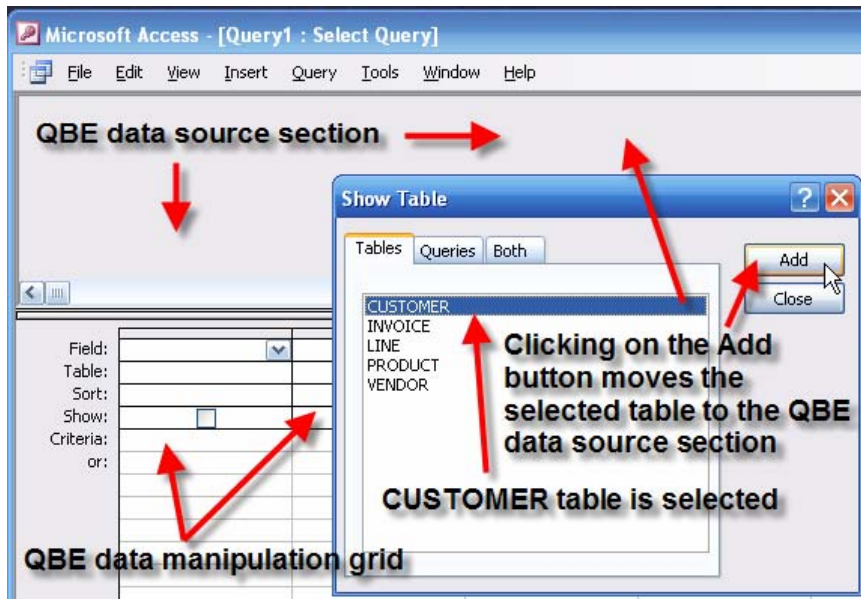
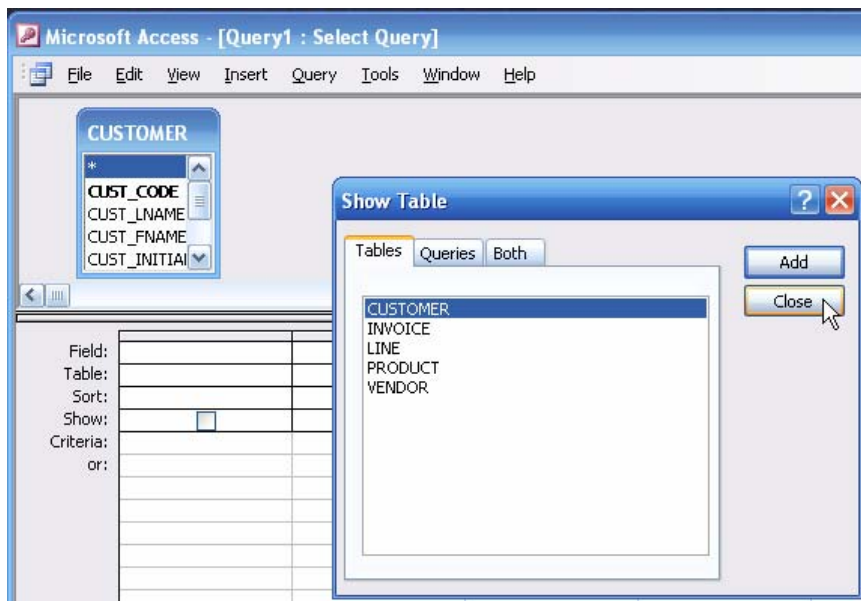


Figure 52 shows three QBE (query by example) window components.

1. The top – blank -- portion of the window is the QBE window’s *data source* display, which may be a table or another query.
2. The bottom – lined -- portion shows the available options in the QBE *grid*. The grid represents the *data manipulation* portion of the QBE window. As its name suggest, the **Field:** option lets you place a table’s – or query’s –field on the line. The **Table:** shows the field’s origin. (Clicking on the **Queries** tab will show the list of queries that are available as data sources. You can query a query.) The **Sort:** option lets you sort selected field values in either ascending or descending order. The **Show:** option lets you select whether or not to display a selected field value set. (Note that the box on the **Show:** line is not marked at this point, indicating that a selected field’s values will not be shown.) The **Criteria:** option lets you select various display options and constraints. And the **Or:** option lets you modify the **Criteria:** constraint selection. (For example, you might place a restriction of the query output for a CUSTOMER table to show only customers named “Smith” ... *or* to also show customers whose current balance is over \$200.)
3. The **Show Table** dialog box lets you select tables or queries to be placed as data sources in the first portion of the screen. (If the **Show Table** dialog box is not shown, right-click on the blank portion of the screen.)

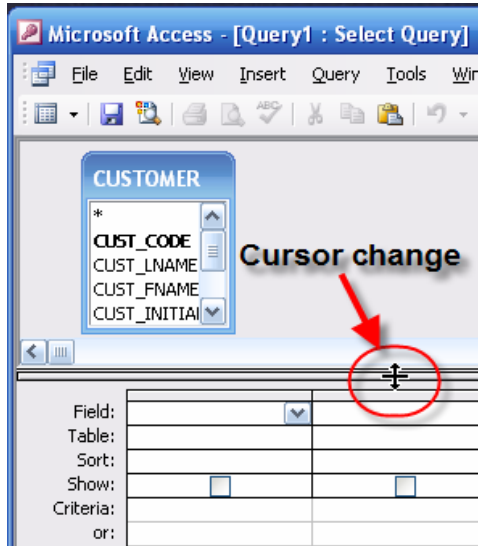
To place the CUSTOMER table on the data source portion of the screen, either select the CUSTOMER as shown in Figure 52 and then click on the **Add** button or double-click on the CUSTOMER table. When you have placed all the tables you need on the top (data source) portion of the screen, close the **Show Table** dialog box by clicking on its **Close** button. (See Figure 53.) In this example, only the CUSTOMER table has been selected as the data source.

Figure 53 Closing the Show Table Window



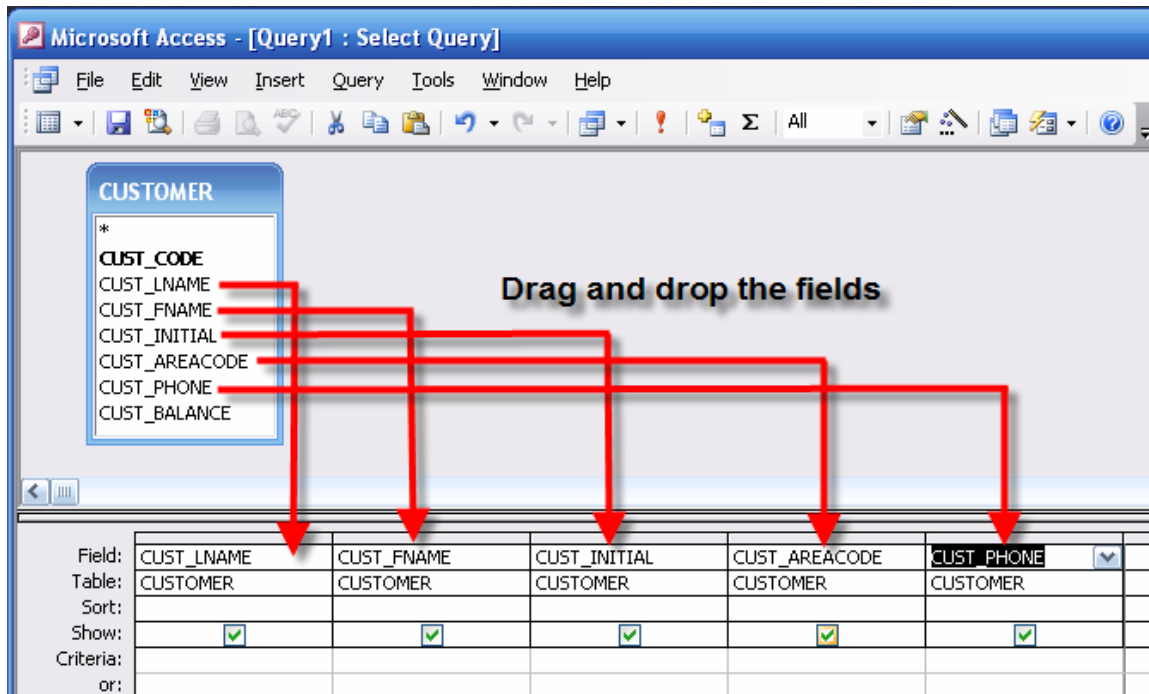
After you have selected the data source – the CUSTOMER table – and placed it on the data source portion of the QBE screen, size the QBE window’s components by dragging their limits. Figure 54 shows that the cursor changes when you place it on the boundary between the data source component that now contains the CUSTOMER table and the data manipulation grid component. When the cursor changes its shape to the two-sided arrow seen here, you can drag the boundary up or down.

Figure 54 Size the Table Window



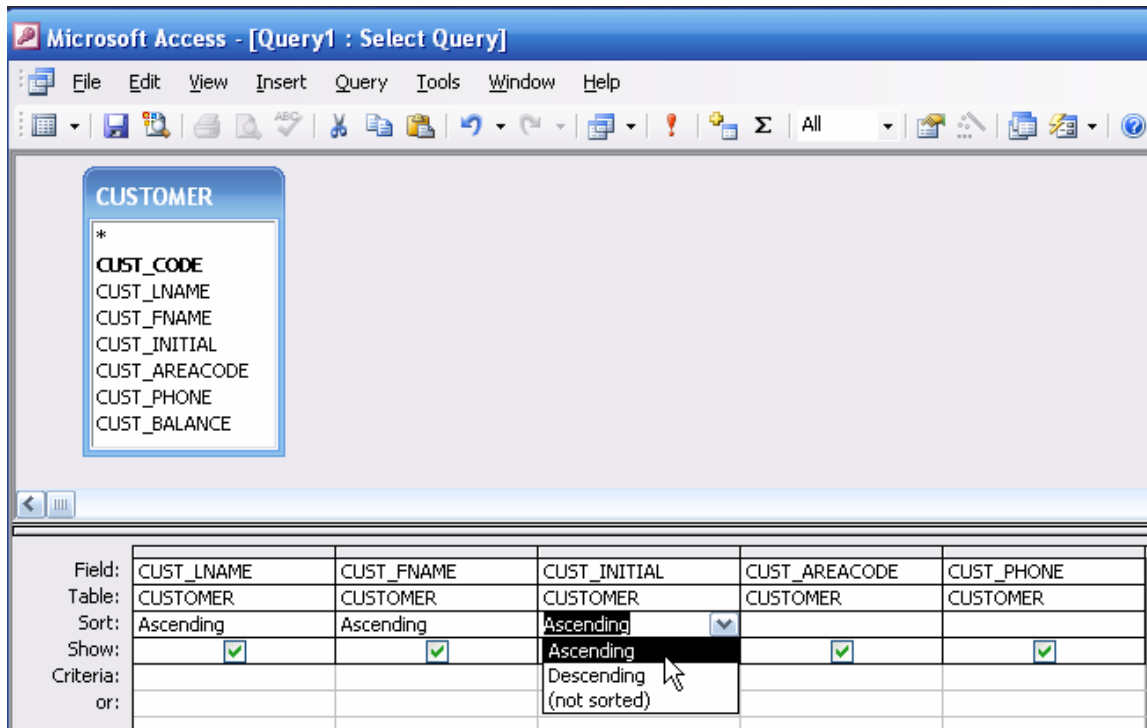
After enlarging the data component portion of the QBE window, drag the CUSTOMER limits to show all the fields and their names. Then select the CUSTOMER table’s fields and drag and drop them in the **Field:** spaces as shown in Figure 55. You can drag and drop the individual fields from the CUSTOMER table to the field spaces in the data manipulation section or you can select multiple fields and drop them on a single field space. (Access will automatically space them across the field spaces.) Note that the field origin is automatically displayed in the **Table:** portion of the grid.

Figure 55 Drag and Drop the Fields



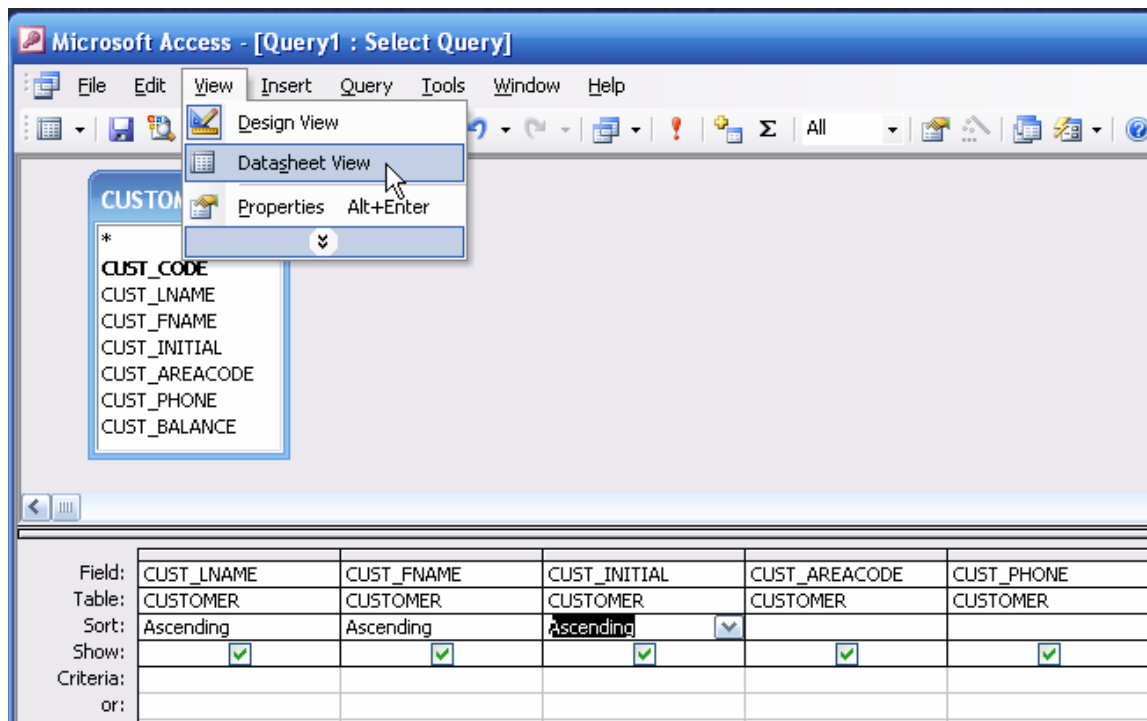
After you have selected the fields you want to use in the query, you can choose whether or not to sort the field values by clicking on the grid's **Sort:** option. Clicking on this option yields a drop-down list of option you see in Figure 56. Click on the option you want to use to move the option to the grid. (As you can tell, the CUST_LNAME, CUST_FNAME, and CUST_INITIAL show the selection of the ascending option.

Figure 56 Select Ascending Sort



If you want to see the effect of the query actions you have taken thus far, change the **View** from the current **Design View** to the **Datasheet View**, as shown in Figure 57.

Figure 57 Select the Datasheet View



The selected **Datasheet View** yields the query output shown in Figure 58.

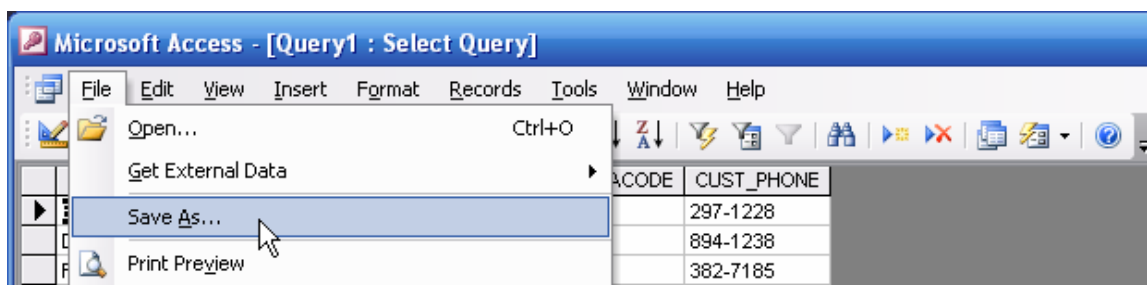
Figure 58 The Datasheet View

	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE
▶	Brown	James	G	615	297-1228
	Dunne	Leona	K	713	894-1238
	Farriss	Anne	G	713	382-7185
	O'Brian	Amy	B	713	442-3381
	Olowski	Paul	F	615	894-2180
	Orlando	Myron		615	222-1672
	Ramas	Alfred	A	615	844-2573
	Smith	Kathy	W	615	894-2285
	Smith	Olette	K	615	297-3809
	Williams	George		615	290-2556
*					

Incidentally, you can easily control the query display characteristics through the **Format** option you see in the button bar. For example, you can change the font size from the (default) 10 value to 8, you can select the font type, and so on. (Try clicking on the **Format** option to see what you can do with the query output's display.)

Before you do any additional work on the query, save it. Figure 59 shows the selection of the **File/Save As...** option.

Figure 59 Select the Save As File Option



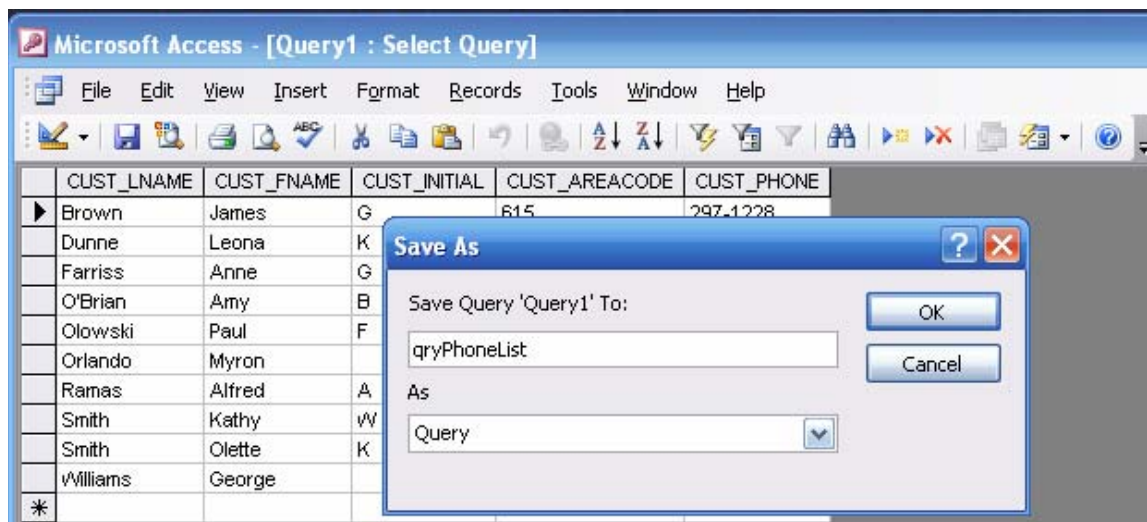
The selection of the **Save As...** option shown in Figure 59 will produce the **Save As** dialog box you see in Figure 60. The dialog box shows a default query name, in this case, **Query1**.

Figure 60 The Save As Dialog Box



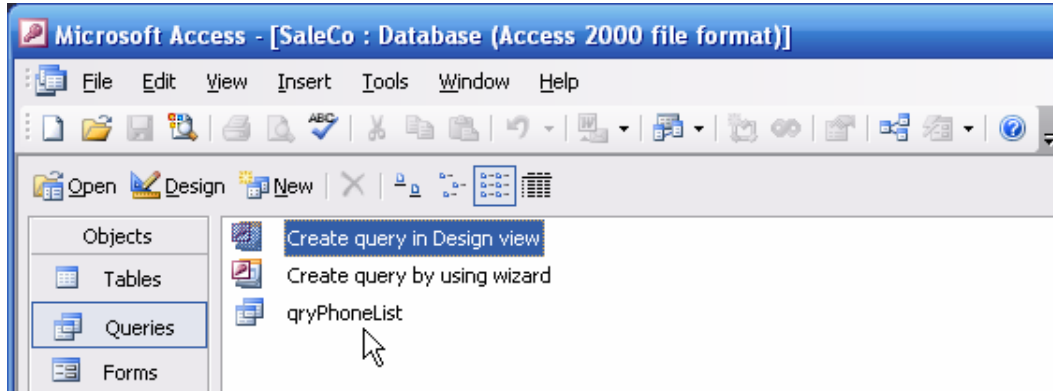
Always use a query name that is self-documenting. In this example, the query will be used as the basis for a phone list. Therefore, **PhoneList** is an appropriate name. However, you want to also show that this object is a query, so use the prefix **qry** to indicate that fact. Therefore, the appropriate name will be **qryPhoneList**. And you see that query name used in Figure 61. (You will discover that this self-documenting naming convention is very desirable, because it enables you to easily keep track of multiple components in an application set. For example, you will learn how to create forms in Section 3 ... and if you see two objects, **frmPhoneList** and **qryPhoneList**, you will know which object you are looking at, and it will be easy to see that the data source for the form is the query that has the same name. Using naming conventions that are *not* self-documenting is *not* a sign of professionalism – sowing confusion is not an ideal worth pursuing.)

Figure 61 The Query Name



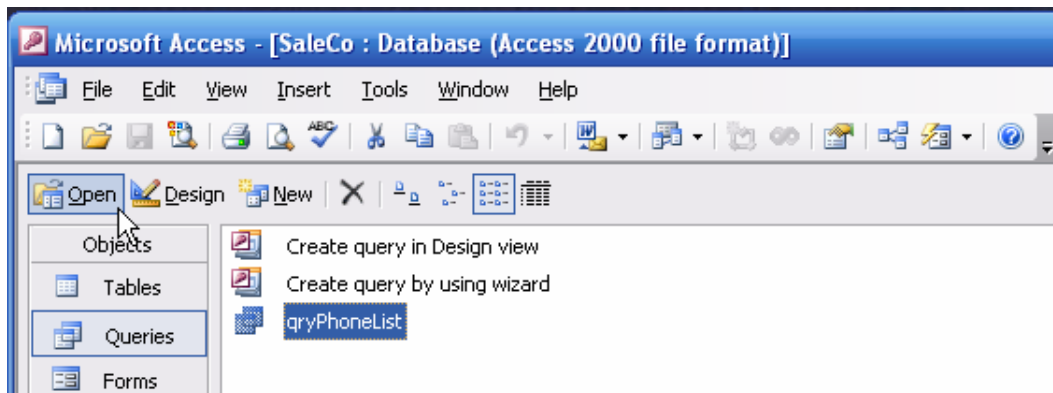
After you type the query name as shown in Figure 61, click **OK** to save the query. Note that the saved query now shows up in the **Queries** list shown in Figure 62.

Figure 62 The First Listed Query



After saving the query, you can take a quick look at its output by selecting the query **Open** option shown in Figure 63. (You can also open the query by double-clicking on the query name. Go ahead and do so, then close the query.)

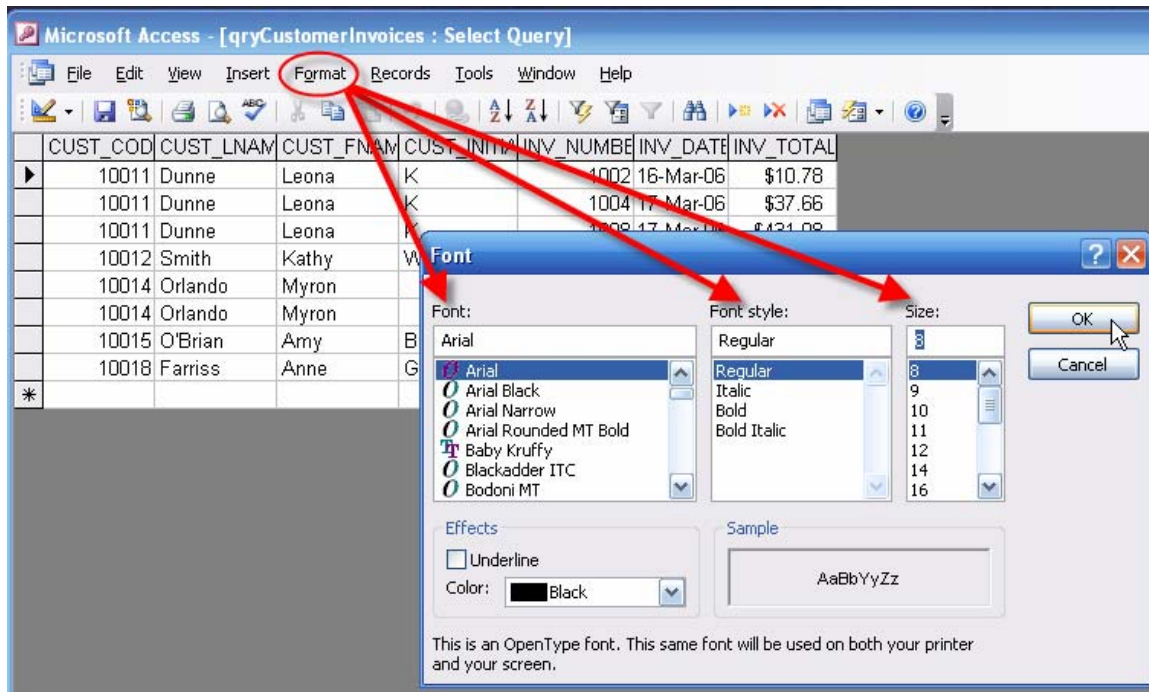
Figure 63 Open the Query



2.1.1 Editing the Query Output

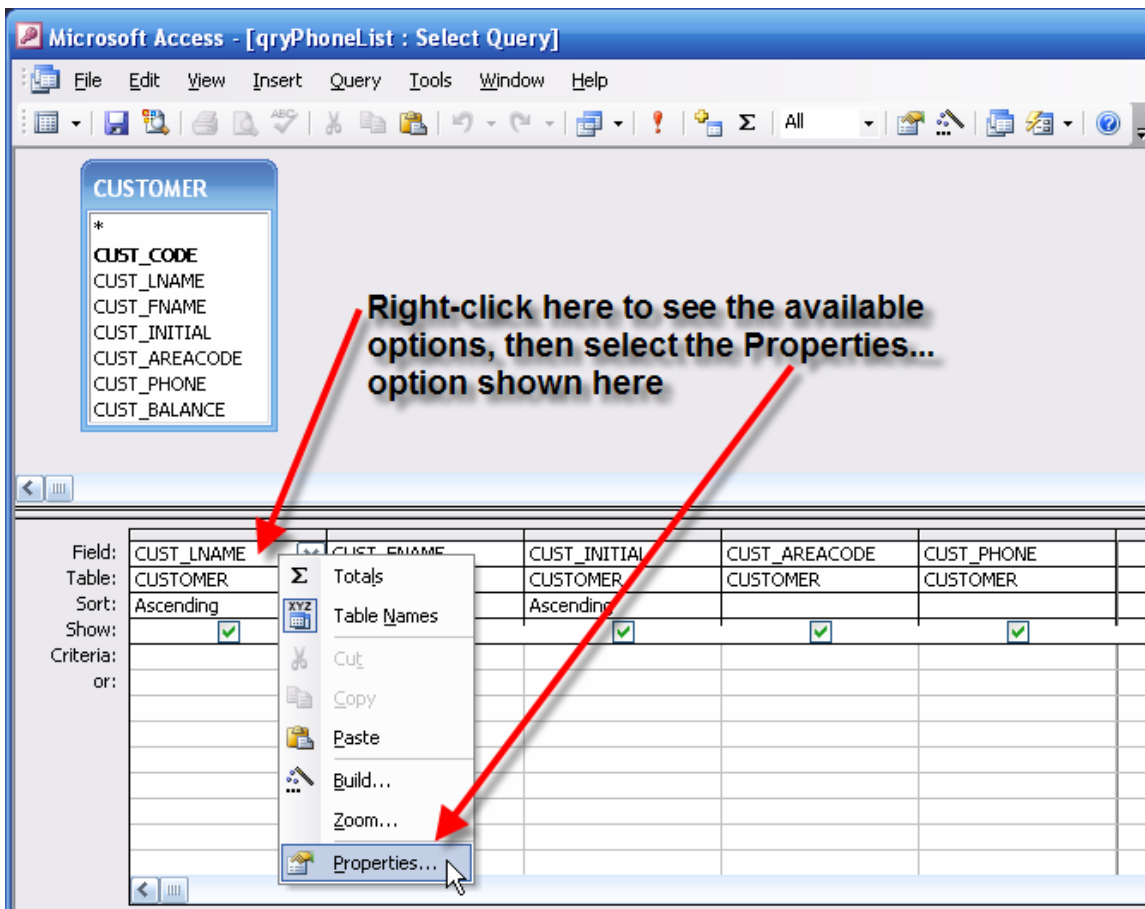
If you are not satisfied with the default font selection when you open the query in its datasheet view, you can easily modify those fonts. Just click on the **Format** button at the top of the screen to open the **Font** dialog box shown in Figure 64. Note that you can select a **Font:**, a **Font Style:**, and a **Size:**. Figure 64 shows the selection of an **Arial** font, a **Regular** font style, and a font size of **8**.

Figure 64 Font Selection



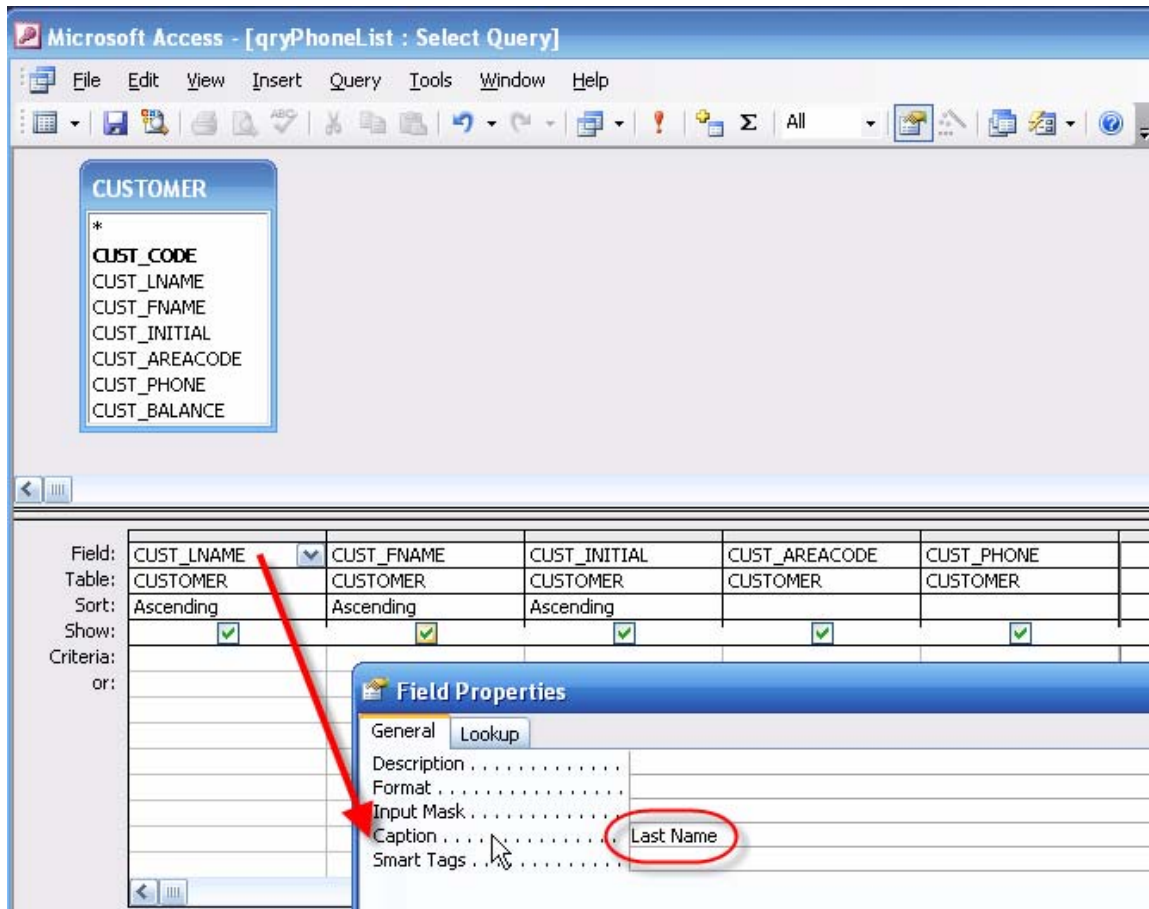
Because the default query output uses the field headers – such as CUST_LNAME – used by the query data source table, you might want to make the presentation more “finished” looking by changing the query field headers. You can get that job done while you are in the query design mode. Note the procedure illustrated in Figure 65. That is, right-click on a selected field name grid cell to see the list of available options, and then click on the **Properties...** option to generate the **Field Properties** dialog box you see in Figure 66.

Figure 65 Selecting the Field Properties Option



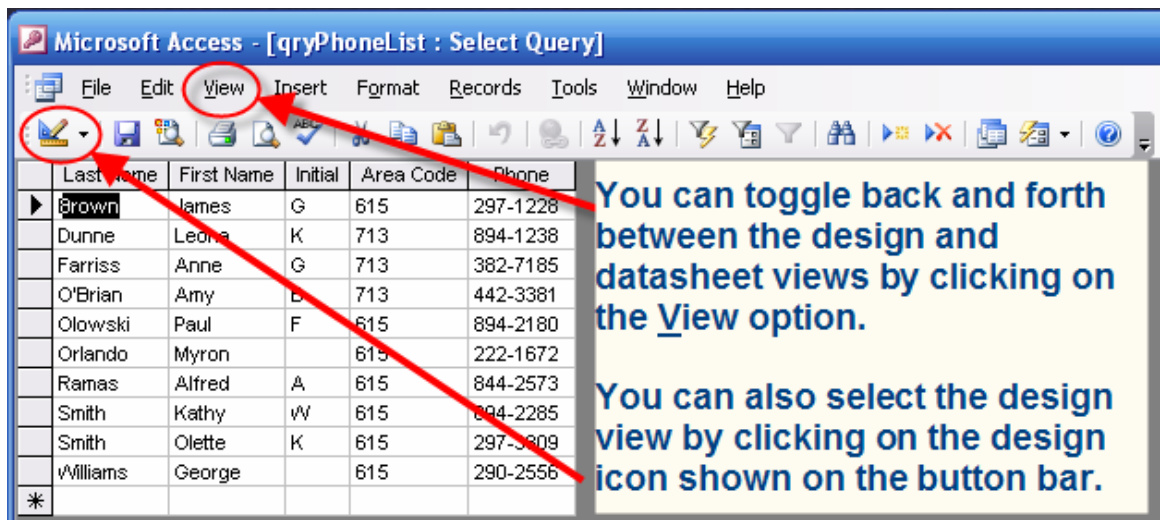
You can change the field header by selecting the **Caption** option in the **Field Properties** dialog box you see in Figure 66, then type in the field header you want to use. In this case, the field header will be **Last Name**. Changing the query header does *not* affect the attribute name (CUST_LNAME) in the query data source – the CUSTOMER table.

Figure 66 Modifying the Field Caption



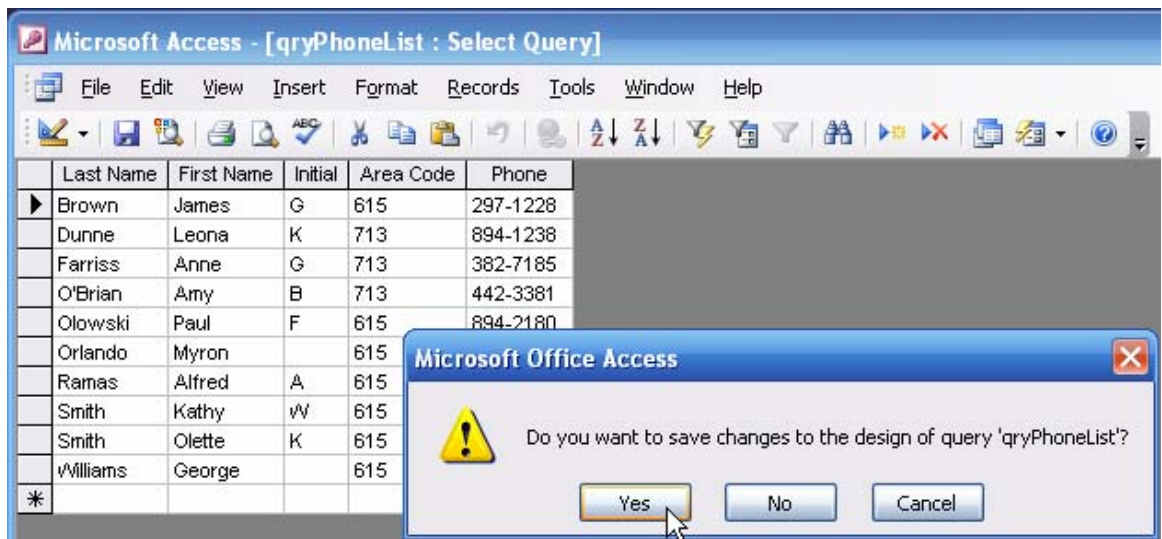
Repeat the editing for the remaining query fields and then open the query in its **Datasheet View** to see the editing results shown in Figure 67.

Figure 67 The Modified Query Field Headers



If you are satisfied with the results, remember to save the query again. (If you forget to save, Access will remind you as shown in Figure 68.)

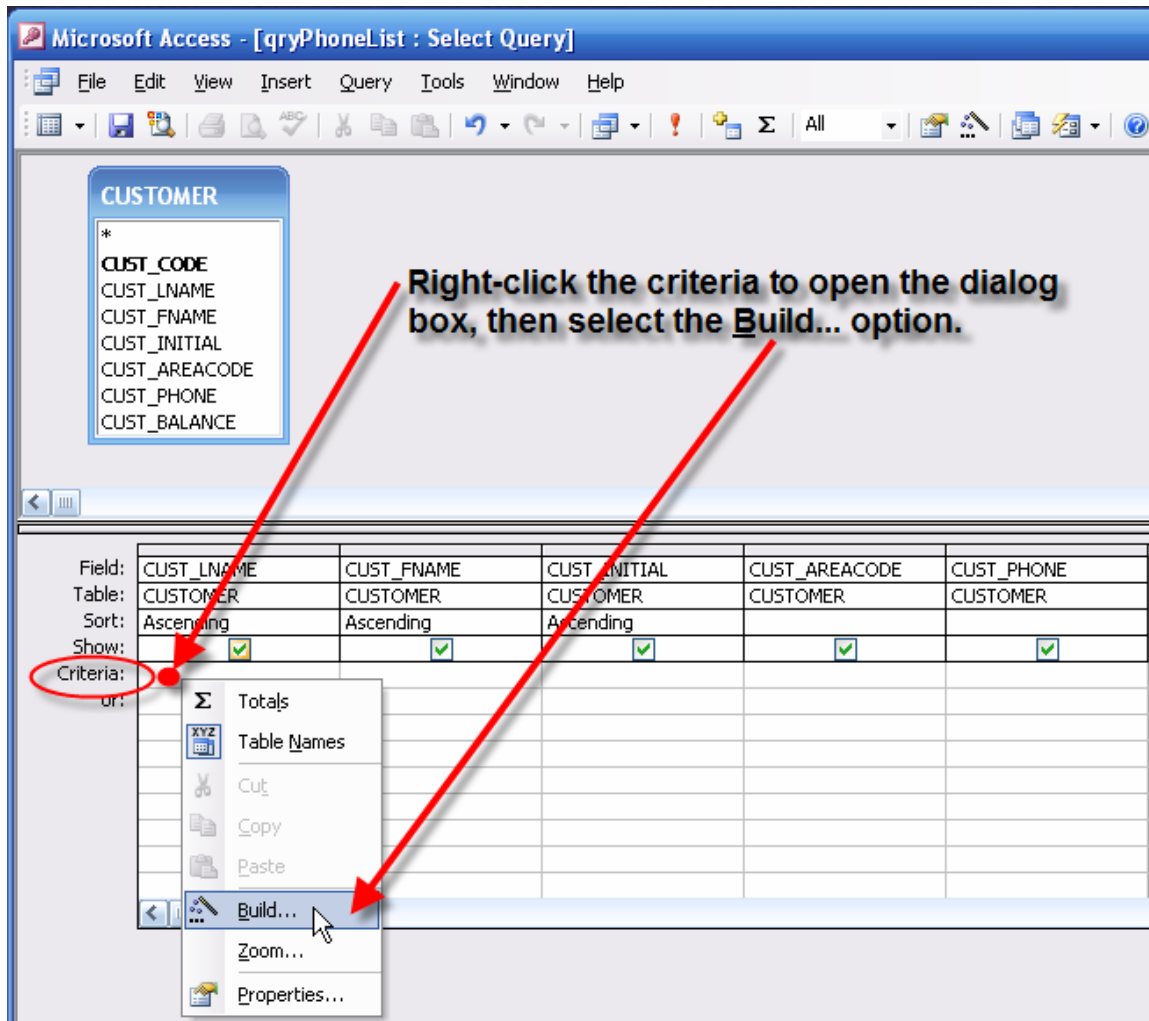
Figure 68 Reminder to Save



2.1.2 Parameter Queries

You can easily create a query in which you specify the *criteria* governing the query output. Note the procedure summarized in Figure 69. In the following example, the objective will be to limit the phone list output to a specified customer’s last name. (A query whose output is limited through specified criteria that restrict output for one or more parameters is also known as a *parameter query*.)

Figure 69 Setting Criteria



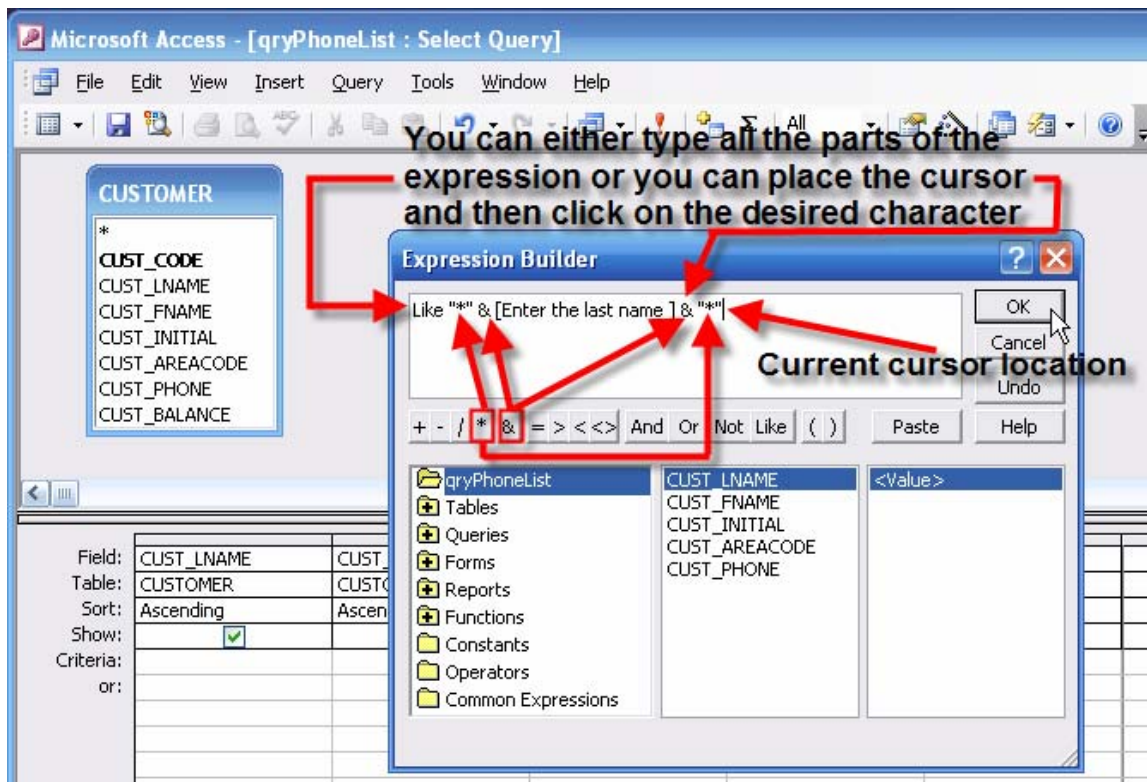
If you want to limit the query phone list output to customers whose last name is “Smith”, you can simply type “Smith” in the CUST_LNAME criteria grid space – marked by the red dot in Figure 69. Unfortunately, that procedure means that the query must be changed each time a different last name limitation is required. You will have a much more flexible query if you let the end user specify the last name restriction through a dialog box. Such a dialog box is created automatically if you type **Like “*” & [Enter customer last name] & “*”** in the CUST_LNAME criteria grid space. (Review Chapter 7, “Introduction to Structured Query Language (SQL)”, Section 7.4.4, to review the LIKE operator and wildcard characters such as “*”).

Note

Although you can type the simple criteria restriction directly into the grid, you should become used to the **Expression Builder**. This tool is especially useful when you later try to enter more complex criteria or even simple criteria with multiple components. In fact, you will discover the Section 5, “Macros,” that you will have a difficult time typing the sometimes long character strings without making errors ... it will be a lot easier to select items from a list than to do the typing. Therefore, we will use the expression builder in most examples.

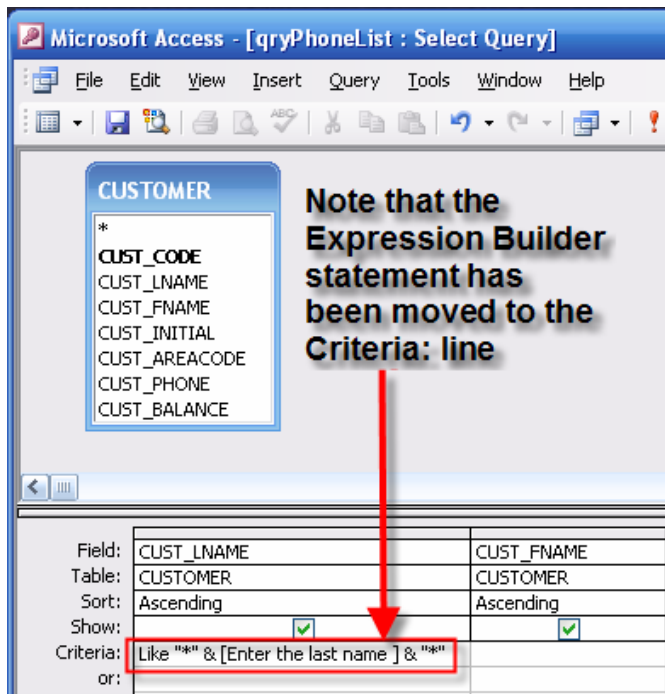
If you click on the **Build...** option shown in Figure 69, you will see the **Expression Builder** dialog box shown in Figure 70. You can either type the entire expression or you can type **Like “**, click on the *****, type **“**, click on the **&** ... and so on. (You do not, of course, type the **“** symbol you see at the end of the expression line ... that’s just the shape of the cursor in the **Expression Builder**.) Although you can type the ***** and **&** symbols easily enough, go ahead and practice using the expression builder’s features while the expressions are very simple. Familiarity breeds, well, competence!

Figure 70 The Expression Builder



When you have completed the expression shown in Figure 70, click **OK** to close the **Expression Builder** and to transfer the expression to the QBE grid as shown in Figure 71. (If you want to see the entire expression in the grid space, drag the grid space limit to widen it, as was done in Figure 71.)

Figure 71 The Completed Criteria Line



Next, open the query in its datasheet view. The expression you see in Figure 71 will trigger the input request you see in Figure 72. Type the last name **Smith** to generate the output shown in Figure 73. Incidentally, the parameter search is not case-sensitive. Therefore, it goes not matter whether you type **SMITH**, **smith**, **Smith**, or any other combination of lower- and upper-case letters. Also, keep in mind that the use of the ***** wildcard character in combination with **Like** will yield these results:

Input	Output
None. (You just tap the Enter key, instead of typing a character and then tapping the Enter key.)	All records.
The letter s .	All records corresponding to a customer whose last name includes the letter s . For example, Ramas , Williams , Olowski , Farriss , and Smith would all be included.
The letters br .	All records corresponding to a customer whose last name includes the letters br . For example, customer O'Brian and Brown would be included.

Figure 72 Name Selection

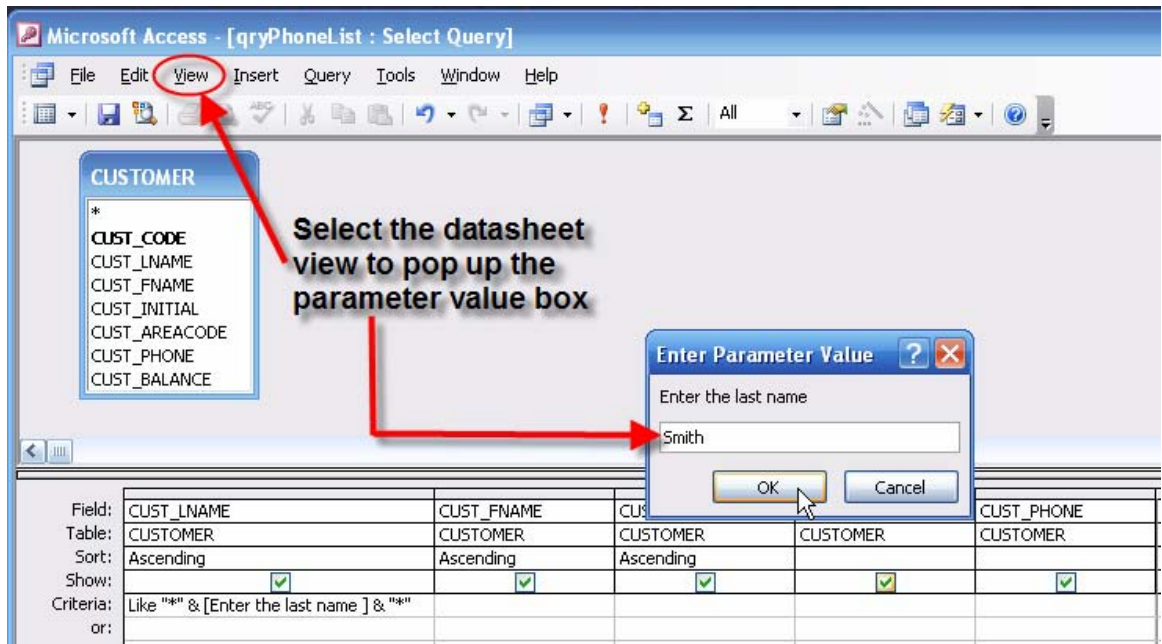
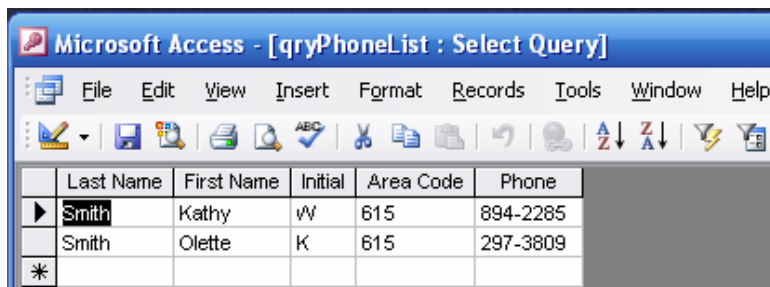
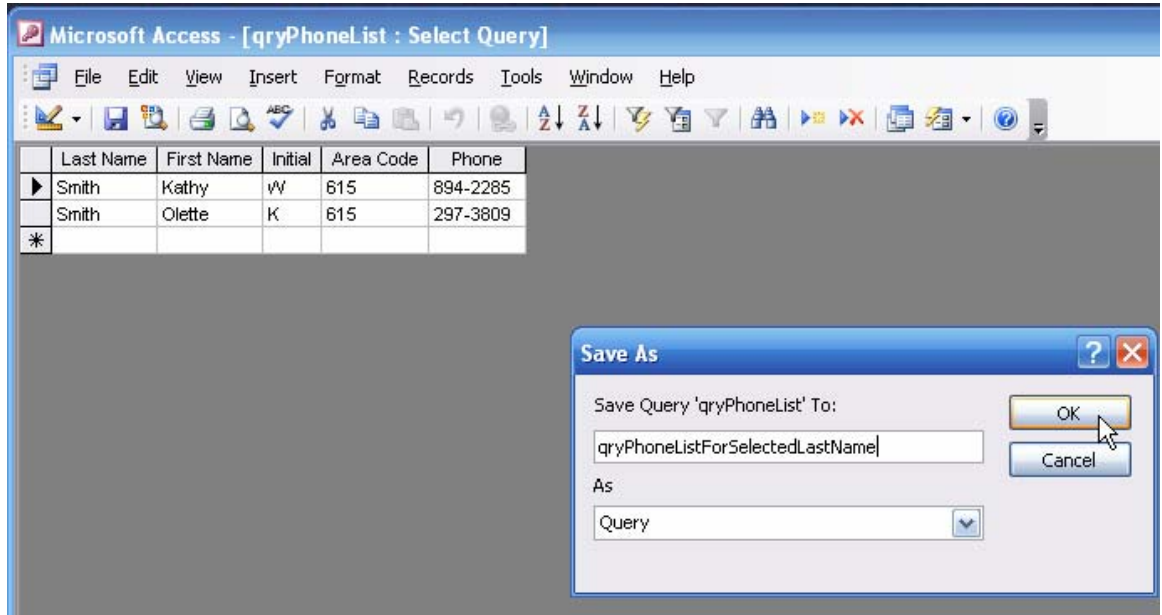


Figure 73 Phone Numbers for Selected Name



Using the **Save As** option to save a file, save the **qryPhoneList** query you have just modified as **qryPhoneListForSelectedLastName**. (See Figure 74. Naturally, if you have maximized the screen, you will only see the **Save As** dialog box.)

Figure 74 Saved Parameter Query



You can use the same technique to limit output by any selected criteria for any field. For example, Figure 75 shows a query that limits its output by VEND_CODE values that are null. (In short, the output will show all products that do not have a known vendor.) The output is shown in Figure 76.

Figure 75 Products without Vendors

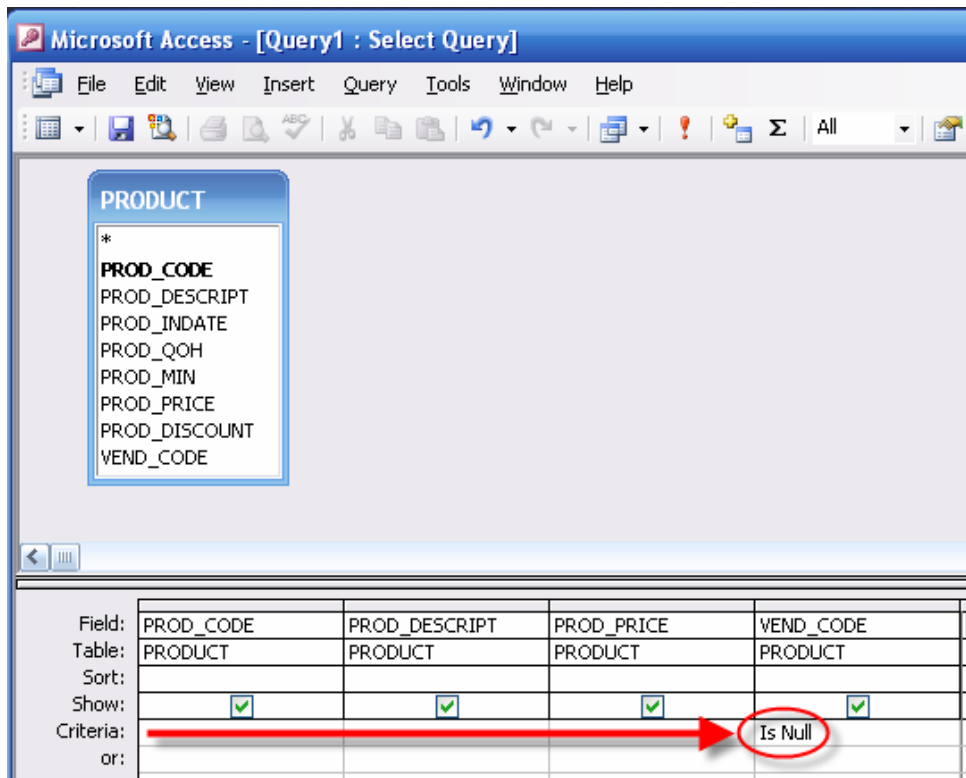


Figure 76 Products without Vendors Output

PROD_CODE	PROD_DESCRIPT	PROD_PRICE	VEND_CODE
23114-AA	Sledge hammer, 12 lb.	\$14.40	
PVC23DRT	PVC pipe, 3.5-in., 8-ft	\$5.87	
*		\$0.00	0

Note that the query was saved as **qryProductsWithoutVendors**.

2.1.3 Multiple Table Queries

You can include any number of tables in your queries. For example, note that the query in Figure 77 includes two tables, CUSTOMER and INVOICE. You do not have to relate the tables, because that was done earlier via the relationships window. (See Section 1.3.)

Figure 77 Multiple Table Query

Note the data source

Field:	CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	INV_NUMBER	INV_DATE	INV_TOTAL
Table:	CUSTOMER	CUSTOMER	CUSTOMER	CUSTOMER	INVOICE	INVOICE	INVOICE
Sort:							
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:							
or:							

Save this query as **qryCustomerInvoices** and then open this query to see its output. (See Figure 77a. To save space, only the first three records are shown.)

Figure 77a The qryCustomerInvoices Output

	CUST_CODE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	INV_NUMBER	INV_DATE	INV_TOTAL
	10011	Dunne	Leona	K	1002	16-Mar-06	\$10.78
	10011	Dunne	Leona	K	1004	17-Mar-06	\$37.66
	10011	Dunne	Leona	K	1008	17-Mar-06	\$431.08

2.1.4 Querying a Query

Suppose you want to know the total sales for each of the customers. If you run the **qryCustomerInvoices** query, you will see all the invoices for each of the customers. For example, if you look at Figure 77a, you see that customer 10011, Leona Dunne, has three invoices for \$10.78, \$37.66, and \$431.08, respectively. The sum of these invoice totals will be \$479.52. How do you write a query that will generate the sum of all the invoice totals for each of the customers? The answer turns out to be simple: As you can see in Figure 78, you can write a query that uses the **qryCustomerInvoices** query output as its data source. (Note that the **Show Table** dialog box shows that the **Queries** tab was selected.)

Figure 78 Querying a Query

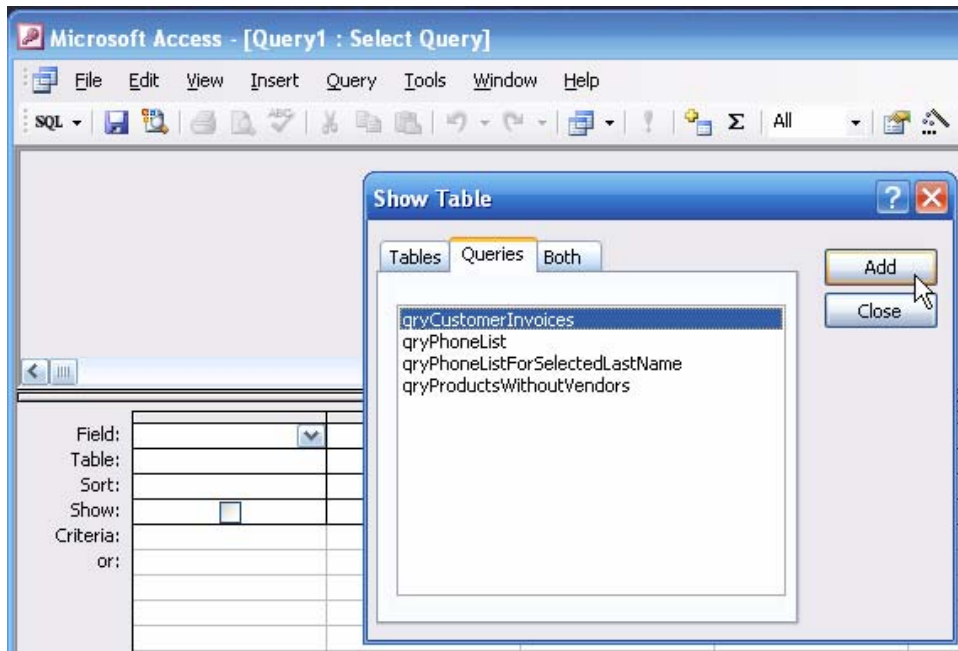


Figure 79 shows the selected fields. It also shows that, after the fields have been dragged and dropped to their **Field:** spaces. Finally, it shows the selection of the **Totals** button at the top of the screen. (The button, circled in red, shows the summation symbol.)

Figure 79 Invoice Totals by Customer

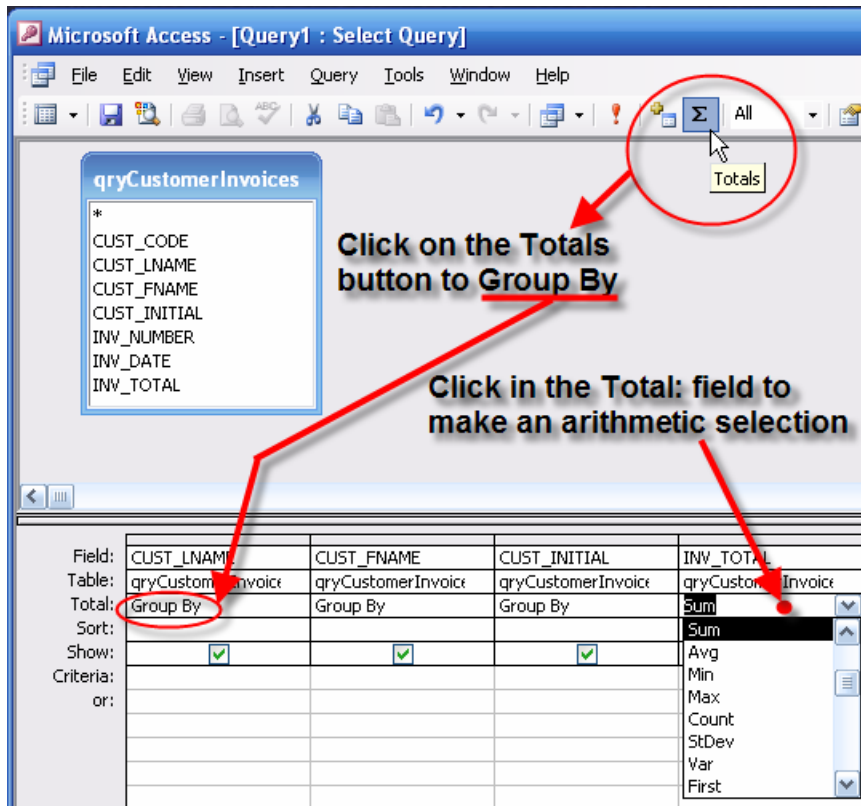


Figure 79 also shows the remaining actions required to complete the query design:

- Click on the **Totals** button to insert the **Group By** entry into all the **Total:** spaces.
- Click on the INV_TOTAL field space to produce the drop-down list.
- Select the **Sum** option from the list

Now check the completed query in its **Datasheet View** to generate the results shown in Figure 81. Note that the \$479.52 sum for customer Leona Dunne is correct. (Customer Leona Dunne’s customer number is 10011. If you take a look at the INVOICE table contents displayed in Figure 80A, you will see that customer 10011 has invoice totals of \$10.78, \$37.66, and \$411.08. The sum of these three invoice totals is \$479.52.)

Figure 80 Customer Invoice Totals

CUST_LNAME	CUST_FNAME	CUST_INITIAL	SumOfINV_TOTAL
Dunne	Leona	K	\$479.52
Farriss	Anne	G	\$76.08
O'Brian	Amy	B	\$37.77
Orlando	Myron		\$456.60
Smith	Kathy	vV	\$166.16

Figure 80A INVOICE Table Entries for Customer 10011

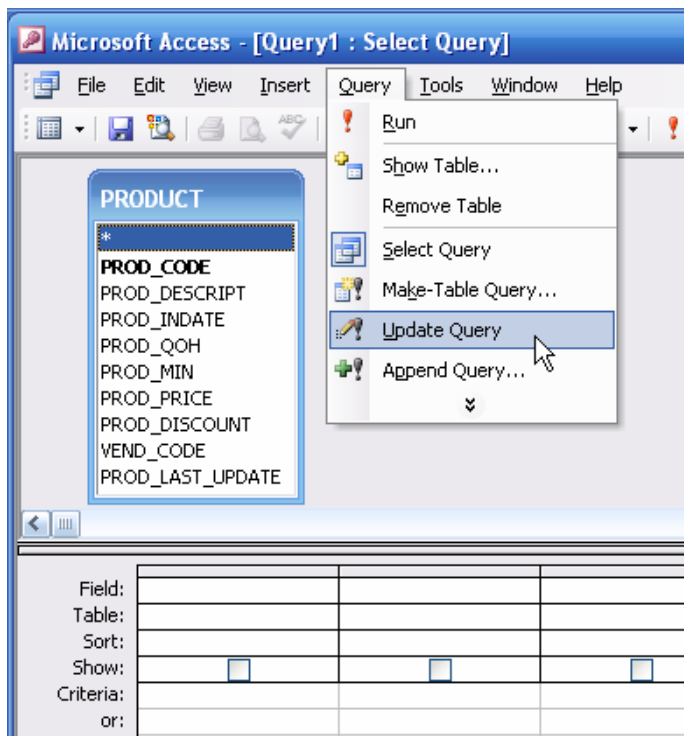
	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
▶ +	1001	10014	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
+	1002	10011	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
+	1003	10012	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
+	1004	10011	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
+	1005	10018	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
+	1006	10014	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
+	1007	10015	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
+	1008	10011	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
+	1009	10019	27-Mar-06	\$180.74	\$14.46	\$195.20	\$0.00	\$0.00
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

Save the query as **qryCustomerInvoiceTotals** to complete the query design process.

2.1.5 Update Queries

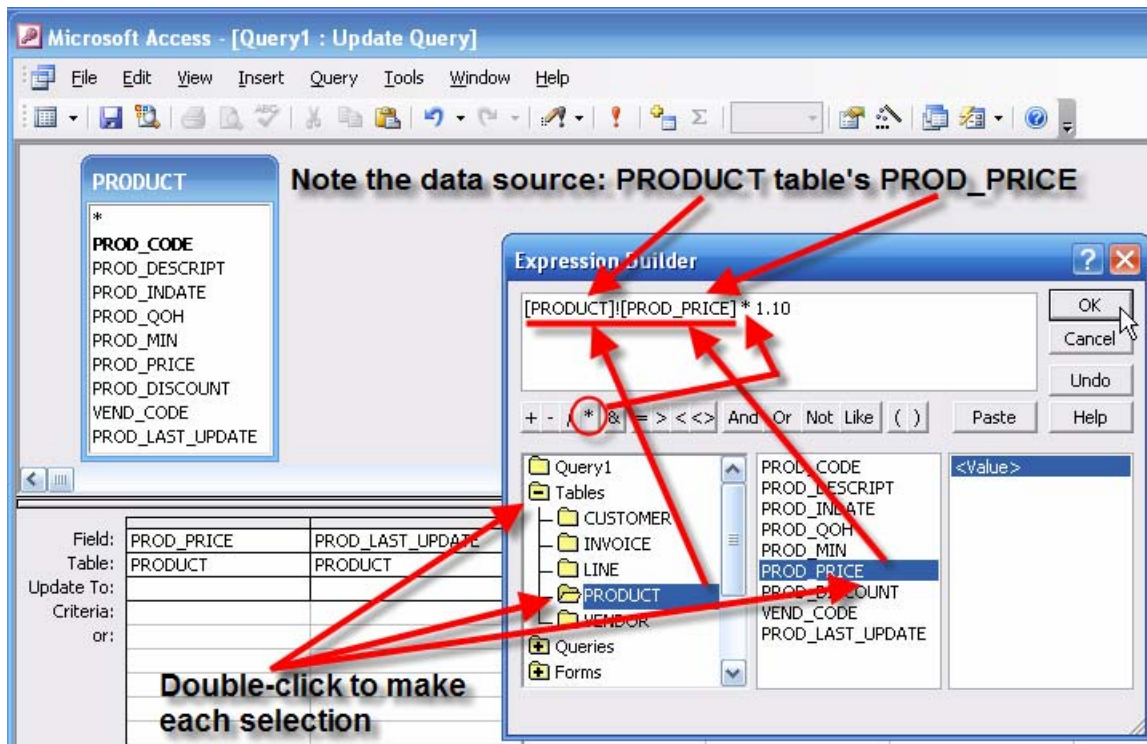
An update query, as its name suggests, is used to update one or more attributes (fields) in a table. To illustrate the development and use of a simple update query, let's use one to alter the PRODUCT table's product price for a selected vendor. Because it is common to record the date of the update, first add a new PRODUCT table attribute named PROD_LAST_UPDATE. Use a time/date data type and a short date format. Note that this attribute shows up in Figure 81. Next, start a new query, select the PRODUCT table, and then click on the **Query** button to generate the dialog box shown in Figure 81. Note that the **Update Query...** has been marked for selection.

Figure 81 Selecting the Update Query Type



Next, drag and drop the PROD_PRICE and PROD_LATE_UPDATE to the field locations as shown in Figure 82. There will be two updates: The PROD_PRICE will be increased by 10% for a selected vendor and the current date of the update will be recorded. As you can tell by looking at Figure 82, the **Expression Builder** was selected to produce the price change. (Although these updates are simple enough to type into the **Update To:** space directly, use the expression builder on all of the updates to make sure that the expression builder becomes a thoroughly familiar tool. (You will also have to update the table to enter the current date into the PROD_LATE_UPDATE field and to create a parameter entry for the selected vendor.)

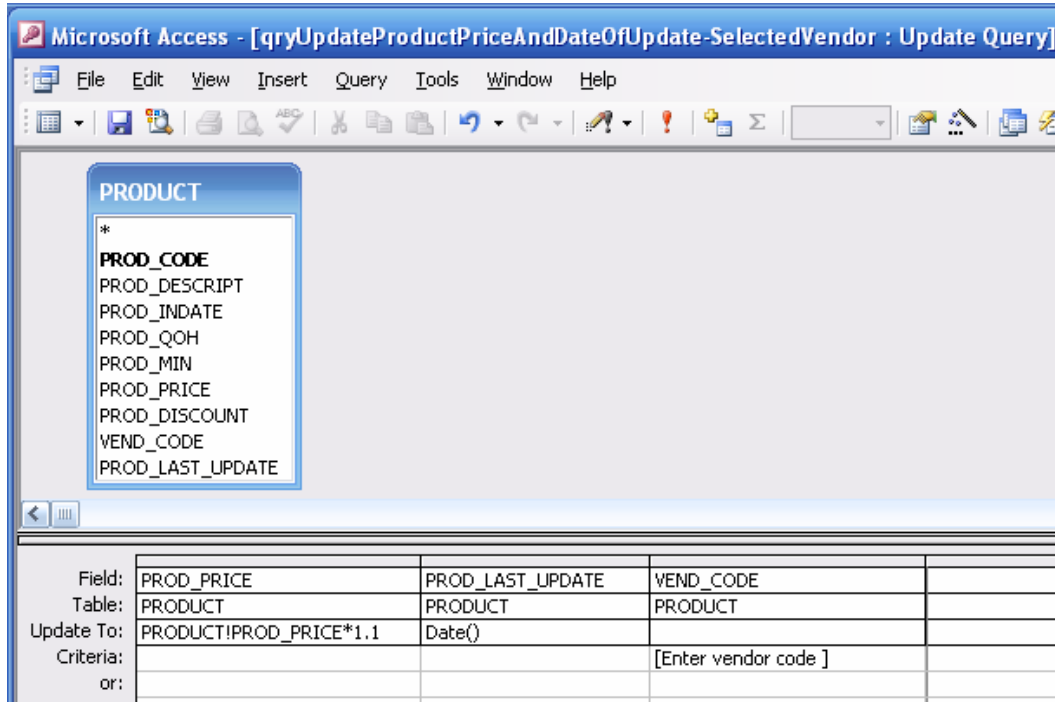
Figure 82 Using the Expression Builder for the Update



As you examine the PROD_PRICE update statement in Figure 82, note that the asterisk indicates a multiplication. You can either type the asterisk or you can click on the asterisk button shown in Figure 82. Type the value 1.10 to indicate the 10% increase.

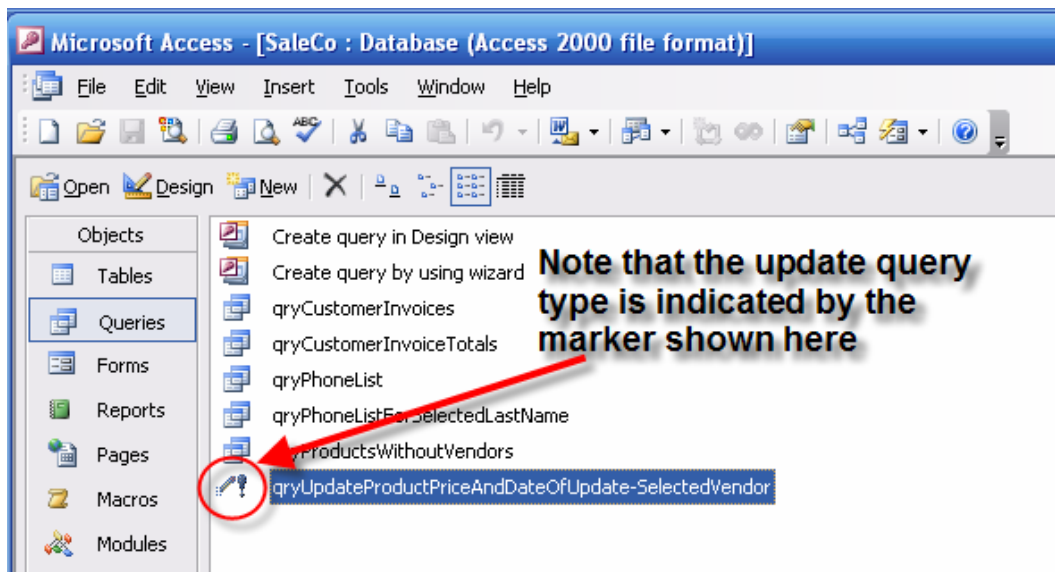
Because the update is probably too broad – it includes all vendors – modify the update query by adding a parameter entry that will generate a dialog box. Figure 83 shows the modification that will yield a dialog box that will request the vendor code for the update. Note also that the date of the update is included. (The system date is provided by the **Date()** function.)

Figure 83 Completed Update Requirements



Save the query as **qryUpdateProductPriceAndDateOfUpdate-SelectedVendor** and then close the query. When you click on the **Queries** option in the **Objects** column, you will see the new query listed as shown in Figure 84. (Note that the update query is identified by a special marker.)

Figure 84 Update Query Marker



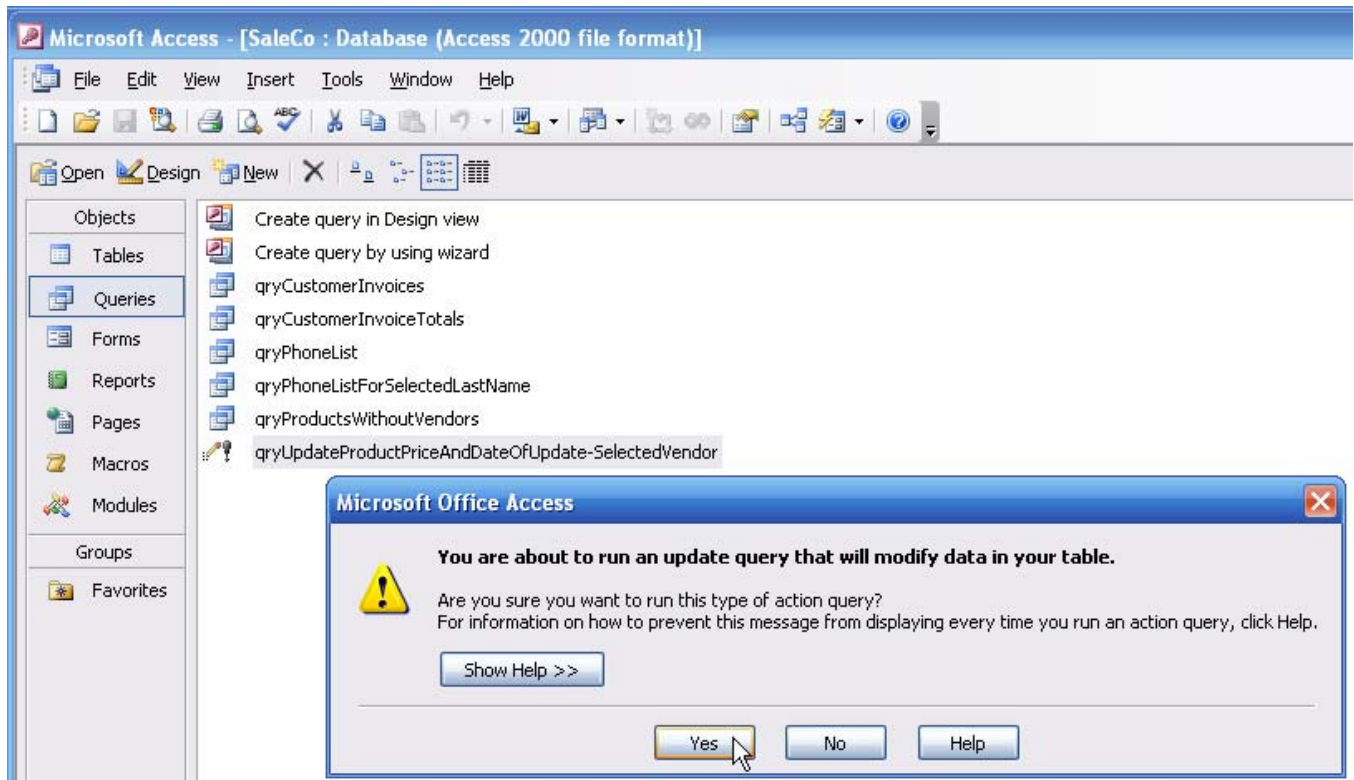
Before you run the update query you have just created and saved, take a look at the current prices for vendor 21344, shown in Figure 85.

Figure 85 Prices for Vendor 21344 before Update

	PROD_CODE	PROD_DESCRIPT	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_CODE	PROD_LAST_UPDATE
▶	1QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-05	8	5	\$109.99	0.00	25595	
+	13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-05	24	15	\$14.99	0.05	21344	
+	14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-05	18	12	\$17.49	0.00	21344	
+	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-06	14	8	\$39.95	0.00	23119	
+	1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-06	23	5	\$43.99	0.00	23119	
+	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-05	7	5	\$109.92	0.05	24288	
+	2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-05	6	5	\$99.87	0.05	24288	
+	2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-06	11	5	\$38.95	0.05	25595	
+	23109-HB	Claw hammer	20-Jan-06	18	10	\$9.95	0.10	21225	
+	23114-AA	Sledge hammer, 12 lb.	02-Jan-06	8	5	\$14.40	0.05		
+	54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-05	37	20	\$4.99	0.00	21344	
+	89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-06	11	5	\$256.99	0.05	24288	
+	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-06	171	75	\$5.87	0.00		
+	SM-18277	1.25-in. metal screw, 25	01-Mar-06	169	75	\$6.99	0.00	21225	
+	SW-23116	2.5-in. wd. screw, 50	24-Feb-06	237	100	\$8.45	0.00	21231	
+	WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-06	15	5	\$119.95	0.10	25595	
*				0	0	\$0.00	0.00	0	

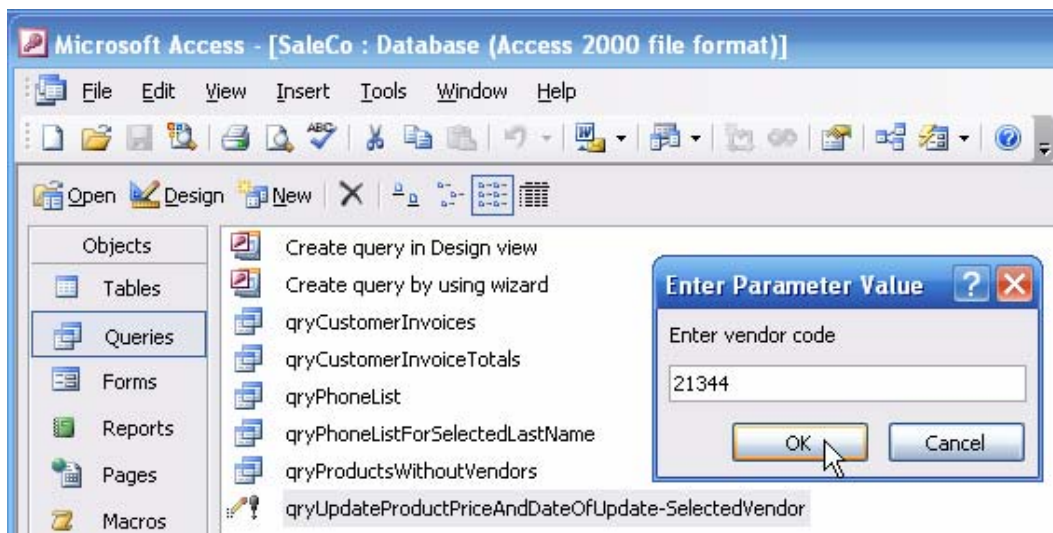
Next, run the [qryUpdateProductPriceAndDateOfUpdate-SelectedVendor](#) and note the effect of doing so. Because the update query modifies the table contents, Access will give you several opportunities to change your mind. Figure 86 shows the first of the warnings.

Figure 86 Update Query Warning



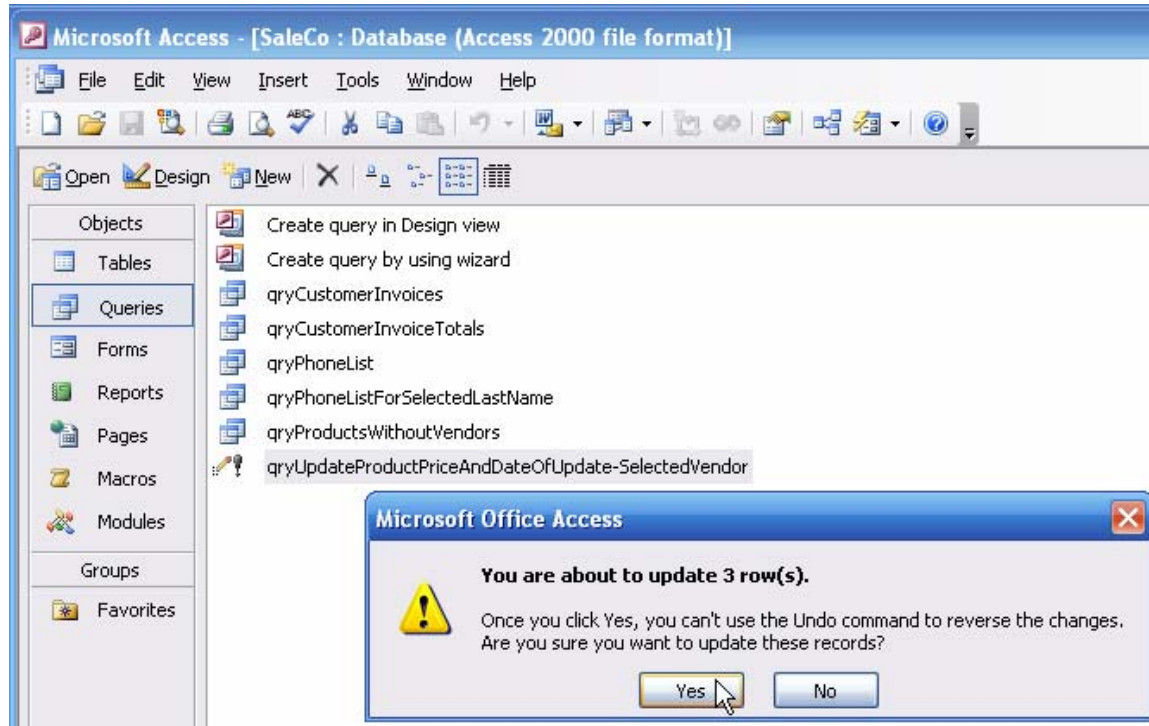
If you click on the Yes button shown in Figure 86, your update query will generate the parameter request through the dialog box you see in Figure 87. Note that the vendor code **21344** has been entered to match the display in Figure 85.

Figure 87 Parameter Value Request



As soon as you click on the **OK** button shown in Figure 87's parameter request, the update query will be launched. But before it actually updates the table, Access gives you one final warning and a way to back out of the update. Note that the warning shown in Figure 88 shows that you will be updating three rows. (If you check Figure 85 again, you'll see that this is the correct number of rows.)

Figure 88 Final Update Warning



If you now click the **Yes** button shown in Figure 88, the update query will make the requested updates. Open the **PRODUCT** table after running the update query and check the results. Figure 89 shows that the changes were made as requested.

Figure 89 Prices for Vendor 21344 after the Update

	PROD_CODE	PROD_DESCRPT	PROD_INDATE	PROD_QOH	PROD_MIN	PROD_PRICE	PROD_DISCOUNT	VEND_CODE	PROD_LAST_UPDATE
▶	1QER/31	Power painter, 15 psi., 3-nozzle	03-Nov-05	8	5	\$109.99	0.00	25595	
+	13-Q2/P2	7.25-in. pwr. saw blade	13-Dec-05	24	15	\$16.49	0.05	21344	27-Mar-06
+	14-Q1/L3	9.00-in. pwr. saw blade	13-Nov-05	18	12	\$19.24	0.00	21344	27-Mar-06
+	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Jan-06	14	8	\$39.95	0.00	23119	
+	1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-06	23	5	\$43.99	0.00	23119	
+	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-05	7	5	\$109.92	0.05	24288	
+	2232/QWE	B&D jigsaw, 8-in. blade	24-Dec-05	6	5	\$99.87	0.05	24288	
+	2238/QPD	B&D cordless drill, 1/2-in.	20-Jan-06	11	5	\$38.95	0.05	25595	
+	23109-HB	Claw hammer	20-Jan-06	18	10	\$9.95	0.10	21225	
+	23114-AA	Sledge hammer, 12 lb.	02-Jan-06	8	5	\$14.40	0.05		
+	54778-2T	Rat-tail file, 1/8-in. fine	15-Dec-05	37	20	\$5.49	0.00	21344	27-Mar-06
+	89-WRE-Q	Hicut chain saw, 16 in.	07-Feb-06	11	5	\$256.99	0.05	24288	
+	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Feb-06	171	75	\$5.87	0.00		
+	SM-18277	1.25-in. metal screw, 25	01-Mar-06	169	75	\$6.99	0.00	21225	
+	SW-23116	2.5-in. wd. screw, 50	24-Feb-06	237	100	\$8.45	0.00	21231	
+	wR3/TT3	Steel matting, 4'x8'x1/8", 5" mesh	17-Jan-06	15	5	\$119.95	0.10	25595	
*				0	0	\$0.00	0.00	0	

Compare the original prices shown in Figure 85 to those shown in Figure 89. Note that the initial price of PROD_CODE = 13-Q2P2 – supplied by vendor 21344 – was \$14.99. The updated price is $\$14.99 * 1.10 = \16.489 , which is properly rounded to \$16.49 because the PROD_PRICE data type *and* format were both recorded as “currency” when the table was created. Also note that the update was made on March 27, 2006. Incidentally, you can even record the time of the update, using the MS Access **Time()** function for a new field named PROD_UPDATE_TIME. You can even record the identity of the person who made the updates through an authorization code that may have been recorded on a scanning device of some sort. By recording the update date, time, and person you can track all changes and assign proper responsibility.

Using Update Queries to Manage Transactions

Using the update query techniques you have just learned, you can manage a sales transaction. For example, suppose that customer 10019 had bought 20 units of product SW-23116 @ \$8.45 each and 2 units of product PVC23DRT @ \$5.87 each on March 27, 2006. Let's suppose you have already entered the following INVOICE table values:

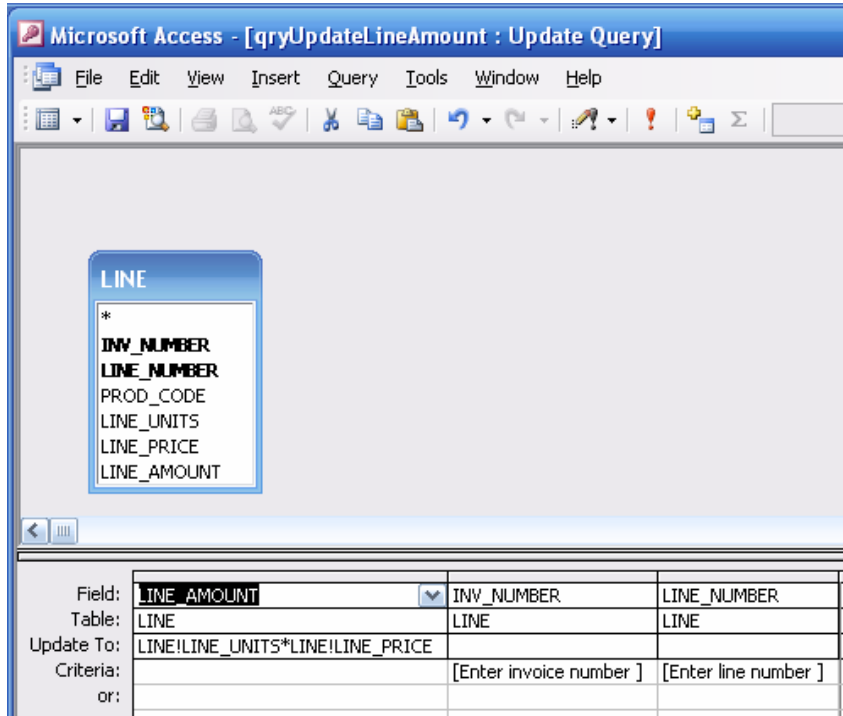
- INV_NUMBER = 1009
- CUST_CODE = 10019
- INV_DATE = 27 March 2006. (Actually, you could use an update query to enter the current date of the transaction, but go ahead and enter the date manually to keep the process simple.)

Next, open the LINE table to record the following two sets of LINE table entries. (Note that you are not entering the LINE_AMOUNT values.)

- INV_NUMBER = 1009
- LINE_NUMBER = 1
- PROD_CODE = SW-23116
- LINE_UNITS = 20
- LINE_PRICE = \$8.45. (Actually, you could have used an update query to copy the PROD_PRICE from the PRODUCT table to the LINE table, but go ahead and enter the price manually to keep the process simple.)
- INV_NUMBER = 1009
- LINE_NUMBER = 2
- PROD_CODE = PVC23DRT
- LINE_UNITS = 2
- LINE_PRICE = \$5.87.

After completing the just-described tasks, create an update query to calculate and enter the LINE_AMOUNT value for invoice 1009 and line numbers 1 and 2. The correct update query is shown in Figure 90. (You did remember to use the **Expression Builder**, didn't you?)

Figure 90 Update Query to Calculate LINE_AMOUNT



When you run the query shown in Figure 90, you will be prompted to enter the invoice number 1009 and the line number 1 and the query will properly execute to produce the first line total. Next, run the query again, this time entering the invoice number 1009 and the line number 2. When you are done, open the LINE table to see if the amounts were calculated properly. Figure 91 indicates that they were.

Figure 91 The Calculated LINE_AMOUNT Values

INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE_UNITS	LINE_PRICE	LINE_AMOUNT
1001	1	13-Q2/P2	1	\$14.99	\$14.99
1001	2	23109-HB	1	\$9.95	\$9.95
1002	1	54778-2T	2	\$4.99	\$9.98
1003	1	2238/QPD	1	\$38.95	\$38.95
1003	2	1546-QQ2	1	\$39.95	\$39.95
1003	3	13-Q2/P2	5	\$14.99	\$74.95
1004	1	54778-2T	3	\$4.99	\$14.97
1004	2	23109-HB	2	\$9.95	\$19.90
1005	1	PVC23DRT	12	\$5.87	\$70.44
1006	1	SM-18277	3	\$6.99	\$20.97
1006	2	2232/QTY	1	\$109.92	\$109.92
1006	3	23109-HB	1	\$9.95	\$9.95
1006	4	89-WRE-Q	1	\$256.99	\$256.99
1007	1	13-Q2/P2	2	\$14.99	\$29.98
1007	2	54778-2T	1	\$4.99	\$4.99
1008	1	PVC23DRT	5	\$5.87	\$29.35
1008	2	wR3/TT3	3	\$119.95	\$359.85
1008	3	23109-HB	1	\$9.95	\$9.95
1009	1	SW-23116	20	\$8.45	\$169.00
1009	2	PVC23DRT	2	\$5.87	\$11.74
*	0	0	0	\$0.00	\$0.00

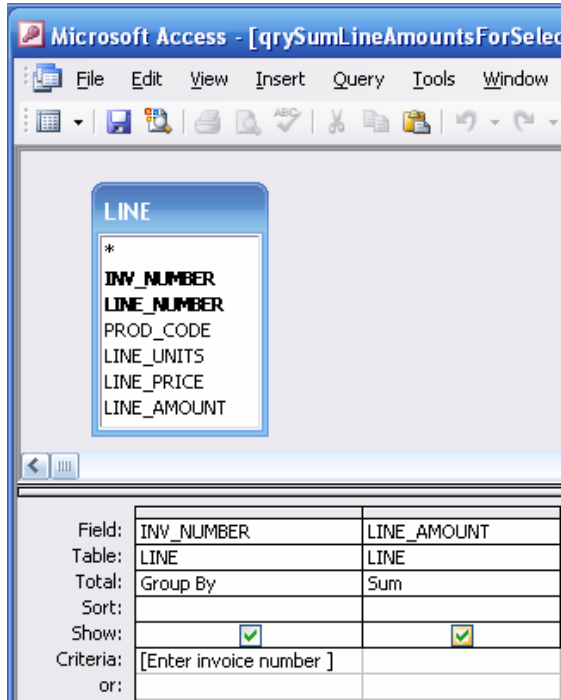
You can use additional update queries to calculate the invoice subtotal, the tax, and the total. Then enter the amount paid, \$100.00, and then use an update query to calculate the balance due and the customer balance. At this point, the INVOICE table's INV_NUMBER 1009 shows no entry for any of the values that are yet to be calculated. (See Figure 92.)

Figure 92 The INVOICE Table values before the Updates

INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
1001	10014	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
1002	10011	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
1003	10012	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
1004	10011	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
1005	10018	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
1006	10014	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
1007	10015	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
1008	10011	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
1009	10019	27-Mar-06	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00
*	0	0	\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

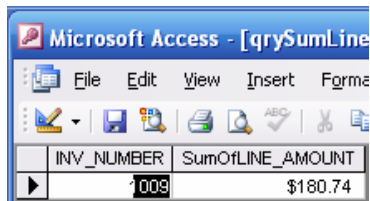
Keep in mind that you cannot get a total for a set of values in a query and then use those values during the same query run. For example, to get the sales for invoice 1009, you will have to add \$169.00 and \$11.74 and then use that sum to update the INVOICE table's INV_AMOUNT. Figure 93 shows a simple parameter query to generate the sum of the line amounts for the selected invoice number. (Note that this query used the sum function to get its job done. *This is not an update query.*)

Figure 93 Sum the Line Amounts



Save the query shown in Figure 93 as **qrySumLineAmountsForSelectedInvoice** and run it to check its execution. Figure 94 indicates that the query performed its job properly. (Note that the sum of the last two LINE_AMOUNT values in Figure 91 should be \$169.00 + \$11.74 = \$180.74.)

Figure 94 The Correct Line Amount Sum



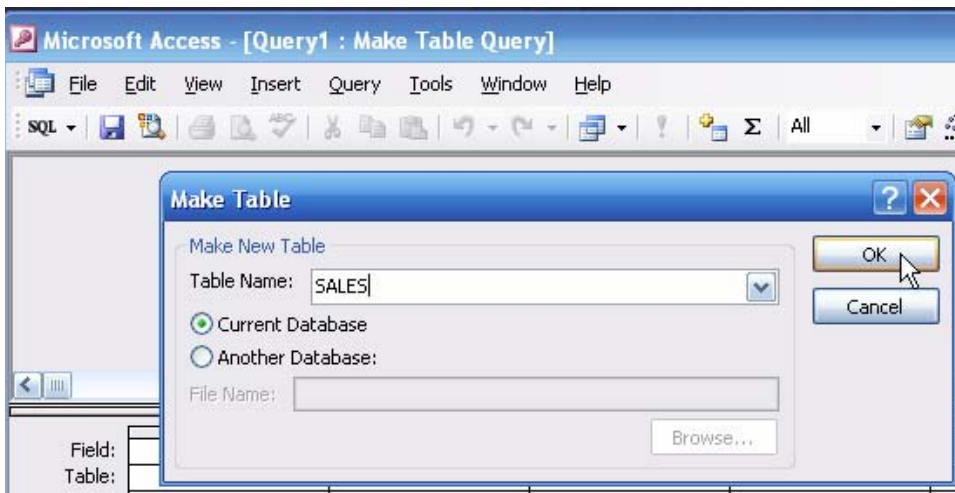
There are many ways to enter the \$180.74 value into the INVOICE table. (Although you can create an update query that uses the results of the parameter query in Figure 93 to update the INVOICE table, you will learn in Section 5.1 how to use some simple, but more effective ways to get that job done.) However, since this section deals with various query types, let's take a look at some other queries that will turn out to be useful.

2.1.6 Make Table Queries

One of the very useful MS Access features is its ability to create new tables based on other tables or even other queries. (You should recall from the text’s Chapter 7, “The Data Warehouse,” that temporary tables may be used as a data source for the data warehouse applications.)

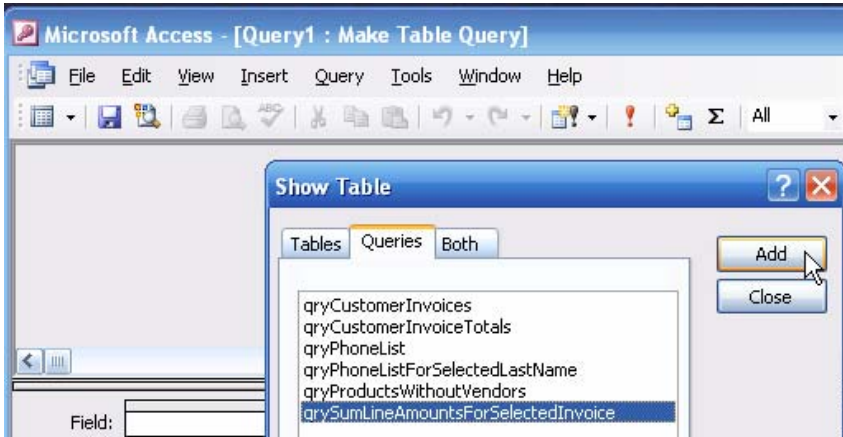
You should now be familiar with the basic query development routine. Therefore, to create a **Make Table** query, start as you did before. That is, first select the query design option. . If the **Show Table** dialog box is shown on the screen, close it and then use the **Query** button at the top of the screen to produce the query option list. Select the **Make Table Query...** option to generate the **Make Table** dialog box you see in Figure 95. You will create a SALES table to store the results of running the **qrySumLineAmountsForSelectedInvoice** you saw in Figures 93 and 94, so name the new table SALES, as shown in Figure 95.

Figure 95 Make Table Query



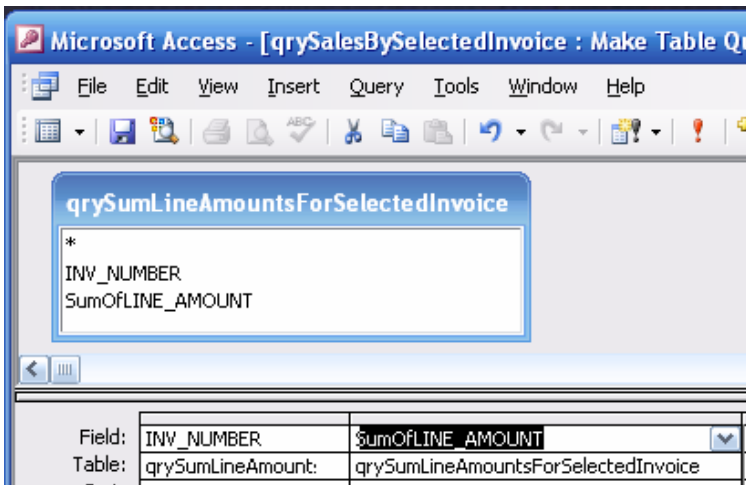
Next, select the data source for the new table as shown in Figure 96. In this case, the data source will be the **qrySumLineAmountsForSelectedInvoice** query, so select the **Queries** tab and then select the query from the list.

Figure 96 Selecting the Data Source



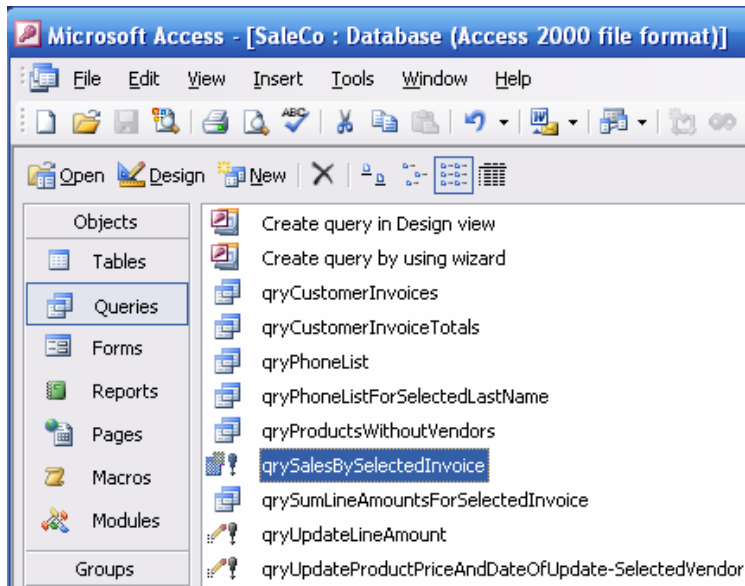
Adding the query will place it on the design screen. Next, drag and drop the query fields on the Field: lines as shown in Figure 97. Make sure that you save the just-completed **Make Table** query. (As you can tell by looking at Figure 97, the query has been saved as **qrySalesBySelectedInvoice**.)

Figure 97 Completed Make Table Query



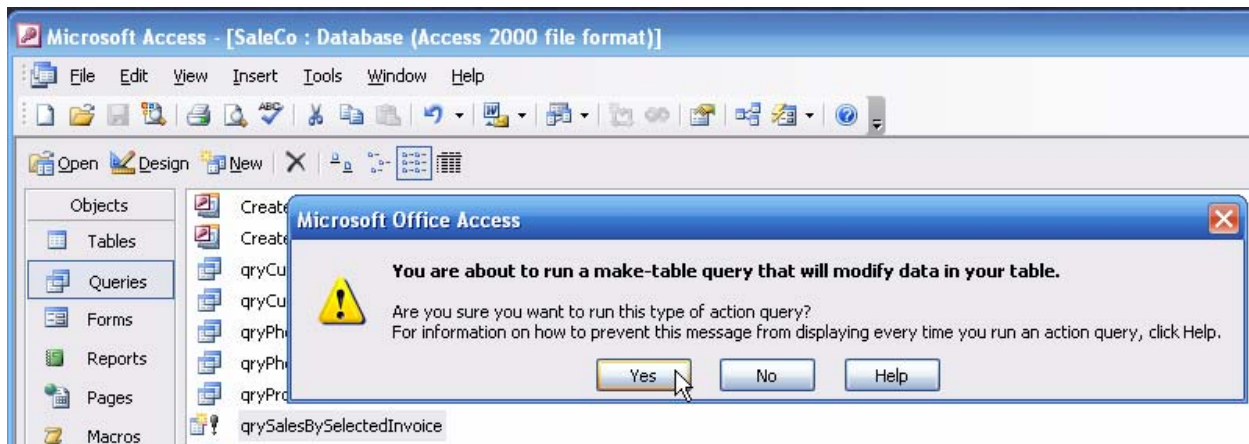
Note that the updated query list in Figure 98 includes the new query.

Figure 98 Current Query List



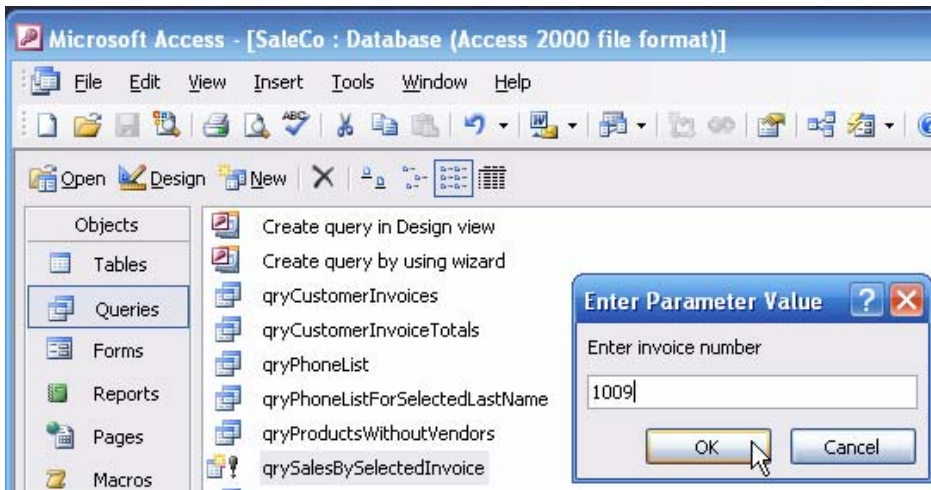
When you run the new **qrySalesBySelectedInvoice** query, Access will alert you to the fact that this query will modify data in a table. (See Figure 99. The new SALES table does not yet contain any data, but that is about to change.)

Figure 99 First Make Table Warning



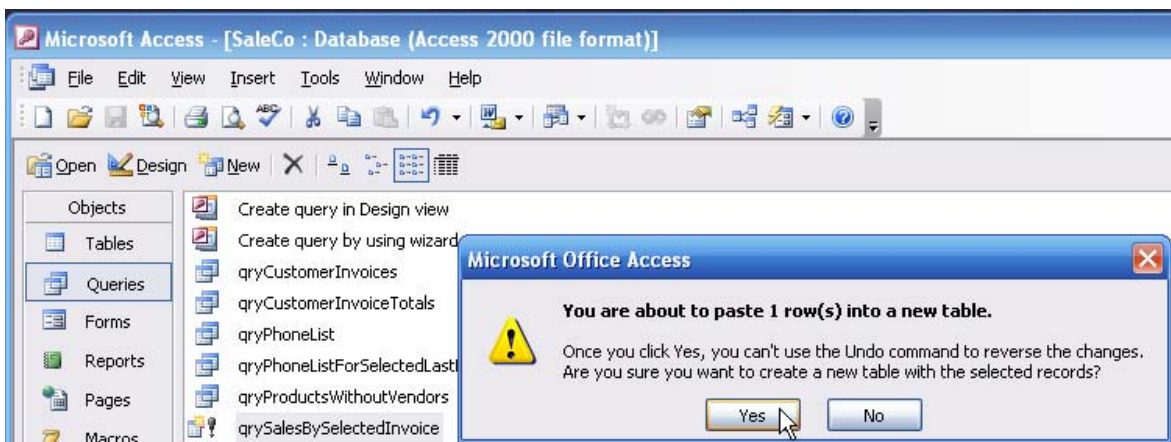
If you accept the fact that the query will modify the data in the new SALES table, click on the **Yes** button shown in Figure 99 to generate the invoice number entry request in Figure 100.

Figure 100 Invoice Number Selection



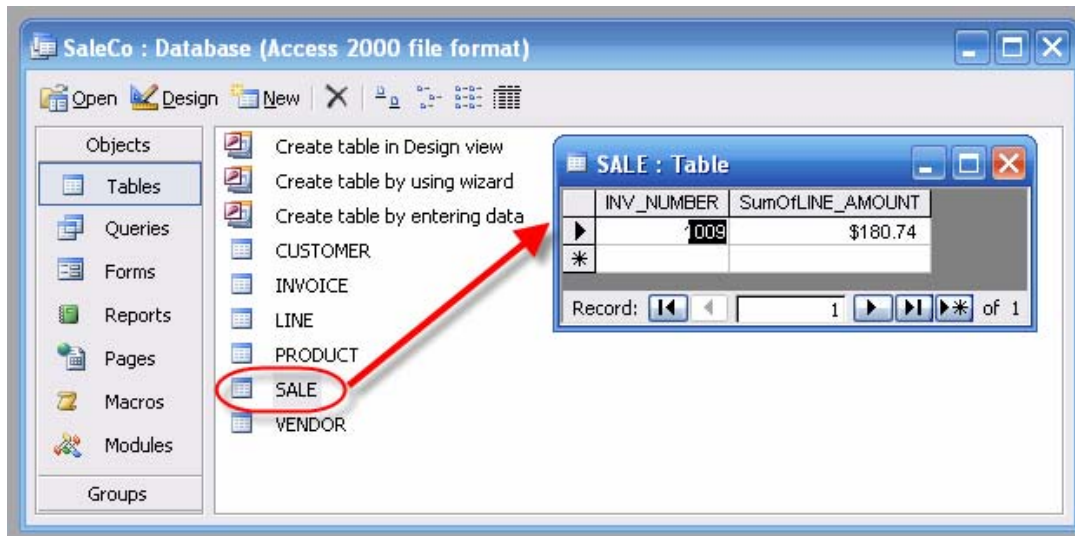
Enter the invoice number 1009 as shown in Figure 100 to produce the warning shown in Figure 101. Click the **Yes** button to complete the **Make Table** action.

Figure 101 Make Table Warning



The results are shown in Figure 102. Note that the SALES table has been added to the list of tables and that the table contents show the invoice number 1009 and the sales (invoice) amount \$180.74.

Figure 102 The New Sales Table



Now that you have a good (SALES table) data source, which is the sum of the invoice lines for INV_NUMBER 1009 in the INVOICE table, you can easily create an update query to update the INVOICE table's INV_AMOUNT. Note that Figures 103 and 104 show the required update query structure and the results of running that query.

Figure 103 Update Invoice Amount Query

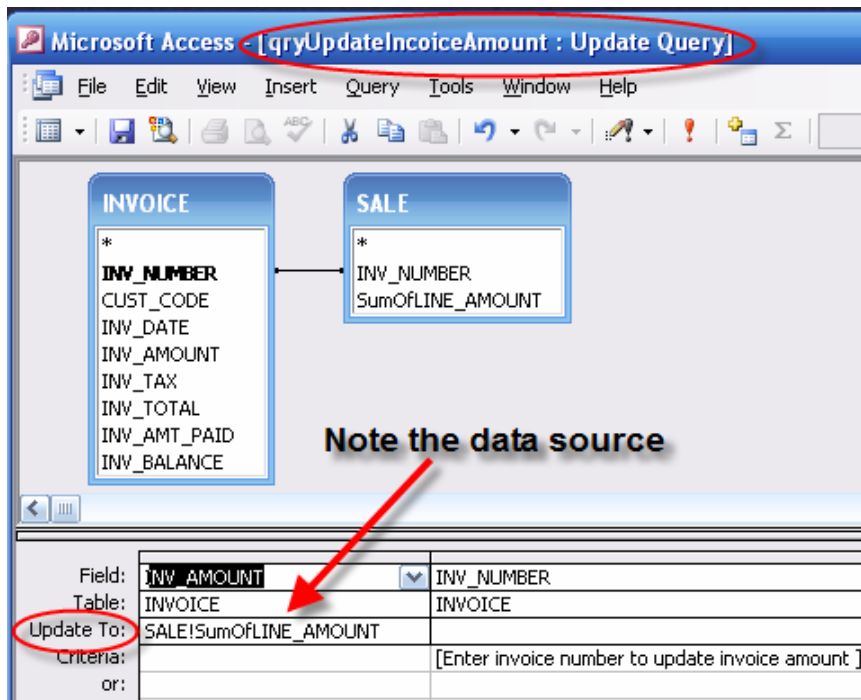


Figure 104 Updated Invoice Amount

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
▶ +	1001	10014	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
+	1002	10011	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
+	1003	10012	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
+	1004	10011	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
+	1005	10018	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
+	1006	10014	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
+	1007	10015	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
+	1008	10011	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
+	1009	10019	27-Mar-06	\$180.74	\$0.00	\$0.00	\$0.00	\$0.00
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

2.1.7 Append Queries

Append queries do what their name suggests. That is, the append data to an existing table. That feature is very useful, because – as you saw in the preceding example -- the new SALES table contents are likely to be used to store the invoice amounts that can be used to update the INVOICE table’s INV_AMOUNT values via an update query.

To create an **Append Query**, follow the now familiar new query design steps. However, this time, select the **Append Query** from the query type list. In the following sequence – shown in Figures 105 and 106 - - you will see how an append query can be used to add data to an existing table. There are two important limitations you should know about when you append data to an existing table:

- For each field, the data type of the data to be appended must be identical to the existing data type of the data in the receiving table.
- For each field, the field name of the data to be appended must be identical to the existing data field name of the data in the receiving table.

Figure 105 Append Query Design

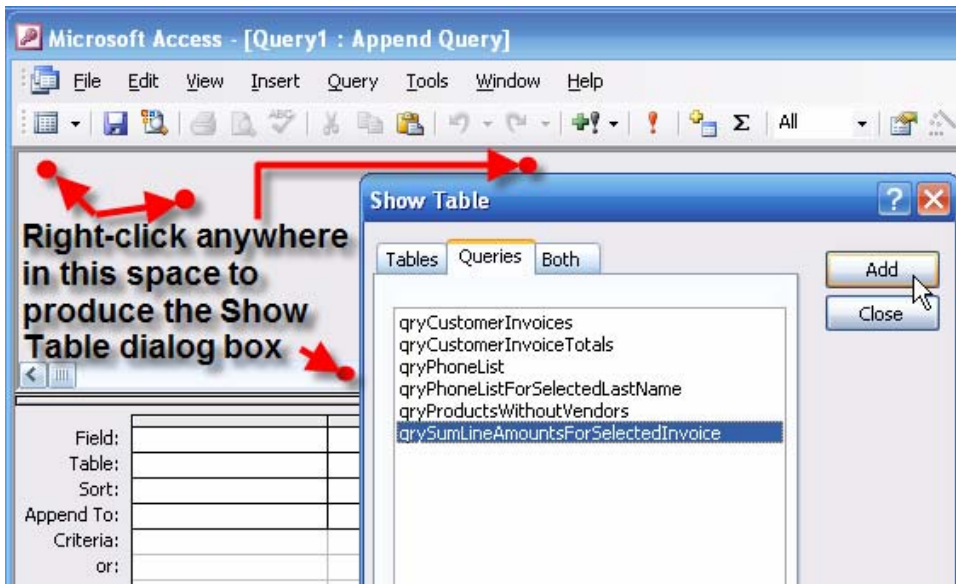
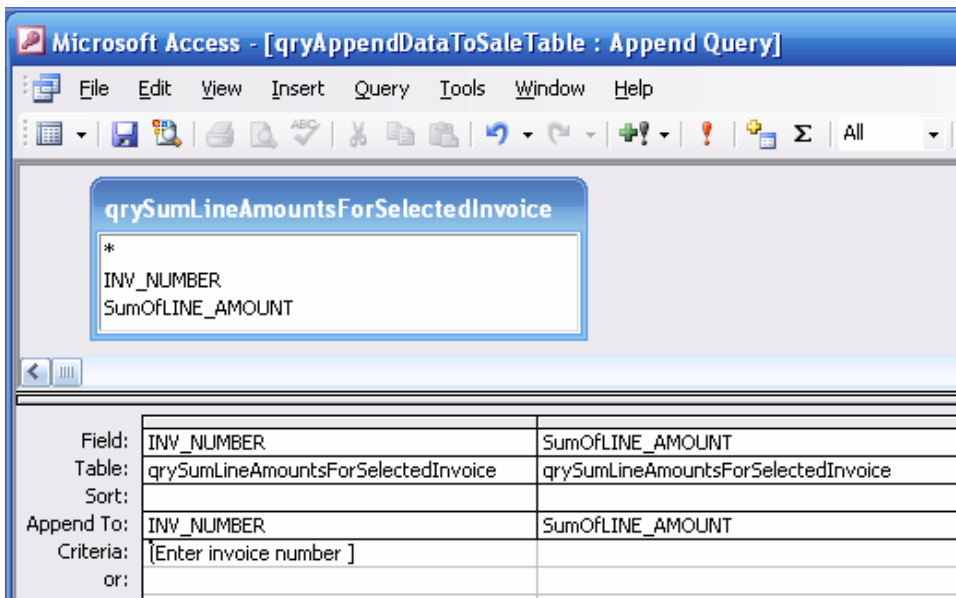
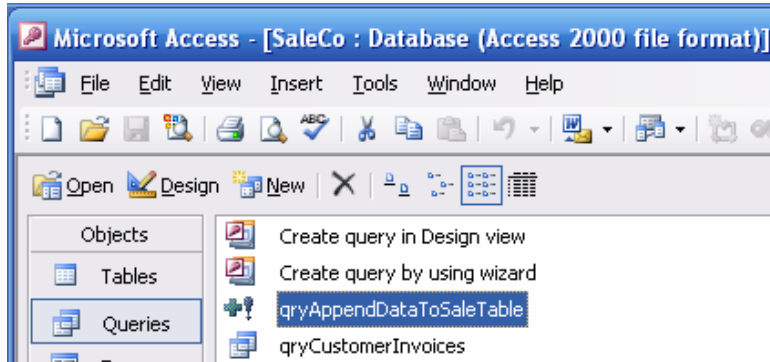


Figure 106 Design View of Completed Append Query



Save the new append query as **qryAppendDateToSaleTable** and then close the query. You will now see the new query as shown in Figure 107.

Figure 107 The Saved Append Query



Running the new append query will cause Access to provide the appropriate warnings and the parameter request as shown in Figures 108, 109, and 110.

Figure 108 First Append Query Action Warning



Figure 109 Parameter Entry

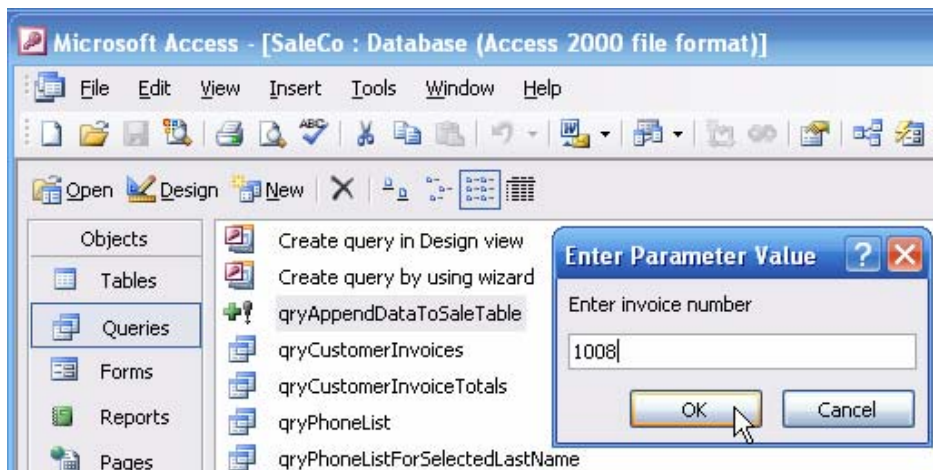
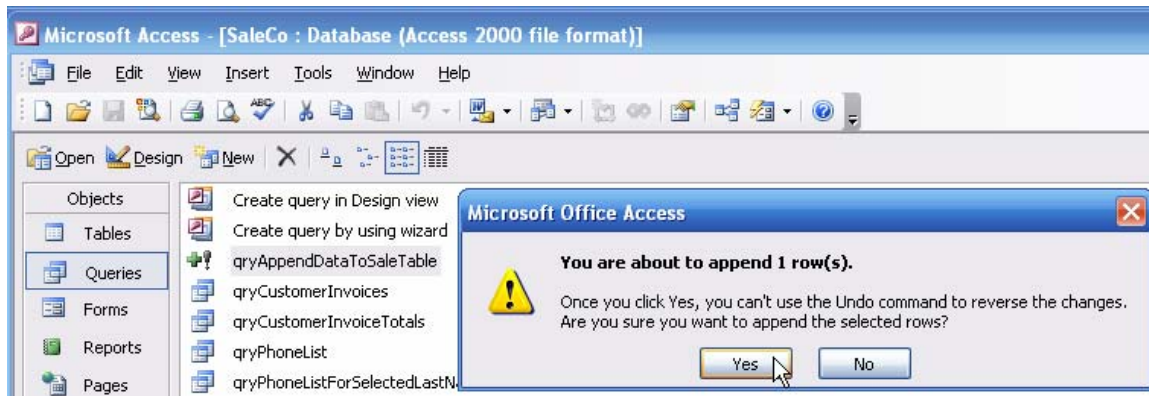
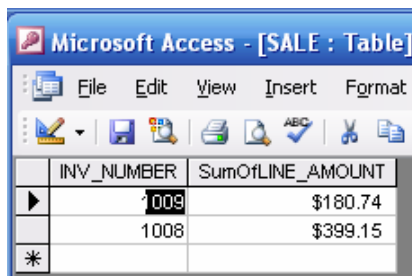


Figure 110 Final Append Action Warning



You can check the results of running the append query by checking the SALE table. Figure 111 shows that the append query performed its work as intended.

Figure 111 SALE Table Append Result



You can now run the [qryUpdateInvoiceAmountQuery](#) again – see Figures 103 and 104 – to enter the INV_AMOUNT into the INVOICE table for invoice 1008. (Actually, that entry was already made manually before as you can see in Figure 39’s INVOICE table, so this \$399.15 value from the SALE table simply replaces that value.)

At this point, you have learned how to use various query types to store, update, and transfer data. Therefore, you have the basic tools at your disposal to help you automate the sales transaction process. For example, you can now create update queries to update the remaining fields in the INVOICE table, to reduce the QOH values in the PRODUCT table, to update the CUST_BALANCE in the CUSTOMER table, and so on. Macros or Visual Basic code can then be used to complete the automation process by executing the queries “behind the scenes.” Figures 112 through 115 show several update queries and their effects.

Figure 112 INVOICE Tax and Total before Update

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
+	1001	10014	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
+	1002	10011	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
+	1003	10012	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
+	1004	10011	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
+	1005	10018	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
+	1006	10014	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
+	1007	10015	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
+	1008	10011	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
+	1009	10019	27-Mar-06	\$180.74	\$0.00	\$0.00	\$0.00	\$0.00
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

Figure 113 Calculate Invoice Tax Update Query

Microsoft Access - [qryCalculateInvoiceTax : Update Query]

Field: **INV_TAX** INV_NUMBER

Table: **INVOICE** INVOICE

Update To: **INVOICE!INV_AMOUNT*0.08**

Criteria: [Enter invoice number]

Figure 114 Calculate Invoice Total Update Query

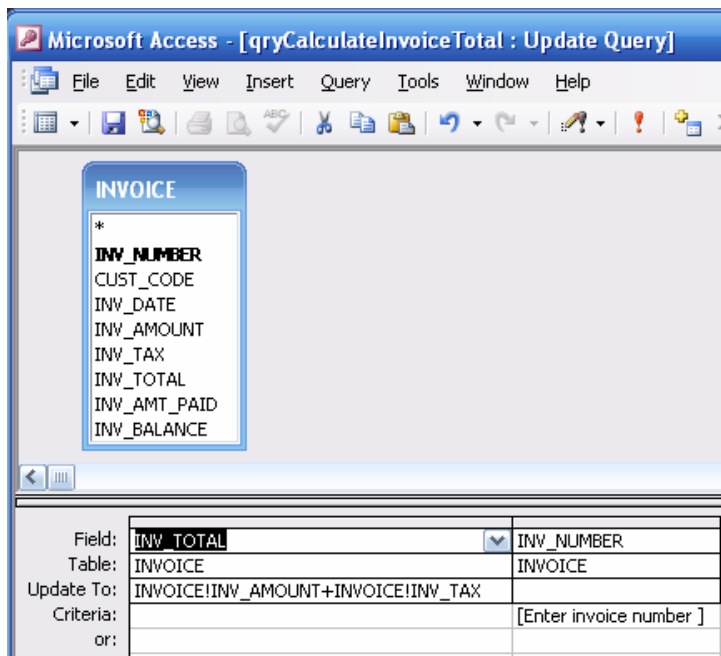


Figure 115 Updated Invoice Tax and Total

Microsoft Access - [INVOICE : Table]

File Edit View Insert Format Records Tools Window Help

	INV_NUMBER	CUST_CODE	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
+	1001	10014	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
+	1002	10011	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
+	1003	10012	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
+	1004	10011	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
+	1005	10018	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
+	1006	10014	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
+	1007	10015	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
+	1008	10011	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
+	1009	10019	27-Mar-06	\$180.74	\$14.46	\$195.20	\$0.00	\$0.00
*	0	0		\$0.00	\$0.00	\$0.00	\$0.00	\$0.00

Keep in mind that all the queries must be run in the proper order. For example, you cannot calculate the sales tax before you have the updated invoice total. And the invoice total cannot be calculated unless all the invoice line totals have been calculated and summed. The table values will become meaningless if the queries were executed in the wrong order. Therefore, it would not be wise to trust humans to remember the proper sequence. Fortunately, you can use programming techniques or macros to automate the process. (You will learn how to create and use macros in section 5.)

Note

Although the update and append queries can be very useful in managing transactions such as invoicing and letting customers make payments on their accounts, there are other techniques available that will perform the transaction management job more efficiently. You will learn how to use a **Set Value** technique to manage customer payments on account and to automate the process through macros in Section 5.

3.1 Forms

While queries let you get data and/or information from the database, forms let you control the presentation format much better. In addition, forms will enable you to control data input and to present the results from multiple queries and/or tables. Forms can also be used to tie the application components together through menus and other devices. In short, forms are the way in which the end user is best connected to the database ... and they provide that “professional” look to your data management efforts.

Forms may be based on tables and/or queries. The simplest and most efficient way to create a form is through a *form wizard*, which lets you automatically generate the code that produces the form. Figure 116 illustrates the use of one of the many form wizards. Note that the form production process in Figure 116 is set in motion through the following sequence:

1. Select the **Forms** option on the left side of the screen.
2. Select the **New** option to produce the **New Form** dialog box.
3. Select the **Autoform: Columnar** option.
4. Select the data source by clicking on the down arrow. This action will produce a drop-down list – not shown here – from which you select the data source. Note the selection of the CUSTOMER table.
5. Click **OK** to create the form shown in Figure 117.

Figure 116 Starting a New Form

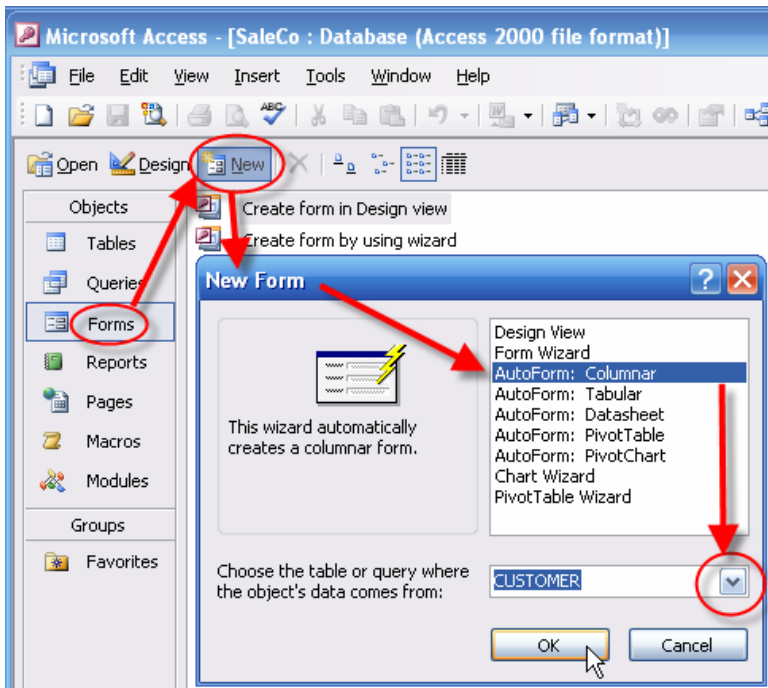
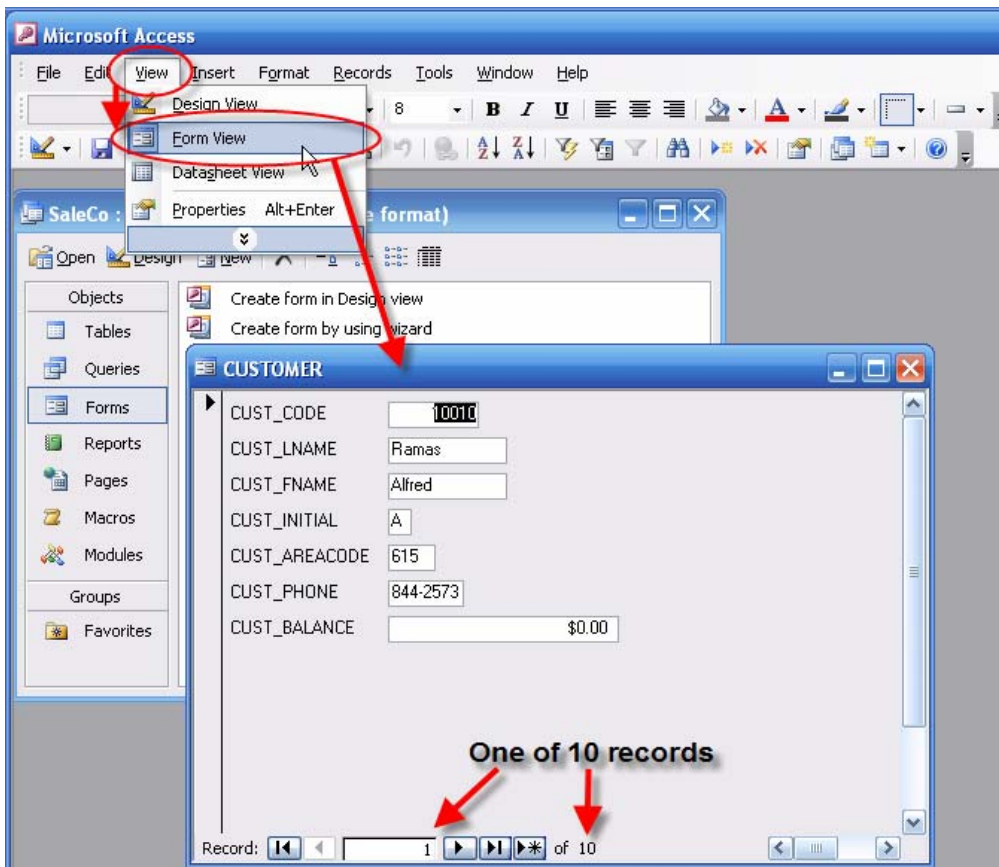


Figure 117 The Access-Generated Customer Form



As you can see by looking at Figure 117, the form opens up in its **Form View** format. (Given the work you have done with the query development, you should be familiar with the various views.)

Save the form before continuing the form design process. Figure 118 shows that the new form was named **frmCUSTOMER**.

Figure 118 Save the New Customer Form



Note

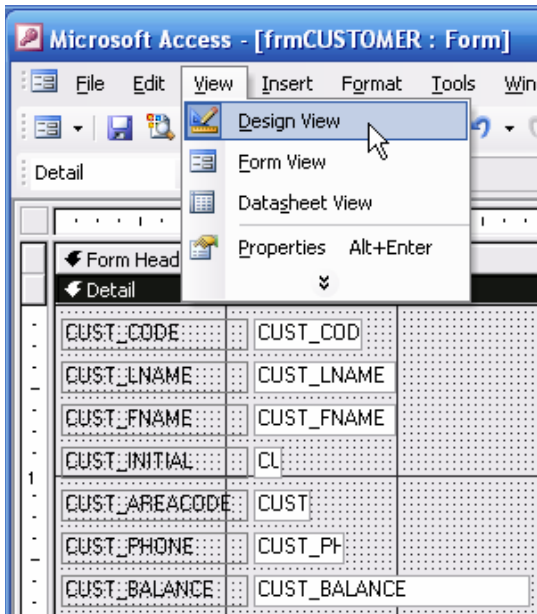
Here is another example of self-documentation: the **frm** prefix indicates that the object is a form and the capital letters used in the **CUSTOMER** portion indicate that the query data source was a *table* named CUSTOMER. If the form's data source had been a *query* named **qryCustomer**, the form name would have indicated the data source through the use of lowercase letters, using "caps" only to separate the components. Therefore, if the form had based on the **qryCustomer** query, the form name would have been **frmCustomer**. In either case, looking at the object name would immediately tell you that the object is a form and that it displays customer information.

*Making it easy to keep track of the many components in a set of database applications is a mark of professionalism. Nobody wins if nobody can figure out what the database objects are or what they do. **Documentation conventions should be made in clear in the formal application documentation.***

3.1.1 Editing the Form

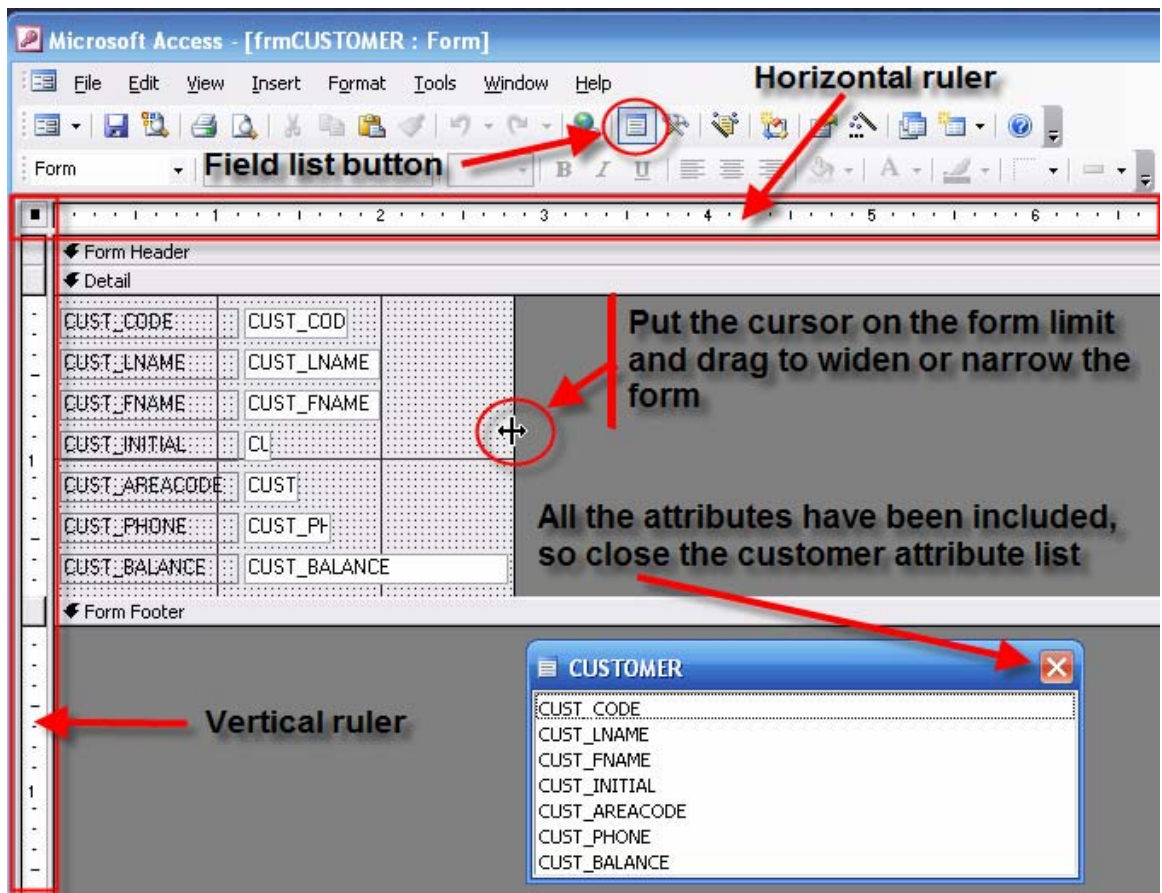
While the **frmCUSTOMER** form presents the data properly, it is a good idea to modify the form's format for eye-appeal reasons. Your expertise is often graded on the basis of how "professional" your forms look. Therefore, put the form in its **Design View** format, as shown in Figure 119.

Figure 119 Change the Customer Form to Design View



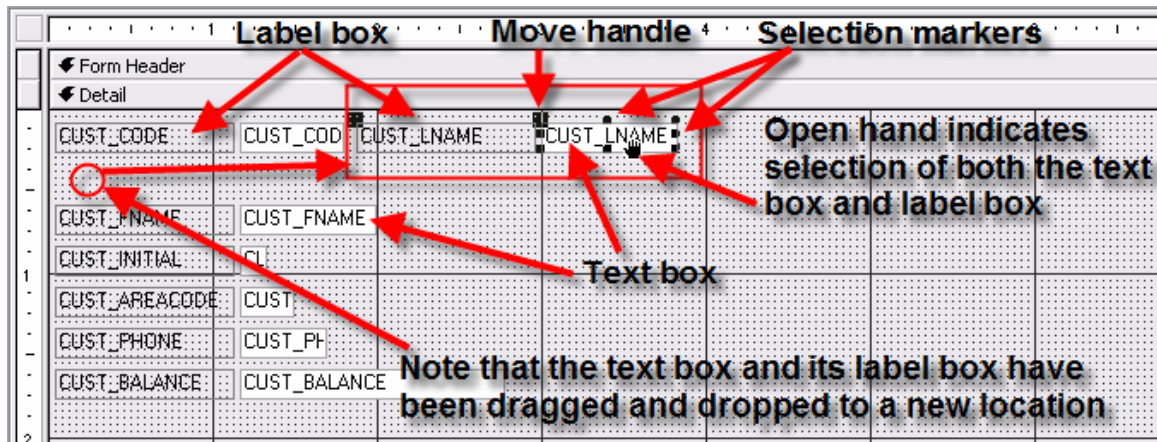
The first time you open the form in its design format, you will see the attribute list shown in Figure 120. All the CUSTOMER table's attributes have already been included in the form, so go ahead and close this field list. (If you later want to open this field list box again, click on the **Field list** button shown in Figure 120.)

Figure 120 Control the Form Width



If you want to widen the form, put the cursor on the form's edge to change the cursor to the format shown in Figure 120, and then drag the form limit to wherever you want it to be. You can control the vertical size the same way. Just put the cursor on the *top edge* of the **Form Footer** and drag up or down to suit your needs. If you put the cursor on the *bottom edge* of the **Form Footer**, you will be able to create a footer and control its width. (Incidentally, the horizontal and vertical rulers can be used later to help you line up selected output components or to mark multiple selection on the form.)

Figure 121 shows that the form limits have been dragged down and to the right to produce a larger form surface on which to place the form contents. Also note the various design components that have been marked on the figure.

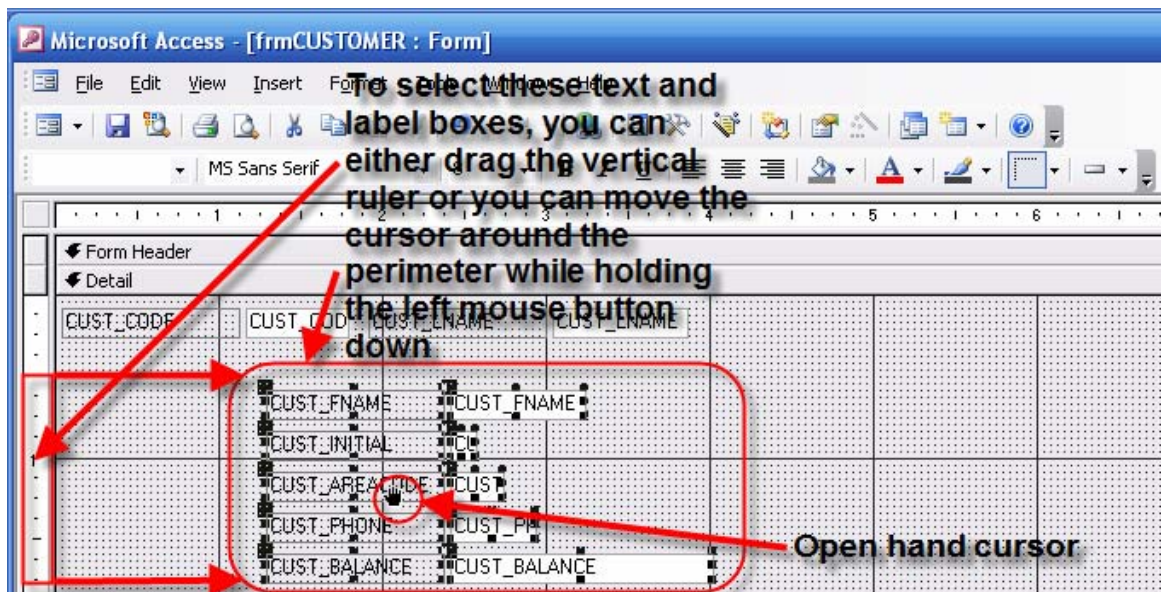
Figure 121 Design Components

As you examine Figure 121, keep the following points in mind:

- Clicking on the text box portion selects both the text box and its label box as a unit.
- Clicking on the label box selects only the label box.
- To create the “open hand” cursor, put the cursor on the text box and then move the cursor to the lower or upper limit of the text box to change its shape. (The cursor is rather sensitive, so move it slowly until it changes ...)
- The open hand indicates that both the text box and its associated label box have been selected as a unit. If you hold the left mouse button down, you can drag the text box *and its associated label box* to any location.
- If you put the cursor on a move handle, it will change to a pointing finger. Note that there are two move handles, one for the text box and one for the label box. (This cursor is not shown in Figure 121, because you can only show one cursor position at a time. However, go ahead and place your cursor over a move handle and watch the cursor change to pointing finger.)
- If you drag with the move handle, only the component marked by that move handle will move. Therefore, you can move the text and label boxes to different locations. For example, you can place the label box above the text box.
- If you place the cursor over a selection marker, the cursor shape will change to a two-headed arrow. You can drag on the selection markers to change the text and label box sizes.

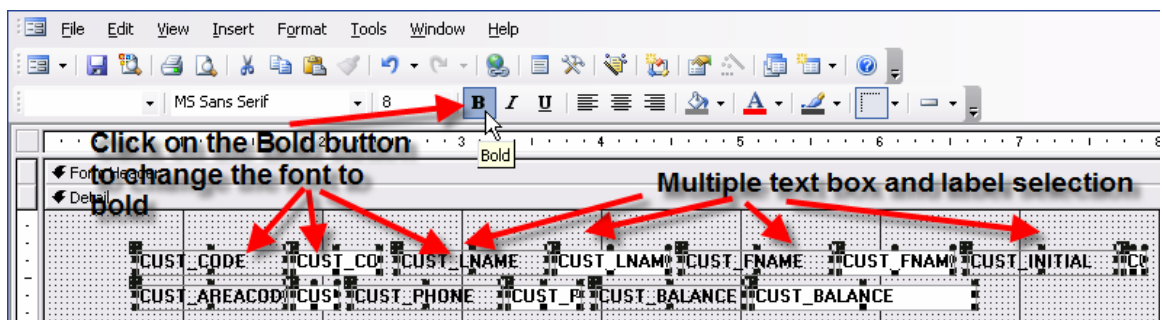
If you want to select all the text and label boxes at once, you can drag the cursor along the rulers. You can also select multiple text/label boxes by moving the cursor while holding the left mouse button down. Once the block of multiple text/label boxes has been selected, you can place the cursor on any marked text box to change the cursor shape to an open hand – you can then drag the block to any position on your form. (See Figure 122 to see a multiple selection.)

Figure 122 Multiple Component Selection



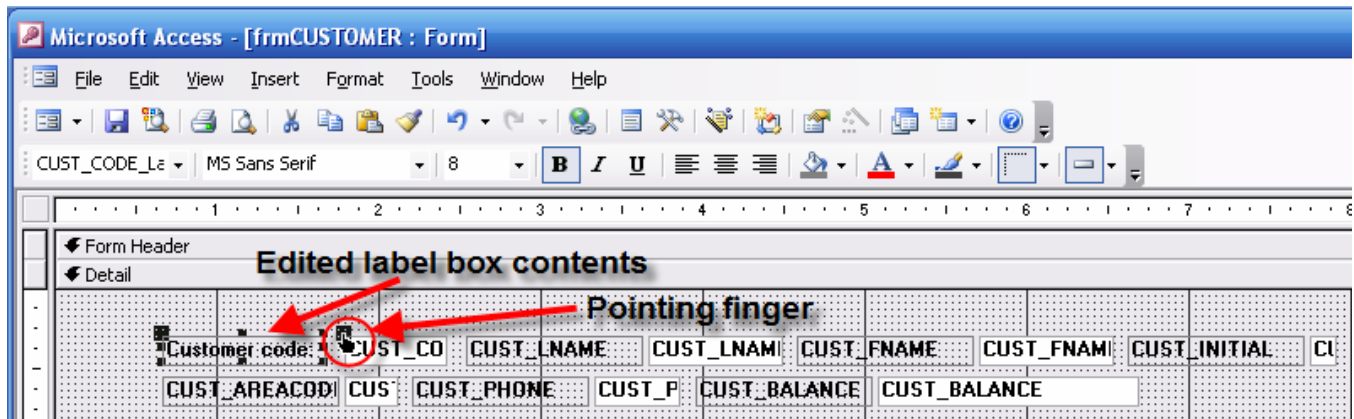
Practice moving the form components around until they approximately match Figure 123. Then change the fonts to Bold as shown in Figure 123.

Figure 123 Font Set to Bold



You can also change the label box contents by selecting the label box and then clicking on that selected label box to put it in edit mode. You can then edit label text. Note that Figure 124 shows that the CUST_CODE was changed to **Customer code**. (Remember that the label box width can be changed by dragging its limits.) Note also that the cursor was placed over the text box move handle to change the cursor shape to a pointing finger – you can now drag the text box to line it up with the edited label box.

Figure 124 Editing the Label Boxes



You have a large number of form design tools available. For example, you can add additional text, create boxes to delineate form components, and so on. If the form **Toolbox** is not open, right-click on any blank space on the form to generate the dialog box you see in Figure 125. Click on the **Toolbox** selection to open the toll box you see in Figure 126.

Figure 125 The Toolbox Option

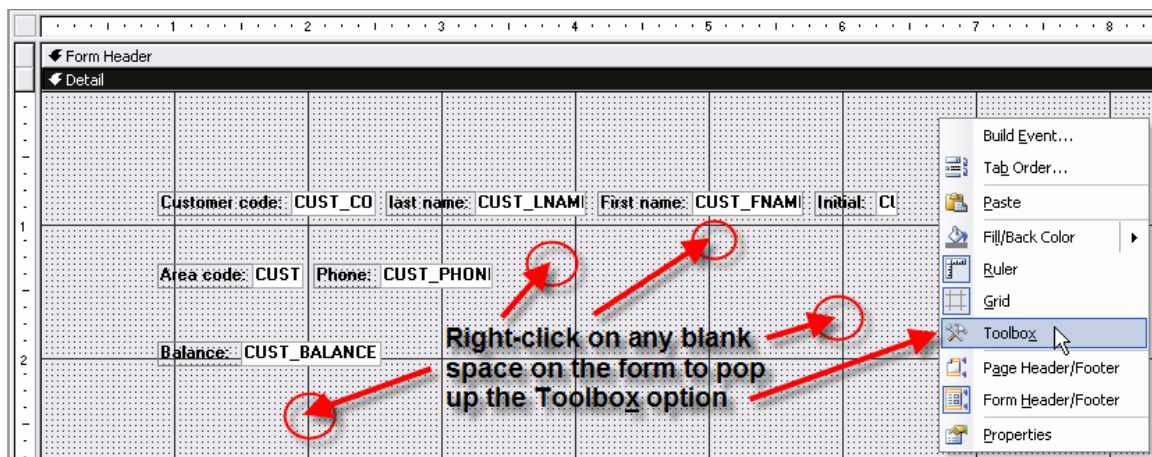
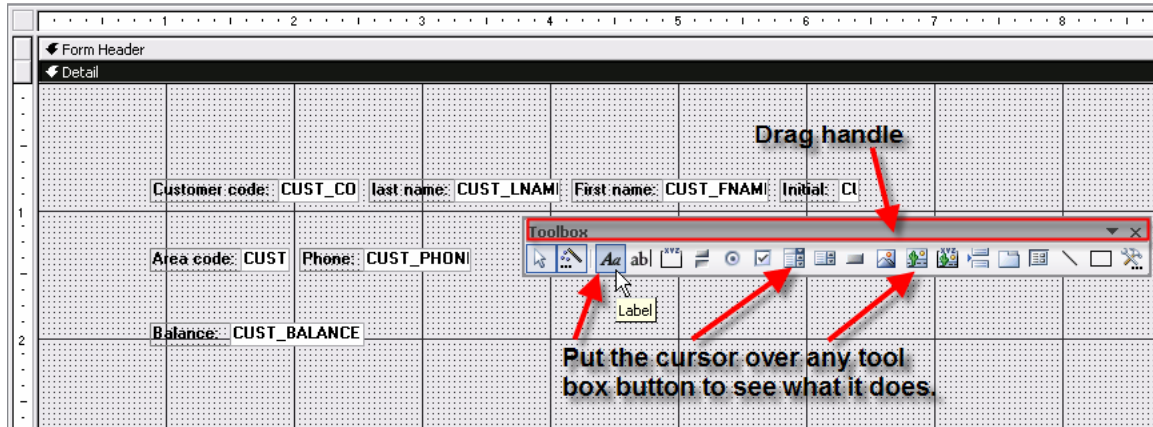


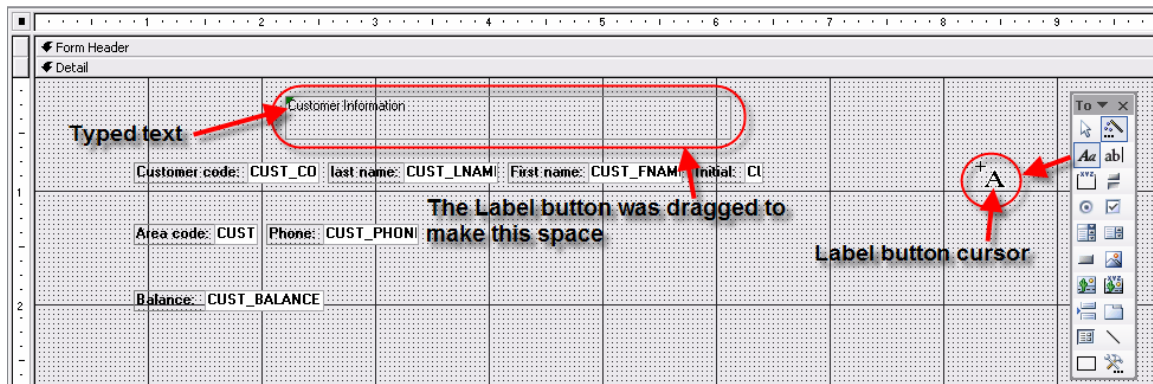
Figure 126 Toolbox Buttons



As you examine Figure 126, note that the tool box has changed shape and location. Like almost any Windows object, you can drag its limits to change its shape and you can drag it by placing the cursor over the “drag handle” and holding the mouse button down while moving the cursor. If you move the tool box into the screen margin, it will become just another button bar. (This was just a reminder ... you should know Windows well enough to be familiar with its operation.) Figure 127 shows that the tool box was moved and its shape was changed to have a double row of buttons arranged vertically.

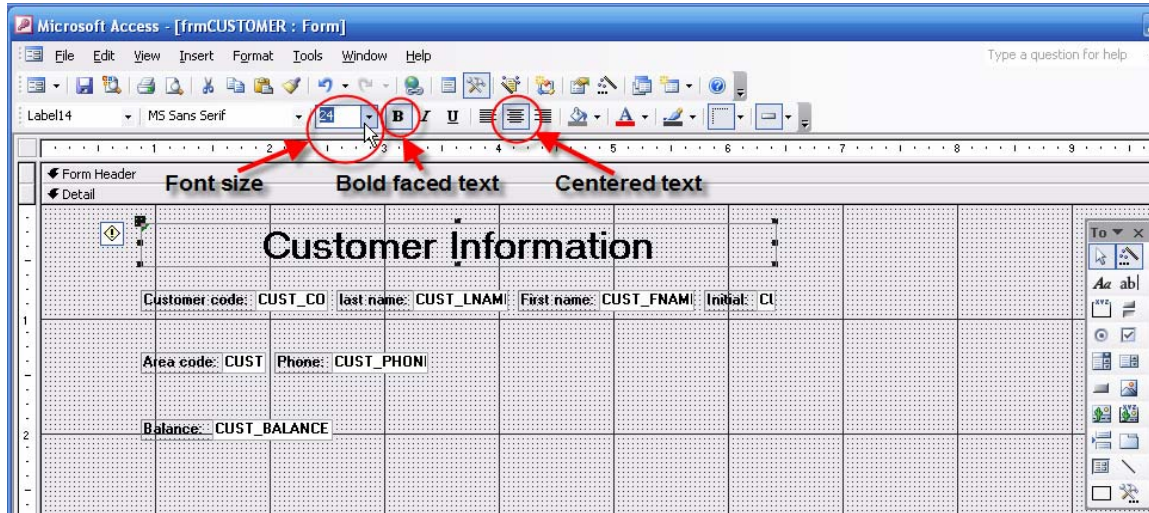
Figure 127 also shows that the **Label** button was selected. This selection changes the cursor to the **+A** shape you see here. If you drag this cursor anywhere on the form, you will create a text space in which you can type whatever is needed. In this example, the typed text is **Customer Information**. Click anywhere outside the text space to return the cursor to its normal function.

Figure 127 Using the Label Tool



You can edit the text as needed. Note that Figure 128 shows that the font size was changed, the text was centered, and the label box was resized.

Figure 128 Edited Text

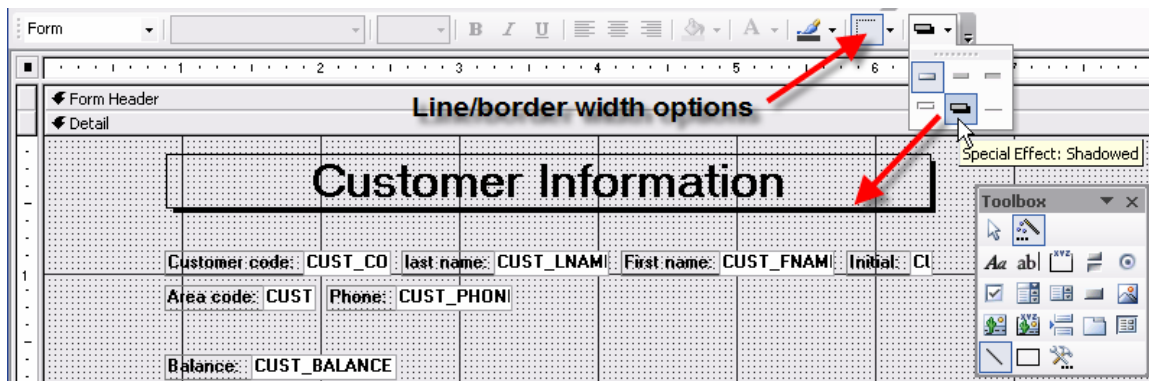


The form's looks can be improved with raised text box limits and color ... and you can use the rectangle tool to create logical groupings of information presented on the form. Figure 129 shows several enhancements that you can accomplish by doing the following:

- Select the **Customer Information** text box you just edited as shown in Figure 128. (The selection is confirmed by the selection markers – the small black squares around the text box perimeter.)
- Click on the **Special Effect: Shadowed** button to generate the special effect options.
- Select the **Shadowed** option by clicking on it. Note the change in the text box edge.

Also note – when you compare Figures 128 and 129 -- that the **Toolbox** has been moved and that its shape has been changed by dragging its limits. (And the attribute text boxes and their label boxes have been moved, too.) Go ahead and change the form you're working on to approximately match this format to keep "in synch" with the remaining discussion. After you have explored the use of many of the form design tools, you can, of course, change the form to suit your interests.

Figure 129 Additional Form Enhancements

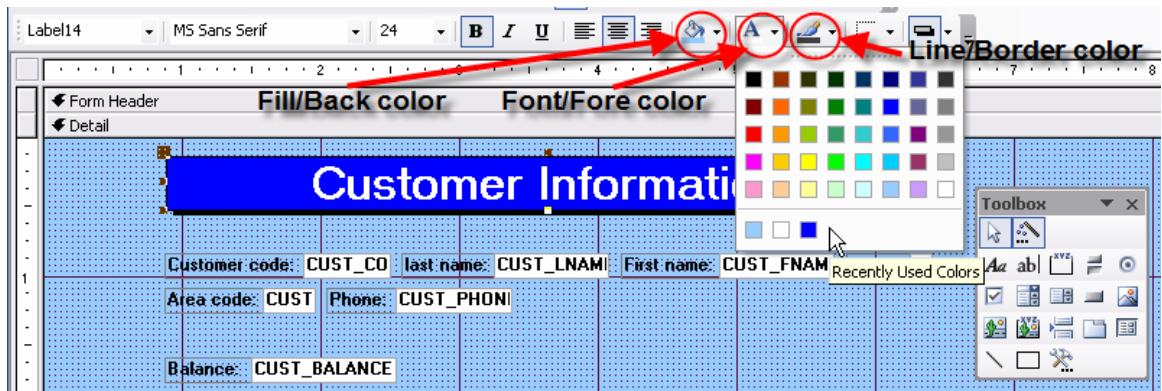


Let's take a look at just a few more form design options. For example, if you look at Figure 130, you will see that there are quite a few color options available. The match the figure you see here, you can take the following actions:

- To make the form light blue, click on any empty portion of the form to select the *entire* form. Next, click on the **Fill/Back color** button and click on the light blue color square.
- To make the **Customer Information** text box dark blue, click on the text box to select it – note that Figure 130 shows the selection markers around the text box perimeter. Next, click on the **Fill/Back color** button and click on the dark blue color square.
- To change the font color in the text box to white, make sure that the text box is still selected. Next, click on the **Font/Fore color** button and click on the white color square.

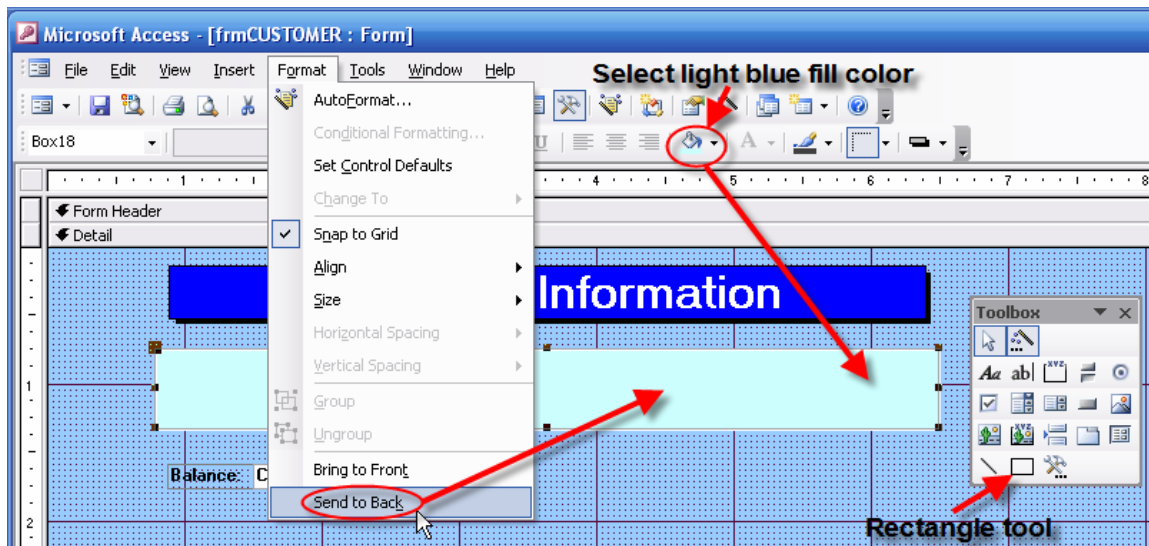
Note that Access keeps track of all the recently selected colors to make it easy to later match color selections of other form components.

Figure 130 Color Selections



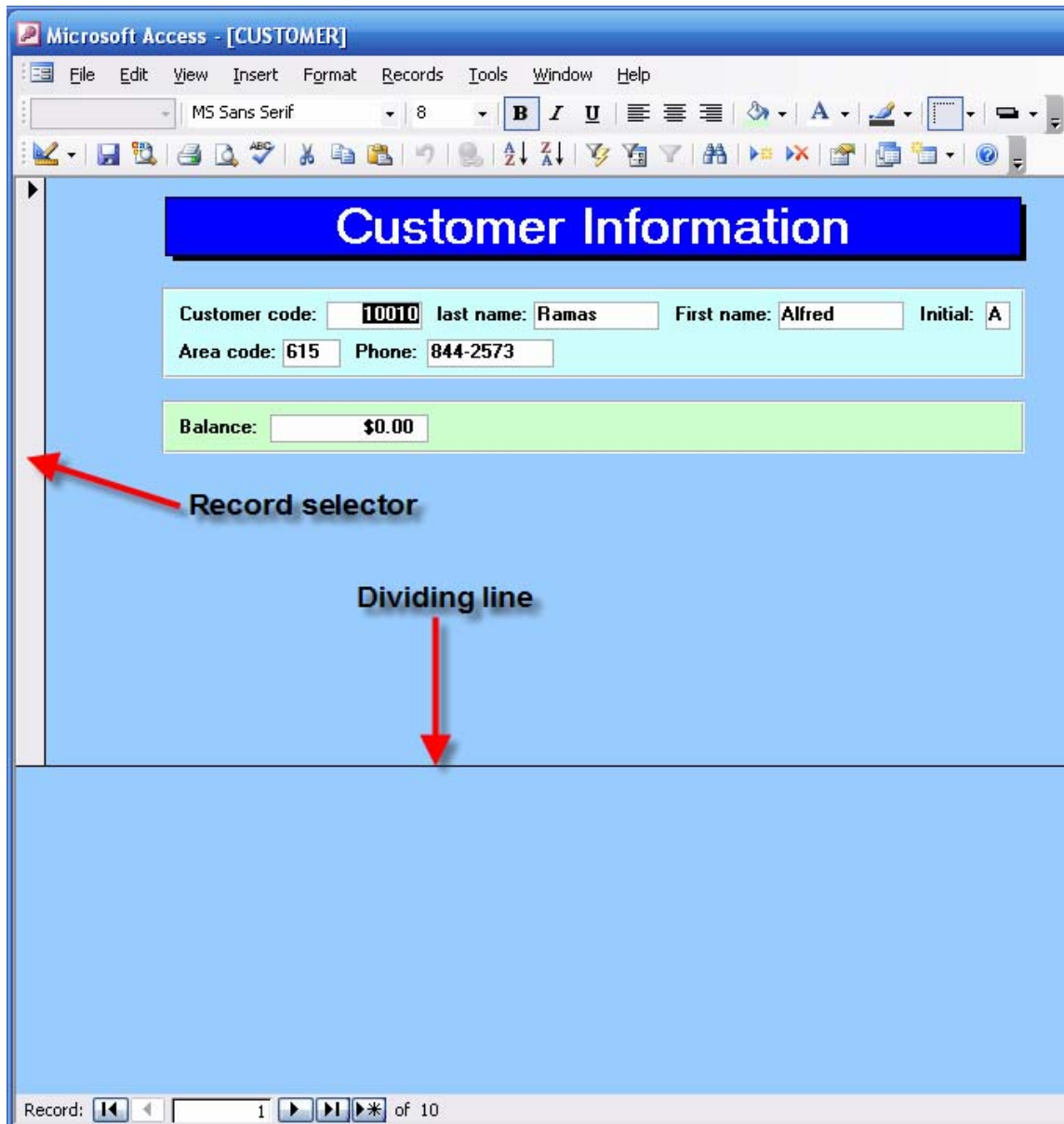
It is often useful to group logically similar attributes together as a visual unit. You can use the **Rectangle tool** to draw a rectangle around such a group. To get the job done, go to the tool box and click on the rectangle tool, then drag a rectangle around the attributes you want to group. The rectangle is initially clear and just shows its outline. However, you can use the **Fill/Back color** option to give the rectangle a color. Figure 131 shows that the fill selected color was light blue. Unfortunately, the default setting on the rectangle tool places the rectangle in front of the attributes, thus shading them out. However, note that you can use the **Format/Send to Back** option to send the now opaque light blue rectangle to the back, thus making the attributes visible again.

Figure 131 Using the Rectangle Tool



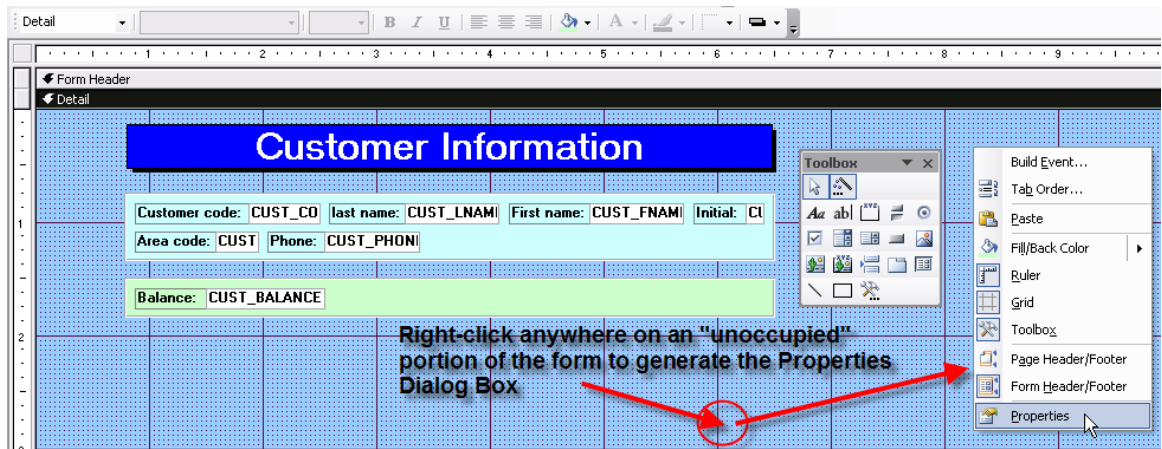
Repeat the process for the balance attribute, using a light green color. Finally, drag the **Customer Information** text box limits to line up with the two rectangles to balance the form. When you are done, open the form in its **Form View** to see Figure 132.

Figure 132 The Form in Form View



The form in Figure 132 shows two features you may want to remove: the record selector and the dividing line. You can control such form presentation features through the properties box. Open the form in its design view and then right-click to produce the **Properties Box** as shown in Figure 133. (In this case, the right-clicking was done on the form's **Detail** section.)

Figure 133 Generating the Properties Dialog Box



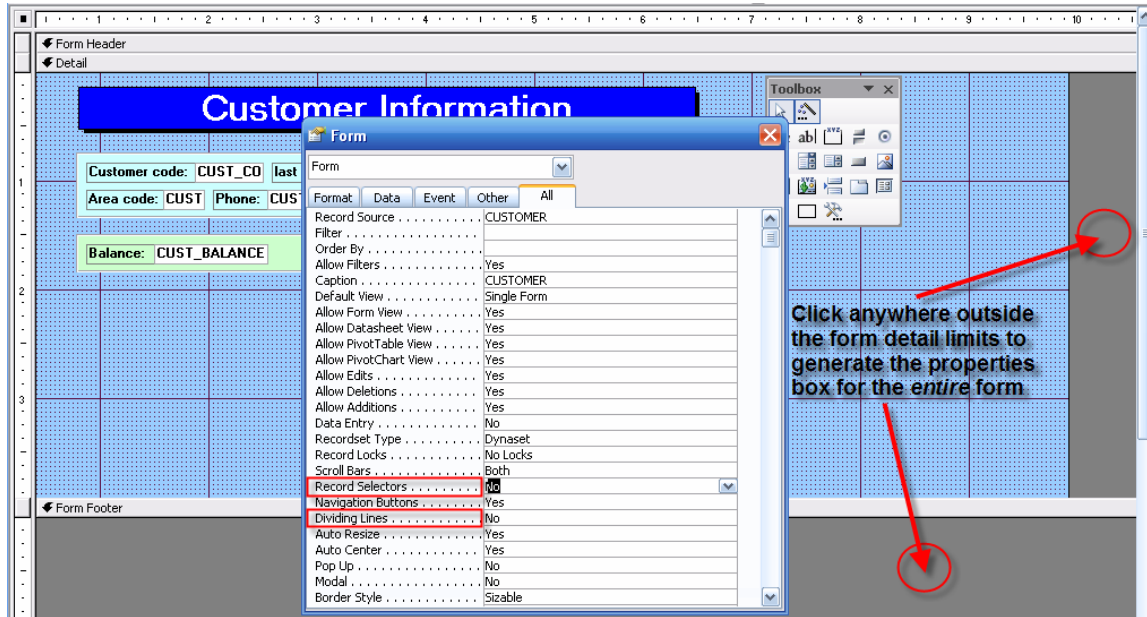
Click on the **Properties** option shown in Figure 133 to produce the properties box for the **frmCUSTOMER** form's detail section. If you want to produce the properties box for the entire form, rather than for some selected portion of that form, click outside the form's limits, as was done in Figure 134.

Note

You can get the properties box for any Access object by selecting it and then right-clicking on your selection. For example, if you want to see the properties of just the CUST_CODE *text* box, click on the CUST_CODE text box to select it and then right-click. If you want to see the properties of the Customer code: *label* box, click on the CUST_CODE textbox to select it and then right-click. And so on.

Regardless of where you right-clicked to produce the initial properties dialog box, you can generate the properties for any Access object by simply clicking on it. Go ahead and perform this action a few times until it becomes routine.

Figure 134 The Properties for the Entire Form



As you examine Figure 134, note that the **Records Selectors** property has been set to **No**. The same action was taken for the **Dividing Lines** property. The available options for each property can be generated by simply clicking on it to produce an arrow head such as the one shown for the **Record Selectors** property – you can then click on the arrow head to see the list from which you can make the selection. When you open the form in **Form View** again, the record selector and dividing line are gone.

The form is essentially completed. However, you may want to put the **Customer Information** text box in a header. You can create the header space by simply dragging the form’s detail limit down. (You can create the form footer by simply dragging the footer limit down.) Finally, if the white background for the textbox contents are annoying to you and you want them to have the same color as the rectangles, you can select them and then use a fill color to match the background. As you can tell by looking at Figure 135, that was done to create this illustration. Figure 136 shows the **frmCUSTOMER** form in **Form View**. (Naturally, given your familiarity with the Windows environment, you can resize the window and the form within that window.

Figure 135 Final Touches

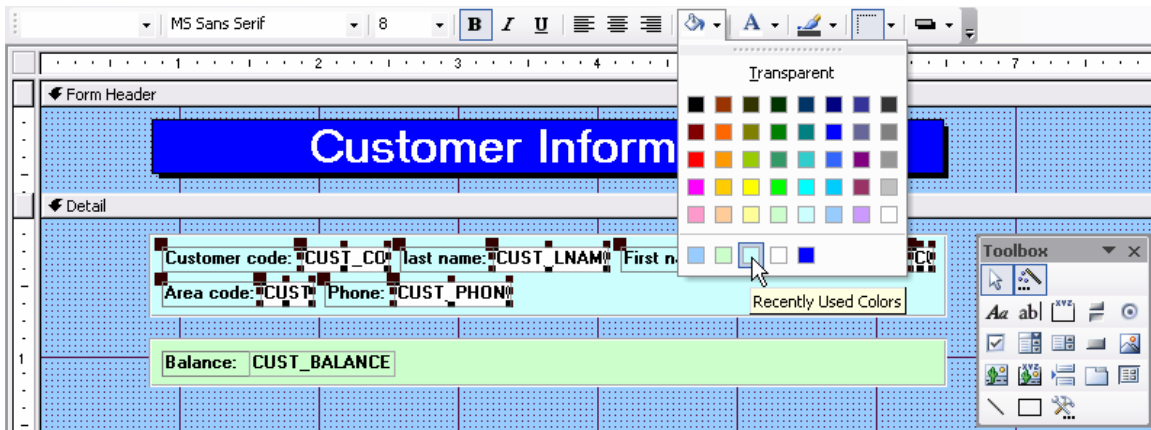
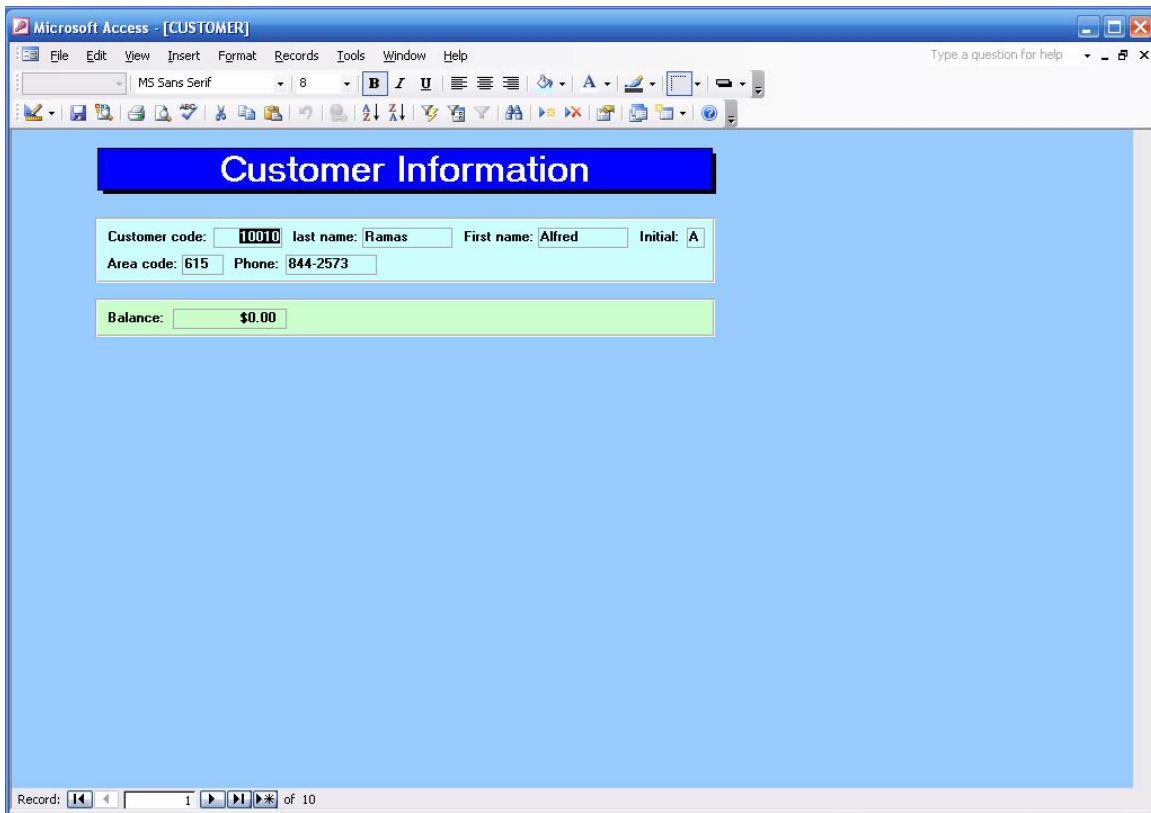


Figure 136 The Completed CUSTOMER Form



3.1.2 Forms Based on Queries

Forms may be used to present data and/or information from multiple sources. Queries are particularly good as a basis for form design, because they can access multiple tables directly.

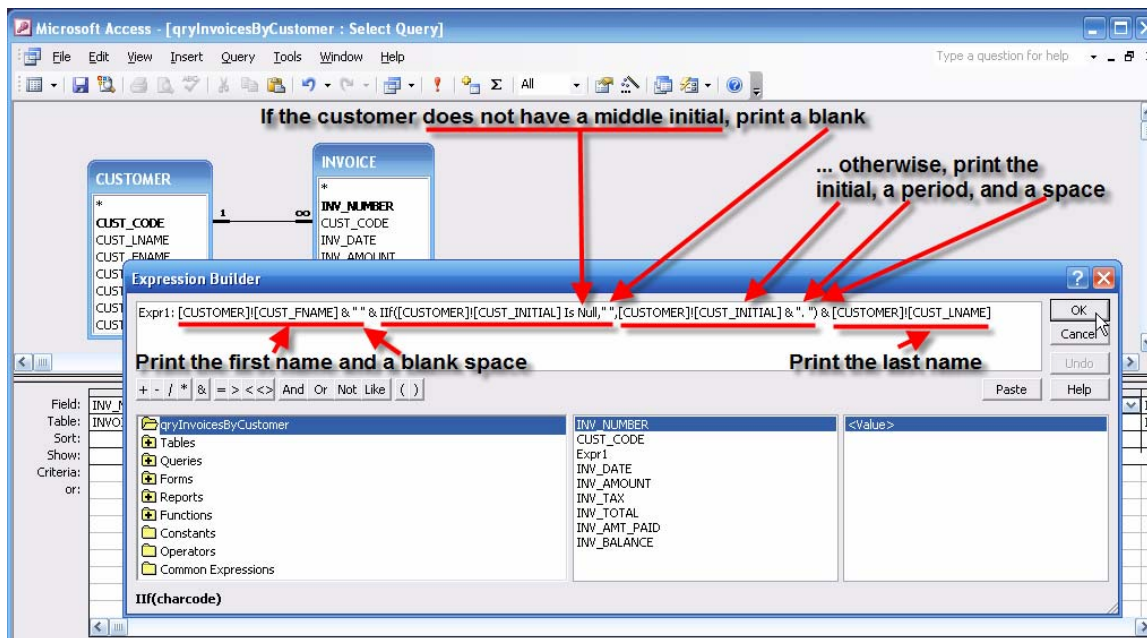
Let's create an invoice form based on the query you see in Figure 137. Note that this query uses in expression that concatenates the customer's first name, initial, and last name. However, because some customers may not have an initial, a logical function must be used. Note the expression in the query builder in Figure 137. The logical "if and only if" (**Iif**) function uses the following format:

Iif(expression, do if true, do if false)

Note that the two possible actions are separated by commas.

In the query shown in Figure 137, if there is no middle initial – that is, the **CUST_INITIAL Is Null** – a blank space must be printed after the customer's first name. If the customer has a middle initial -- that is, the customer initial is *not* null -- the initial must be printed, followed by a period and space. In either case, the customer's last name must be printed, so the CUST_LNAME is not included in the logical **Iif** statement. (To build this query, drag and drop the INVOICE table's INV_NUMBER and then drag and drop the CUSTOMER table's CUST_LNAME – or any of the customer's name components -- to reserve a space for the expression you are about to build. Then right-click on the CUST_LNAME field and select the expression builder to select the CUSTOMER table's CUST_FNAME, CUST_INITIAL, and CUST_LNAME – after you have deleted the CUST_LNAME that you originally dragged to the field -- to build the expression. (You can type the **&**, the blank space enclosed by quotes, and the **Iif** function components.)

Figure 137 The Design View of the Invoice Query



When you are done with the expression builder, click OK to save the expression and then drag and drop all the remaining INVOICE table fields to the QBE screen's field spaces. Save the query as **qryInvoicesByCustomer** and then open it in data sheet form as shown in Figure 138.

Figure 138 The Data Sheet View of the Invoice Query

INV_NUMBER	CUST_CODE	Customer Name	INV_DATE	INV_AMOUNT	INV_TAX	INV_TOTAL	INV_AMT_PAID	INV_BALANCE
1002	10011	Leona K. Dunne	16-Mar-06	\$9.98	\$0.80	\$10.78	\$10.87	\$0.00
1004	10011	Leona K. Dunne	17-Mar-06	\$34.87	\$2.79	\$37.66	\$37.66	\$0.00
1008	10011	Leona K. Dunne	17-Mar-06	\$399.15	\$31.93	\$431.08	\$431.08	\$0.00
1003	10012	Kathy W. Smith	16-Mar-06	\$153.85	\$12.31	\$166.16	\$100.00	\$66.16
1001	10014	Myron Orlando	16-Mar-06	\$24.94	\$2.00	\$26.94	\$20.00	\$6.94
1006	10014	Myron Orlando	17-Mar-06	\$397.83	\$31.83	\$429.66	\$300.00	\$129.66
1007	10015	Amy B. O'Brien	17-Mar-06	\$34.97	\$2.80	\$37.77	\$37.77	\$0.00
1005	10018	Anne G. Farriss	17-Mar-06	\$70.44	\$5.64	\$76.08	\$76.08	\$0.00
1009	10019	Olette K. Smith	27-Mar-06	\$180.74	\$14.46	\$195.20	\$0.00	\$0.00
*								

Next, begin the form design just as you did in the previous section. However, this time you use the **qryInvoicesByCustomer** query as the data source. Use the formatting techniques you learned in the previous section to produce the form you see in Figure 139.

Figure 139 The Design View of the Invoice Form

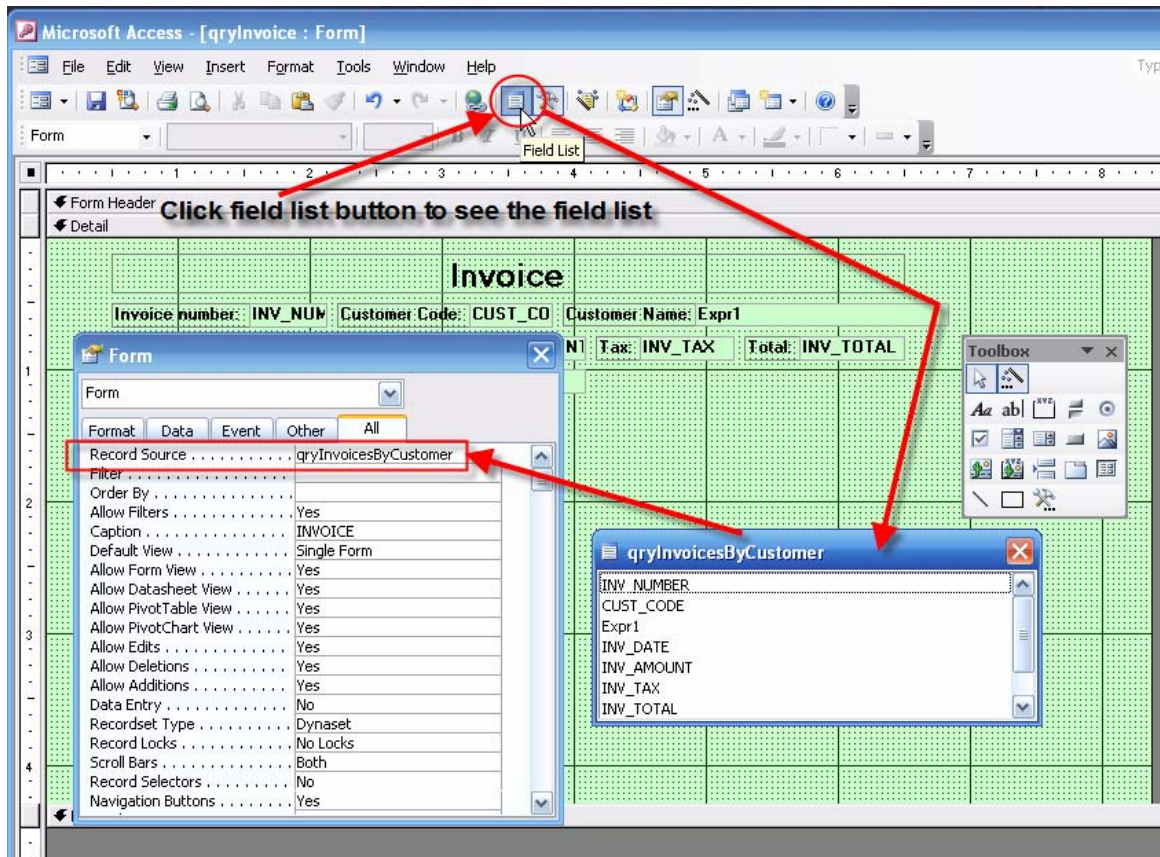
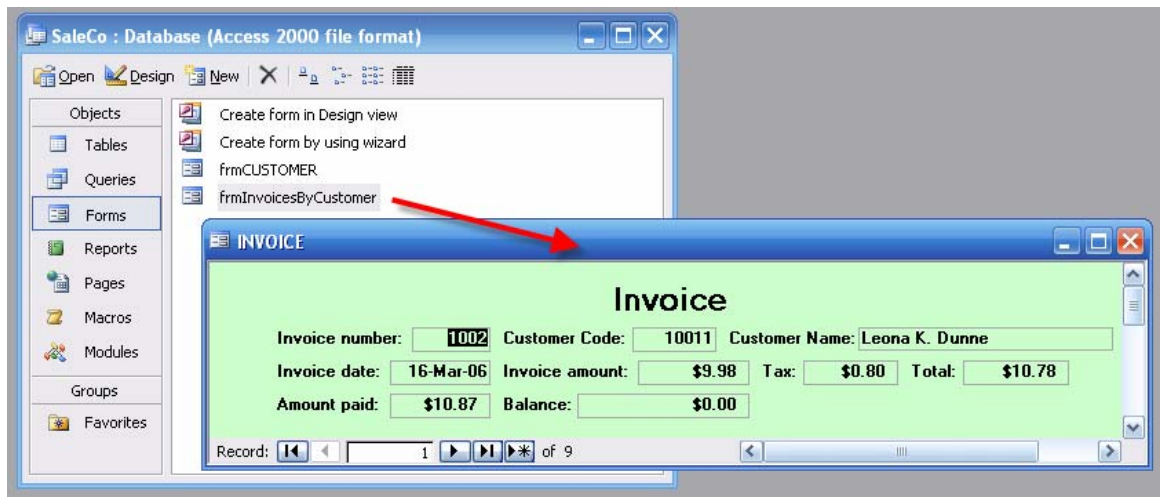


Figure 140 shows the final form view ... the form limits were dragged in to reduce the presentation size and to let you see the form relative to its **Forms** list. (Note that the window has been resized to show both screen components and that the form limits have been dragged to limit its display size.)

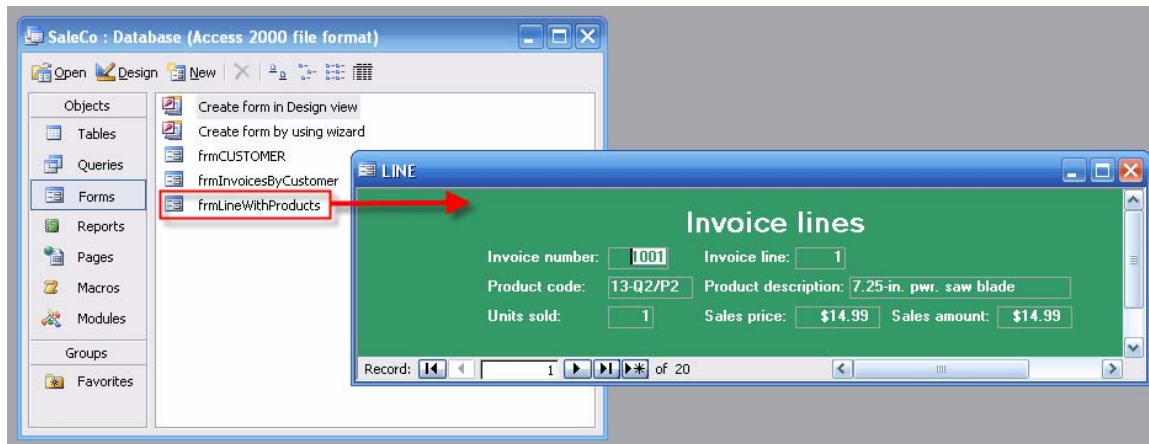
Figure 140 The Form View of the Invoice Form



In the next section you will learn how to use one form as a subform to another, so let’s go ahead and build an invoice line form next. You will also learn some additional techniques that will turn out to be very useful when you try to control input via a form.

Start by creating the form for the invoice lines by repeating the now familiar routine you used early in Section 3.1, Figure 116. (You can use the LINE table as the data source.) Format the form to select color, font size, and other characteristics to match the form you see in Figure 141.

Figure 141 The Line Form

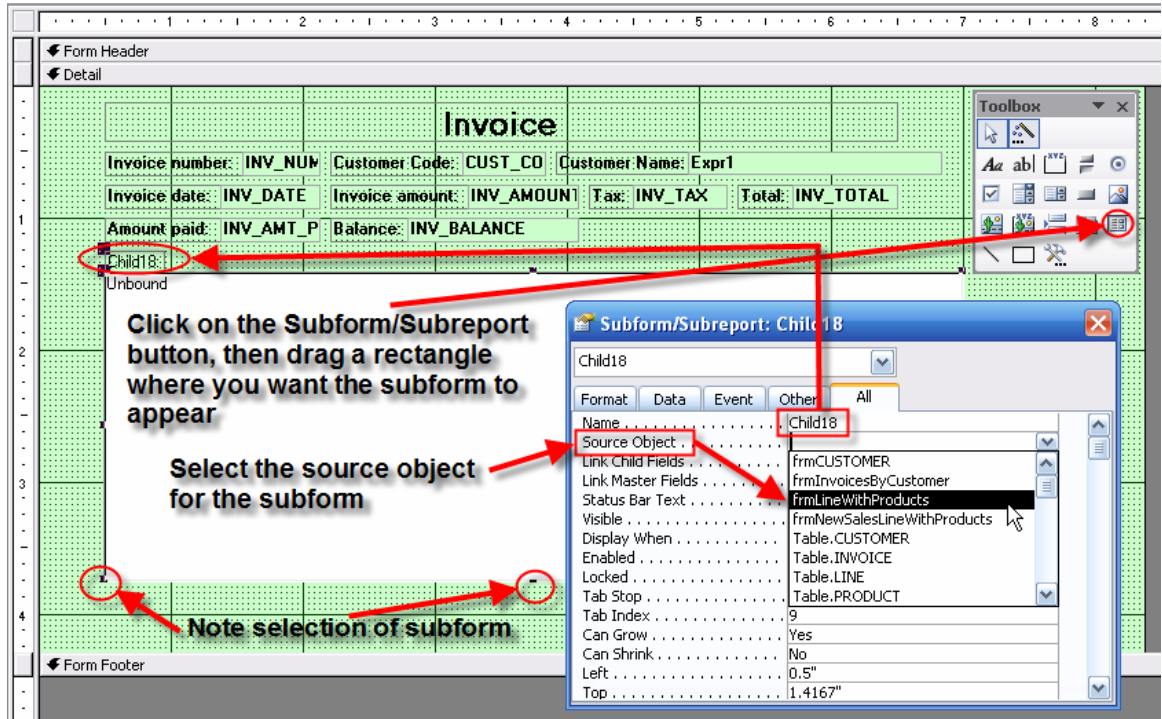


3.1.3 Forms with Subforms

Forms contribute much to the ability to display data and/or information in a meaningful and visually appealing way. However, from an information management point of view it would be very desirable to see the interaction of related data. Fortunately, the information shown in Figures 140 and 141 can be combined by showing the invoice and (related) invoice lines on a single screen through a from/subform combination. In this example, the subform contains the LINE data, while the main form contains the “parent” INVOICE data. (In a 1:M relationship, the “M” side would be found in the subform and the “1” side would be seen in the main form.)

To create an (invoice) form/(line) subform presentation, begin by opening the **frmInvoicesByCustomer** form shown in Figure 140 in its design view, as shown in Figure 142. (To maximize the work space, go ahead and maximize the screen.) Next, click on the **Form/Subform** button you see in the **Toolbox**. (When you do that, the cursor will change to show a small rectangle with a “+” symbol just outside its upper left-hand corner – not shown here.) Using the new cursor symbol, draw the rectangle shown in the bottom portion of the **frmInvoicesByCustomer** (main) form in Figure 142 by dragging the cursor. As soon as you release the mouse button, you will see **Subform Wizard** dialog box appear. **Close this wizard without using it ...** you will lose the ability to control the process if you use that wizard. (That’s why you don’t see the **Subform Wizard** dialog box in Figure 142.)

Figure 142 Creating a Subform



As soon as you have “drawn” the subform outline shown in Figure 142, you will see its automatically generated label – in this example, it is **Child18**. (You will probably see a different “child” number designation, because the number is based on how many times you gave created a “child” object before.)

While the subform is still selected – note the selection squares in Figure 142 – open the **Properties** box for the subform. (Note that the properties box is automatically “labeled” as **Subform/Subreport: Child 18**.) Now click on the **Source Object** and select the **frmLineWithProducts** form from the list as shown in Figure 142.

Next, click on the **Link Child Fields** as shown in Figure 143. (This action will produce the **Subform Field Linker** you see in Figure 143.) Note that the (correct) link is made through the INV_NUMBER, because the INV_NUMBER is the foreign key (FK) in the LINE table that links the LINE table to the INVOICE table. Because this selection is correct, just click OK to accept it. You are done ... that’s all there is to it. Naturally, you may want to drag the form and/or subform limits to line up components or to perform other formatting chores you learned earlier in this section. When you are done, open the form/subform in form view to see Figure 144. (Note that the illustration shows the selection of invoice record 4. This invoice contains three invoice lines. To preserve the original **frmInvoicesByCustomer** form, save the new form as **frmInvoicesByCustomerMainFormWithSubform**.)

Figure 143 Setting the Link

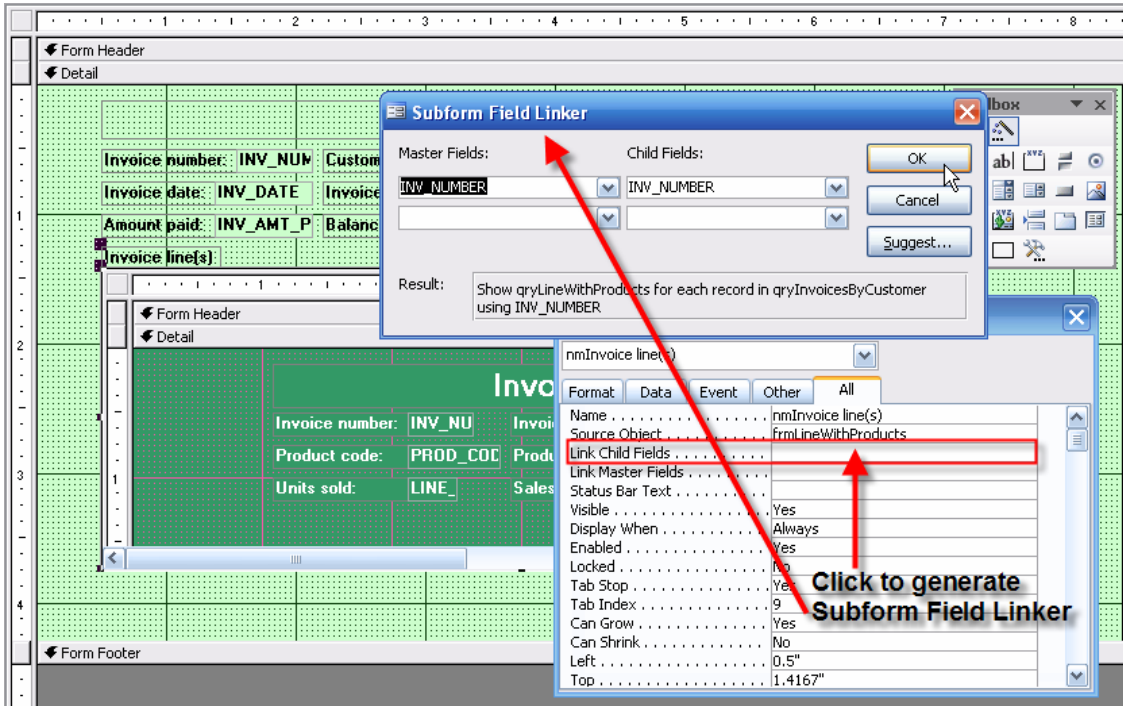
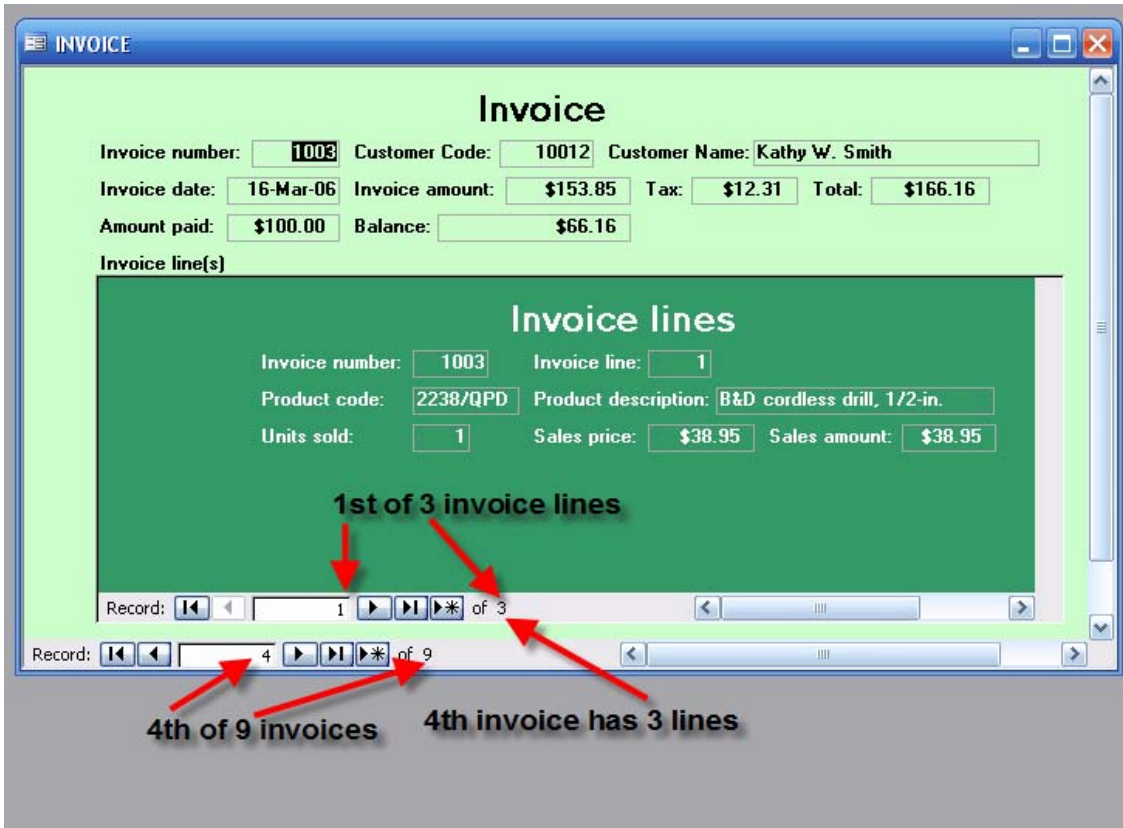


Figure 144 Completed Form-Subform



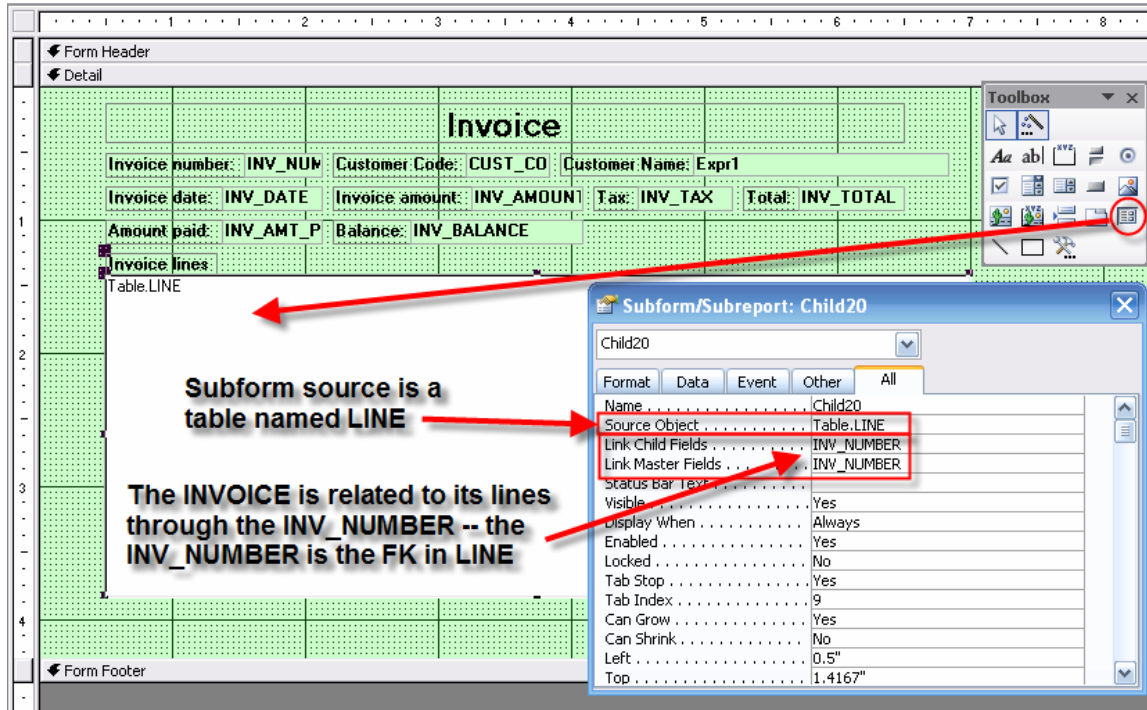
The form/subform design can be extended to include subforms that themselves include subforms. For example, you can use the just-created **frmInvoicesByCustomerMainFormWithSubform** as the subform to the **frmCUSTOMER** you saw in Figure 136. (Use the techniques you just learned – see Figures 142-145 -- to produce the form/subform you see in Figure 145. Save the new form/subform as **frmCUSTOMERwithInvoiceAndInvoiceLines** to preserve the original **frmCUSTOMER form**.

Figure 145 A More Complex Form-Subform

The screenshot shows a multi-level form structure. At the top is a blue header 'Customer Information'. Below it are fields for Customer code (10011), last name (Dunne), first name (Leona), initial (K), area code (713), and phone (894-1238). A balance field shows \$0.00. The main subform, titled 'frmInvoicesByCustomerMainFormWithSubform', contains an 'Invoice' subform. The 'Invoice' subform has fields for invoice number (1002), customer code (10011), customer name (Leona K. Dunne), invoice date (16-Mar-06), invoice amount (\$9.98), tax (\$0.80), total (\$10.78), amount paid (\$10.87), and balance (\$0.00). Below the invoice subform is an 'Invoice line(s)' subform, which is currently displaying one line. This subform has fields for invoice number (1002), invoice line (1), product code (54778-2T), product description (Rat-tail file, 1/8-in. fine), units sold (2), sales price (\$4.99), and sales amount (\$9.98). At the bottom of the form, there are record navigation controls. Three red arrows point to these controls with the following annotations: '2nd customer (10011) has 1 invoice' points to the 'of 1' indicator; 'The invoice has 3 invoice lines' points to the 'of 3' indicator; and '2nd of 10 customers' points to the '2' in the record number field.

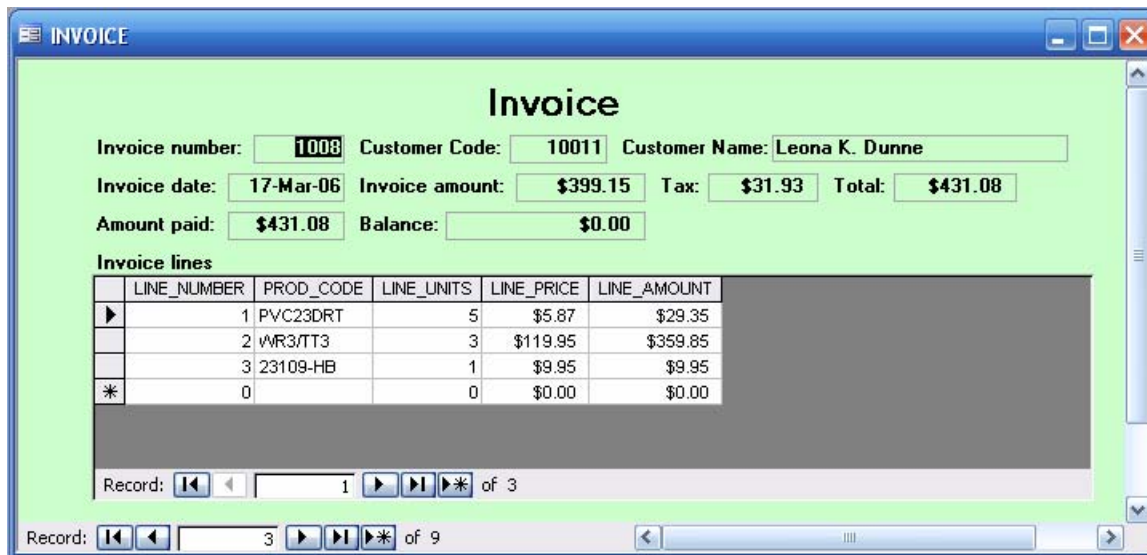
Thus far, you saw how forms could be used a subforms to other forms. However, you can also create subforms based on queries, without first using a form to format the query output. Start the form/subform design with the same technique that you saw in Figure 142. (Remember to turn off the **Form/Subform Wizard** to retain direct control of the design process.) Figure 146 shows that the **Source Object** is the **table** named **LINE**. To preserve the original **frmInvoicesByCustomer** form on which this new form/subform is based, save this second version of the invoice/line form/subform as **frmInvoicesByCustomerMainFormWithSubform-Version2**.

Figure 146 Design View of a Subform Based on a Table



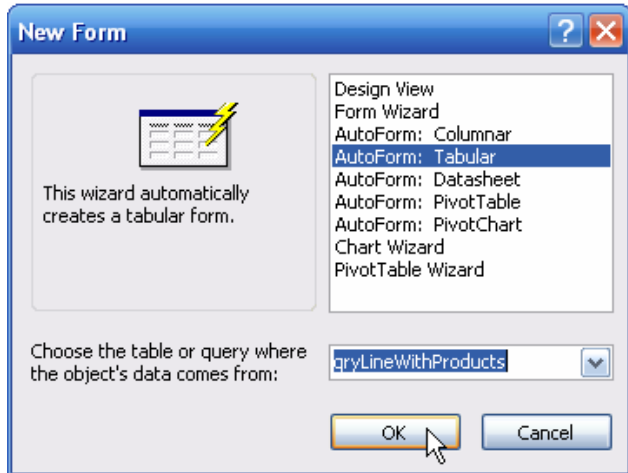
When you open this new form in **Form View**, you will see the output in Figure 147. Whether you use this form/subform design approach or the one in Figure 144 depends largely on end user desires and your own preferences. (Figure 147's output shows all the invoice lines at once, while Figure 144's invoice lines are shown one at a time as you click through the records.)

Figure 147 Form View of a Subform Based on a Table



Incidentally, you can create a form to produce an output that looks remarkably like a query output. For example, take a look at Figure 148, which shows the start of the creation of a form based on a tabular presentation format. (Note that this form will be based on the [qryLineWithProducts](#) query.)

Figure 148 Form Based on Tabular View



When you click **OK** on the **New Form** wizard dialog box shown in Figure 148, you'll see the output in Figure 149. (As you can tell, the window size has been decreased by dragging its limits, using the standard Windows procedure.)

Figure 149 Form with Tabular Output

IUMBER	NUMBER	PROD_	PROD_DESCRIPT	LINE_UNITS	LINE_PRICE	LINE_AMOUNT
1001	1	13-Q2/f	7.25-in. pwr. saw bla	1	\$14.99	\$14.99
1003	3	13-Q2/f	7.25-in. pwr. saw bla	5	\$14.99	\$74.95
1007	1	13-Q2/f	7.25-in. pwr. saw bla	2	\$14.99	\$29.98
1003	2	1546-Q	Hrd. cloth, 1/4-in., 2:	1	\$39.95	\$39.95
1006	2	2232/Q	B&D jigsaw, 12-in. bl	1	\$109.92	\$109.92
1003	1	2238/Q	B&D cordless drill, 1/	1	\$38.95	\$38.95
1001	2	23109+	Claw hammer	1	\$9.95	\$9.95
1004	2	23109+	Claw hammer	2	\$9.95	\$19.90

Record: 1 of 20

Naturally, you can modify the wizard-produced form, using the design techniques you learned earlier in this section. Figure 150 shows the edited form. Save the form as [frmLineWithProducts-TabularView](#).

Figure 150 Edited Form with Tabular Output

Invoice	Line	Prod. Code	Product Description	Units Sold	Line Price	Amount
1001	1	13-Q2/P2	7.25-in. pwr. saw blade	1	\$14.99	\$14.99
1003	3	13-Q2/P2	7.25-in. pwr. saw blade	5	\$14.99	\$74.95
1007	1	13-Q2/P2	7.25-in. pwr. saw blade	2	\$14.99	\$29.98
1003	2	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1	\$39.95	\$39.95
1006	2	2232/QTY	B&D jigsaw, 12-in. blade	1	\$109.92	\$109.92
1003	1	2238/QPD	B&D cordless drill, 1/2-in.	1	\$38.95	\$38.95
1001	2	23109-HB	Claw hammer	1	\$9.95	\$9.95

You can substitute the [frmLineWithProducts-TabularView](#) form in Figure 150 as the subform in the [frmInvoicesByCustomerMainFormWithSubform-Version2](#) form/subform you saw in Figure 146. (Note that Figure 151 shows that you have simply used the [frmLineWithProducts-TabularView](#) as the new **Source Object**.)

Figure 151 Edited Form with Subform

Save the edited form/subform as [frmInvoicesByCustomerMainFormWithSubform-Version3](#) to preserve the original version 2. Open version 3 to yield the output shown in Figure 152. (Compare this output with the one shown in Figure 144.)

Figure 152 Edited Form with Subform Output

The screenshot shows an Access form window titled "INVOICE". The form has a light green background and contains the following fields:

- Invoice number: 1003
- Customer Code: 10012
- Customer Name: Kathy W. Smith
- Invoice date: 16-Mar-06
- Invoice amount: \$153.85
- Tax: \$12.31
- Total: \$166.16
- Amount paid: \$100.00
- Balance: \$66.16

Below these fields is a subform titled "Invoice lines" which displays a table with the following data:

Invoice	Line	Prod. Code	Product Description	Units Sold	Line Price	Amount
1003	3	13-Q2/P2	7.25-in. pwr. saw blade	5	\$14.99	\$74.95
1003	2	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	1	\$39.95	\$39.95
1003	1	2238/QPD	B&D cordless drill, 1/2-in.	1	\$38.95	\$38.95

At the bottom of the form, there are record navigation controls. The main form shows "Record: 1 of 3" and the subform shows "Record: 4 of 9".

3.1.4 Controlling Input with Combo Boxes

In a real world environment, the end user is likely to use a scanner to enter product codes and prices. However, when you are prototyping database applications, you probably don't use such "final product" options. So how do you control end user input when it involves something as convoluted as a product code? The answer is simple – just design the form that uses such inputs to include *combo boxes*. A combo box is simply a textbox that lists the available input options from which the end user may choose.

Microsoft Access makes it easy to produce a combo box. To illustrate its design, open the [frmLineWithProducts](#) in its design view and save it as [frmLineWithProducts-ComboBox](#) to preserve the original form. Figure 153 displays the [frmLineWithProducts-ComboBox](#) in its design view. Note the following procedures and features:

1. Open the [frmLineWithProducts](#) in **Design View** and delete the original **Product code:** field. (Check Figure 141 to see this field. The location of the deleted field is marked in Figure 153.)
2. Click on the **Combo Box** tool in the **Toolbox**.
3. Drag a space on the screen – in this example, the combo box space was created in the upper right-hand portion of the screen. In this illustration, the default name is **Combo23**. (Because this default name is selected by Access based on the previous use of combo box designs, your combo box default name is likely to be different.)
4. Right-click on the combo box to generate its properties listing. Note that the **Combo Box** properties are shown in the figure.
5. Select the appropriate properties – take a look at the figure. You should be familiar with the properties box drop-down lists, so use them. For example, to select the **Control Source**, click on the item and then click on the familiar down arrow – not shown here – to see the sources. In this example, the selected control source is the PROD_CODE in the LINE table, shown as **LINE.PROD_CODE**.
6. The **Row Source Type** is **Table/Query**.

7. Select the **Row Source** as shown. In other words, the row source for this combo box will be the query named **qryNewSalesLineWithProducts**.
8. To let the end user see what is expected to be entered, three fields will be shown. Therefore, the **Column Count** is set at **3**.
9. Because the control source is the PROD_CODE in the LINE table, only the PROD_CODE field can be bound to the LINE table. This field value is the first one in the query, so the **Bound Column** is column 1.
10. To let the end user know the meaning of the combo box items, select **Yes** to mark the **Column Heads** option. (You can edit the column head text by using the properties for each of the three fields at the query level. If you want to edit the column heads, you can open the **qryNewSalesLineWithProducts** query in its **Design View** and make the necessary changes.
11. There are three columns to be displayed. To control the **Column Widths**, enter the column width values in inches. Note the selection of **0.7";2";0.8"** in this option line. (Actually, if you type **.7,2,.8**, Access will format the entry for you.) This first entry is an approximation, so may have to toggle back and forth between the form's **Design View** and **Form View** to get the precise column width setting.
12. The **List Rows** option shows the default value **10**. You can change this value by simply typing the number of rows you would like to see displayed. Access automatically creates a scroll bar for you if the actual number of rows exceeds the specified number of rows.
13. For documentation purposes, change the combo box **Name** to **nmProductCodeComboBox**.
14. Edit the combo box label on the form to change the **Combo23** shown here to **Product Code**:
15. Drag the completed combo box to the original product code location – marked in Figure 153 – and drop it. (Figure 154 shows the combo box in its new location.)
16. Save the newly edited **frmLineWithProducts-ComboBox** form.

Figure 153 Creating a Combo Box

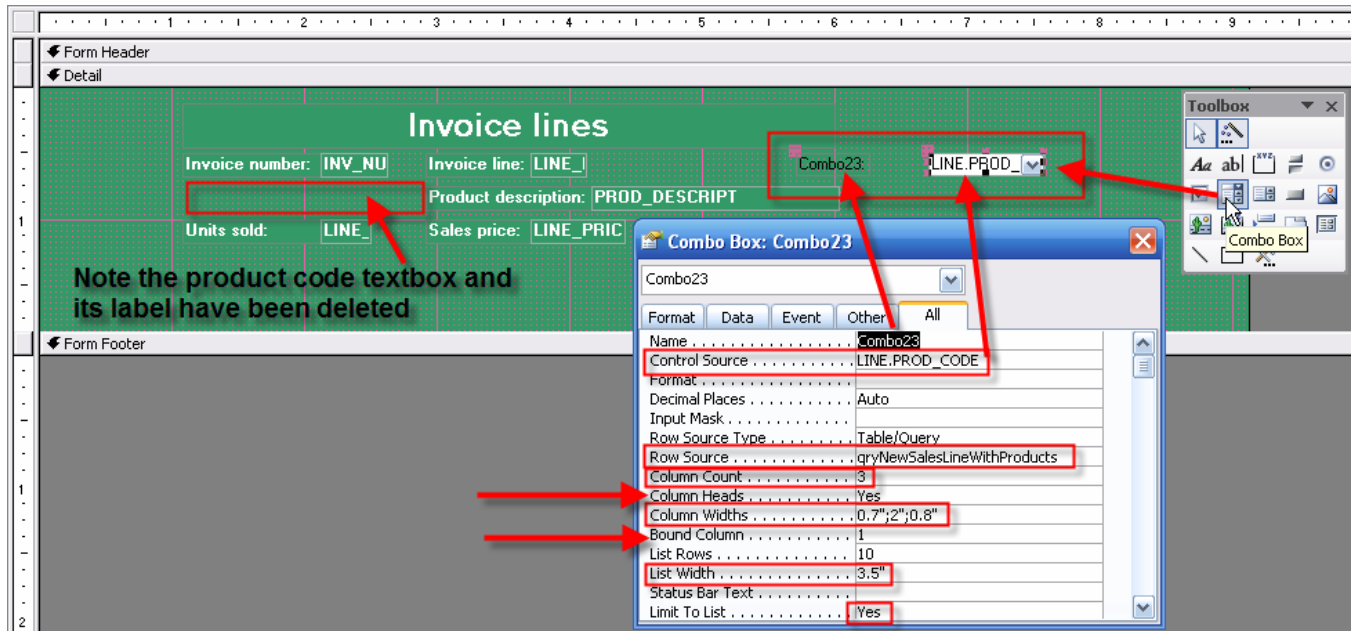


Figure 154 Edited and Moved Combo Box

The screenshot shows the design view of a form titled 'Invoice lines'. The form is set to 'Detail' view. The fields are arranged as follows:

- Invoice number: INV_NU
- Invoice line: LINE_
- Product code: LINE.PROC (with a dropdown arrow)
- Product description: PROD_DESCRIPT
- Units sold: LINE_
- Sales price: LINE_PRIC
- Sales amount: LINE_AMT

Open the [frmLineWithProducts-ComboBox](#) form in its form view to inspect the results shown in Figure 155. If you now click on the down arrow at the combo box margin, you will see the available entries in Figure 156. (You can make a selection by simply clicking on a listed item. However, *do not do so at this point, because you will change the invoice line product code, thus destroying the historical accuracy of the transaction you see here.* You can experiment with the combo box when you make a new sales transaction.)

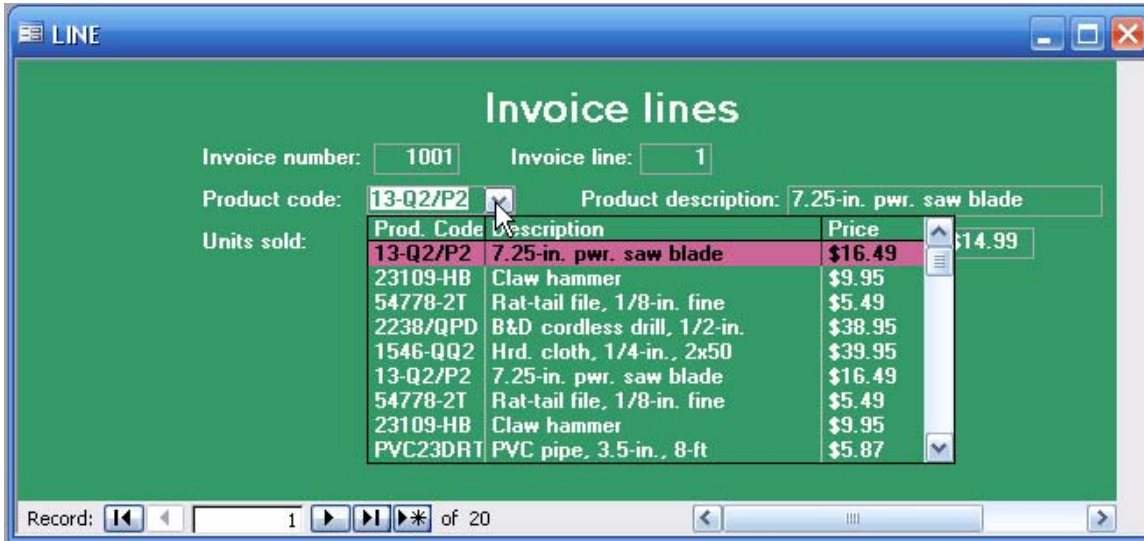
Figure 155 Combo Box in Form View

The screenshot shows the form view of the 'LINE' form. The data displayed is as follows:

- Invoice number: 1001
- Invoice line: 1
- Product code: 13-Q2/P2
- Product description: 7.25-in. pwr. saw blade
- Units sold: 1
- Sales price: \$14.99
- Sales amount: \$14.99

The record navigation bar at the bottom indicates 'Record: 1 of 20'.

Figure 156 Activated Combo Box



3.1.5 Controlling Input via List Boxes

You saw in the previous section that you can control input via a combo box. You can accomplish the same task through a list box. (Generally, list boxes are used to control input for items with a relatively small number of values that may or may not be generated in a query or stored in a table. To demonstrate the creation and use of a list box, start by editing the CUSTOMER table to add a CUST_TITLE field, with a field size of 8. The results are shown in Figure 157.

Figure 157 Added CUST_TITLE Field

Field Name	Data Type
CUST_CODE	Number
CUST_TITLE	Text
CUST_LNAME	Text
CUST_FNAME	Text
CUST_INITIAL	Text
CUST_ADDRESS	Text
CUST_CITY	Text
CUST_STATE	Text
CUST_ZIPCODE	Text
CUST_AREACODE	Text
CUST_PHONE	Text
CUST_BALANCE	Currency

The procedures for creating a list box are similar to those used in the creation of a combo box. One major difference is the starting point ... click on the **List Box** button in the **Toolbox** to get the process started, then drag a space for the list box just as you did for the combo box. Figure 157A shows these features based on the frmCUSTOMER form:

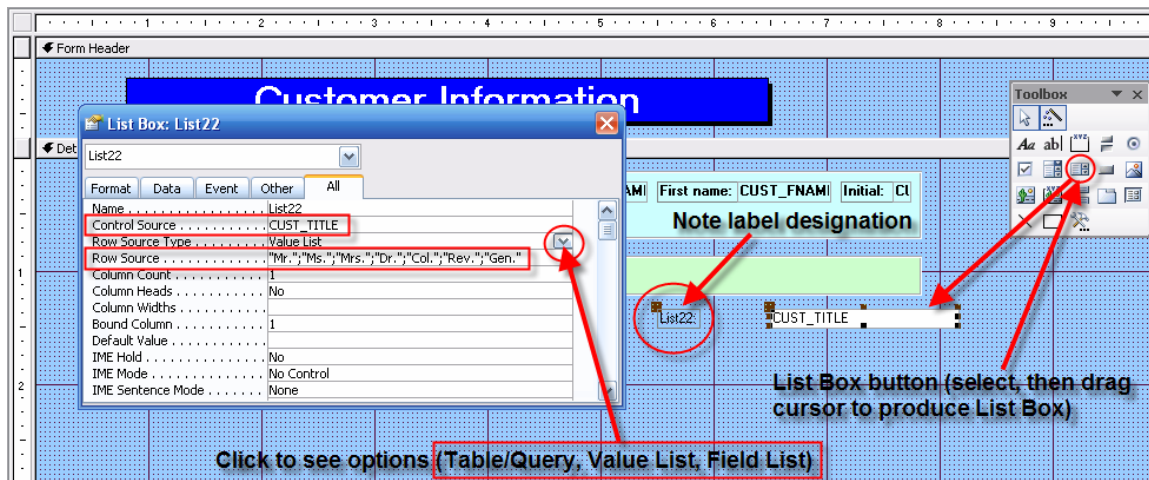
1. The list box was created and that its properties show a CUST_TITLE **Control Source**.
2. The **Row Source Type** is a **Value List**. (You can click on this property and then click on the resulting down arrow to see the other row source type options.)
3. The list of values is shown in the **Row Source**. Note that the available options are “Mr.”; “Ms.” ... and so on. The value list option was selected because there are only a few available values. If

the list of values had been long, the values might be listed in a table created for this purpose and the row source type would then be a table.

4. Note that there is only a single column of values, so the **Column Count** is **1**.
5. The (default) **Bound Column** is **1** – there is only a single column available.
6. The value list does not have a column head, so the **Column Head** selection is **No**.
7. Change the **Name** of the list box shown in Figure 157A from **List22** to the more descriptive **nmCustomerTitleListBox**.

Save the **frmCUSTOMER** form as **frmCUSTOMER-ListBoxDemo** to preserve the original form for further experimentation.

Figure 157A Input Control via List Box



As you examine the modified form in Figure 158, note that the original **Customer title:** textbox has been deleted from the form and the new list box has been substituted.

Figure 158 Edited List Box

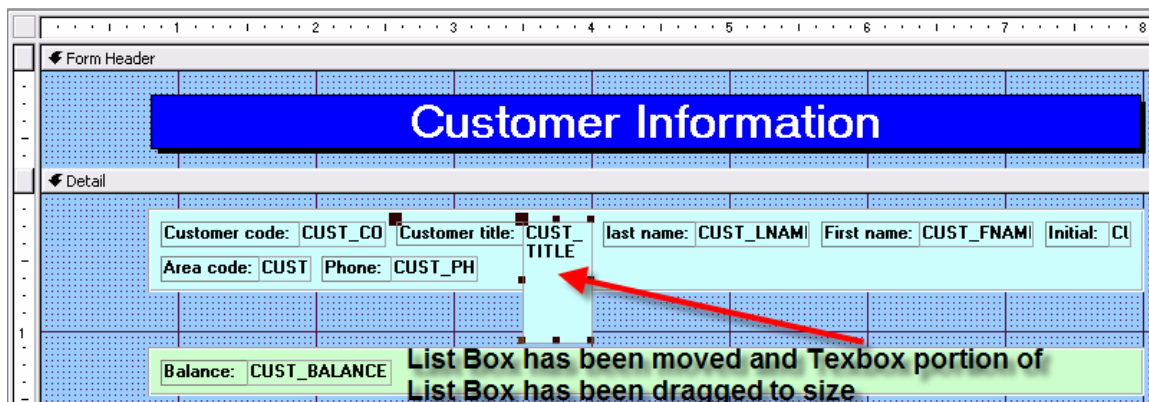
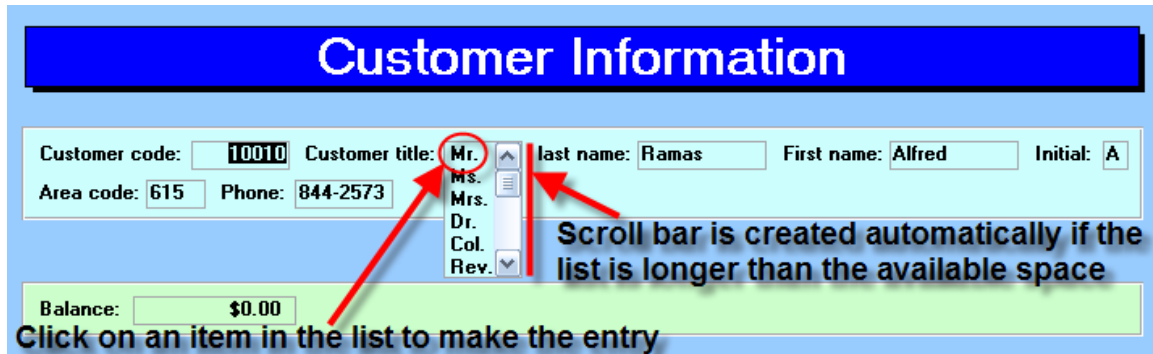


Figure 159 shows the results of the list box modification when the form is opened in its form view.

Figure 159 List Box in Form View



To select a customer title, simply click on an item of the list and then move to the next field. Use the scroll bar to select any item that may be located below the lower limit of the textbox. Figure 160 shows that several customer titles were entered via the list box.

Figure 160 Customer Table Entries via List Box

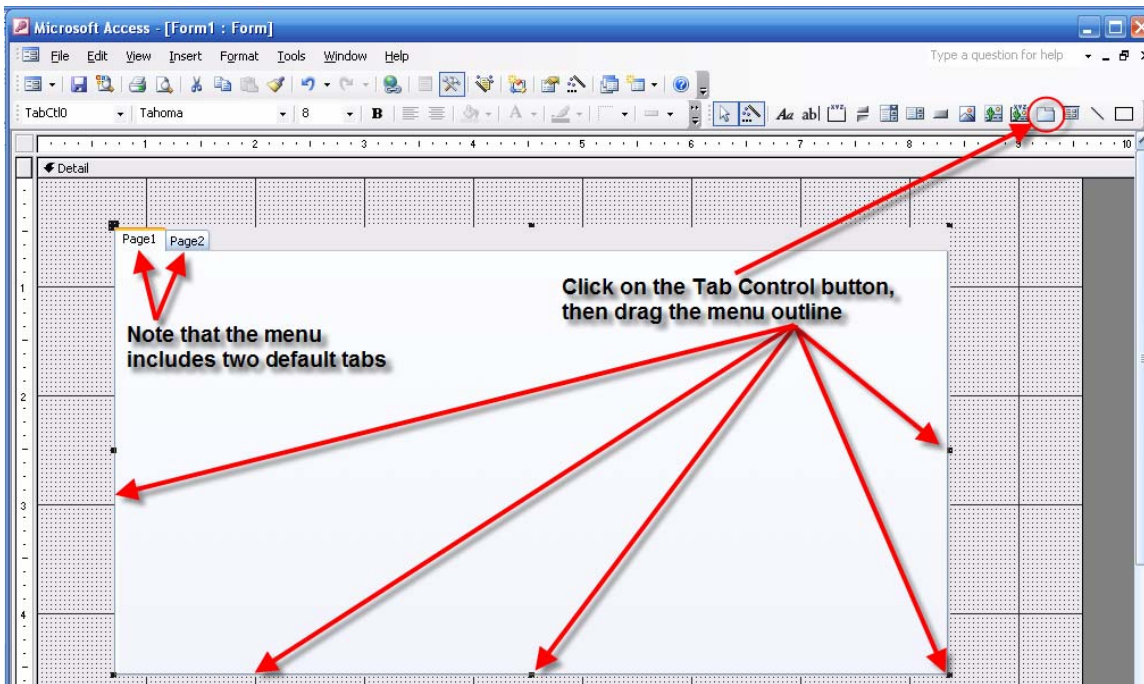
	CUST_CODE	CUST_TITLE	CUST_LNAME	CUST_FNAME	CUST_INITIAL	CUST_AREACODE	CUST_PHONE	CUST_BALANCE
▶ +	10010	Mr.	Ramas	Alfred	A	615	844-2573	\$0.00
+	10011	Ms.	Dunne	Leona	K	713	894-1238	\$0.00
+	10012	Ms.	Smith	Kathy	W	615	894-2285	\$411.02
+	10013	Mr.	Olowski	Paul	P	615	894-2180	\$536.75
+	10014	Col.	Orlando	Myron		615	222-1672	\$136.60
+	10015		O'Brian	Amy	B	713	442-3381	\$0.00
+	10016		Brown	James	G	615	297-1228	\$221.19
+	10017		Williams	George		615	290-2556	\$768.93
+	10018		Farriss	Anne	G	713	382-7185	\$216.55
+	10019		Smith	Olette	K	615	297-3809	\$0.00
*	0							\$0.00

3.1.6 Menus

If you want to make selected queries, forms, and reports easily available, a menu is a good way to get the job done. Creating menus is easy – just select **Forms/New/Design View**, then click **OK**. (*Do not select a table or query, because the menu will not be tied to any one query or table.*) When you click the **OK** button, you will see a “clean” design screen on which to create the menu.

Next, select the tool box, select the **Tab Control** option shown in Figure 161 – note that the cursor changes shape when you do that -- and draw the menu outline by dragging the menu cursor. Figure 161 shows the result of dragging the menu cursor. Note also that the initial use of the tab control automatically generates two tabbed menu pages, **Page1** and **Page2**.

Figure 161 Initial Menu Form in Design View



While the menu form is still selected, right-click on the **Page2** tab to pop up the options list you see in Figure 162. Select the **Insert Page** option. This selection will produce the **Page 3** tab you see in Figure 163.

Figure 162 Insert a Menu Page

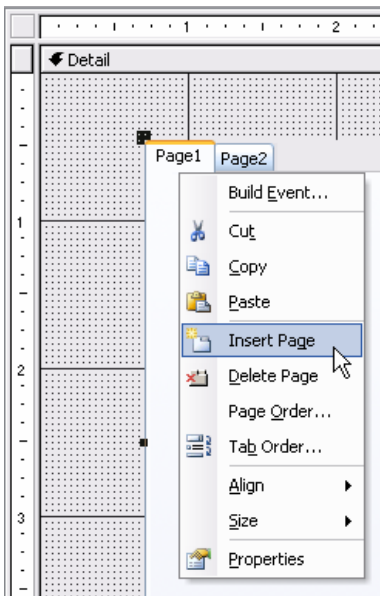
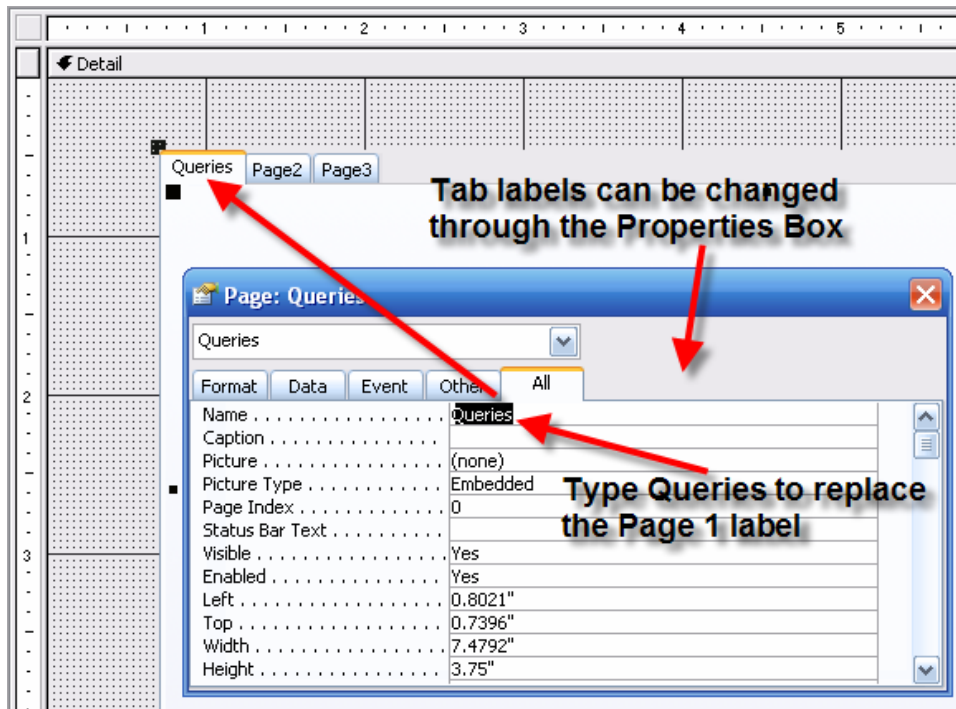


Figure 163 Edit the Tab Label Text

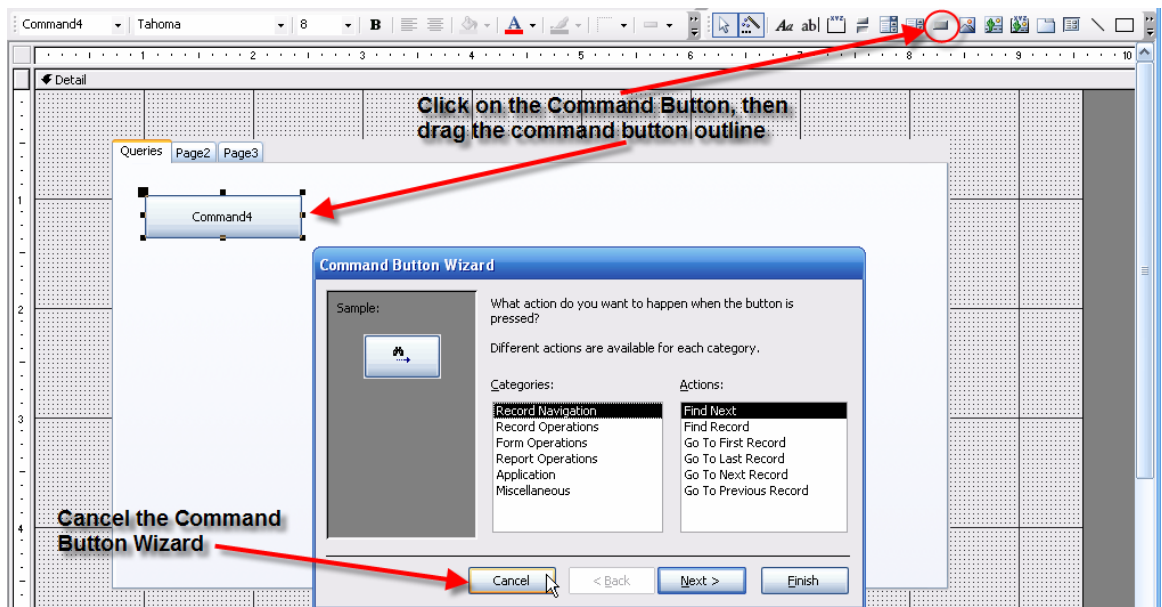
As you can tell by looking at Figure 163, the tab labels can be edited through the **Properties Box**. (As usual, the properties box is generated by right-clicking on the form component and selecting **Properties** from the list – that’s the last item you see on the list in Figure 162.)

You are now ready to place a few command buttons on the menu pages. Such buttons will be used to let the end user select a particular object – in this case, a query, a form, or a report. (You will learn about report generation in Section 3.4.) As the name *command* button suggests, clicking on such a button will generate a command that will execute the selected option. (That is, the command button will perform that duty when it has been linked to a macro or Visual Basic code. You will learn how to create and use macros in Section 3.5. This tutorial will not cover Visual Basic programming.) Figure 164 shows how a command button is created on the first (queries) page. Make sure that the first page of the menu form has been selected – click on the **Queries** tab.

Note

If you do not pay attention to the page you are supposed to be on, you may wind up placing command buttons on all pages simultaneously or you may place a queries command button on a forms or reports page. Placing a command button on the wrong page – or on multiple pages at once – will have major (and unwelcome) consequences when you later connect a macro to open a query and discover that it also opens a form or some other object. Therefore, if you intend to produce one or more command buttons on the **Queries** page, make sure that you select the tab *for that page* when you are in design mode.

Figure 164 Creating a Command Button



Go ahead and look at the form in form view. You can see in Figure 165 that the menu has a few unwanted properties such as record selectors. (The default design of a form also includes the now-familiar dividing lines – not shown here.) Therefore, “clean” the menu form by performing the actions indicated in Figure 166. Save the menu form as **frmMainMenu**.

Figure 165 The Initial Menu in Form View

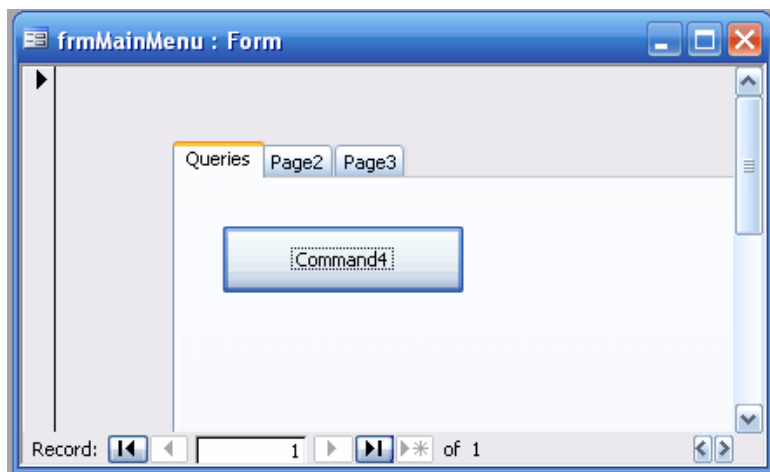
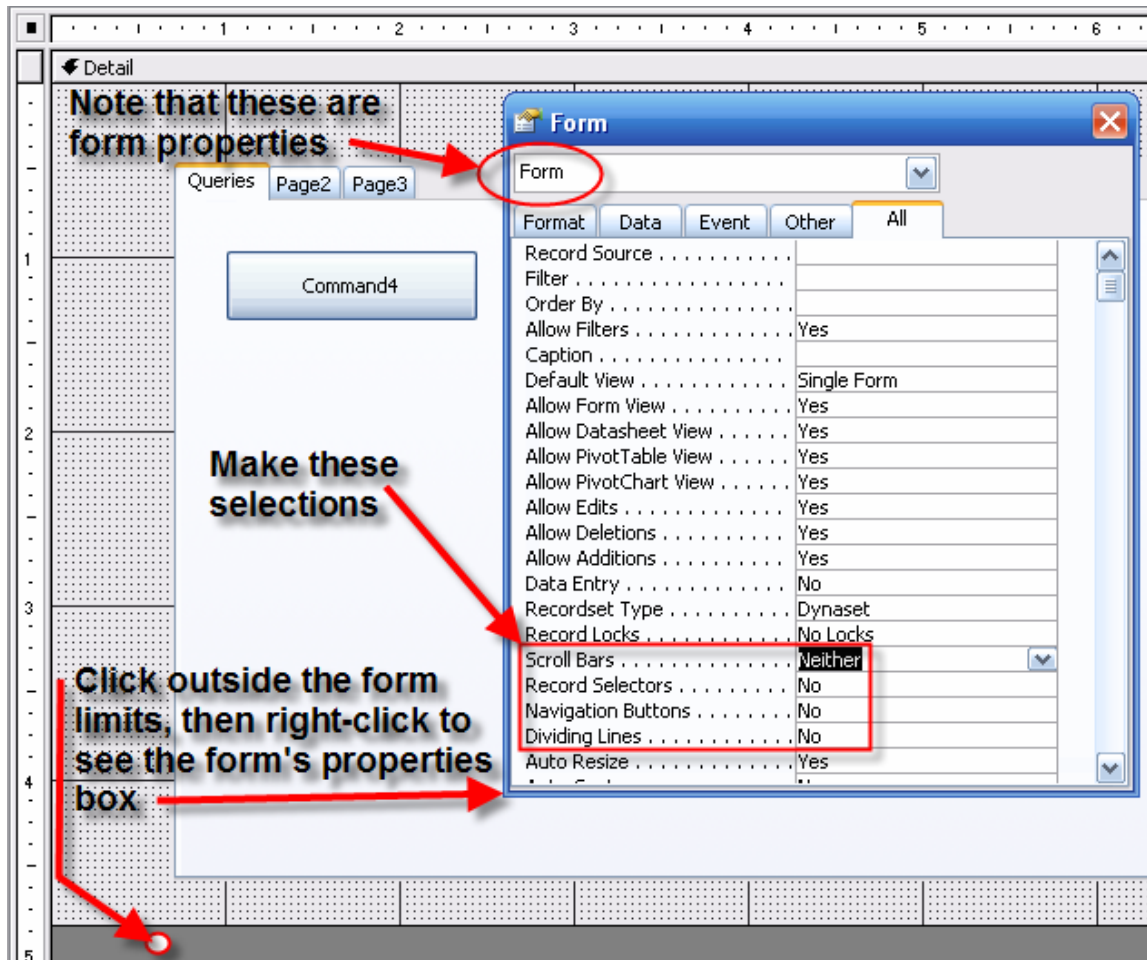
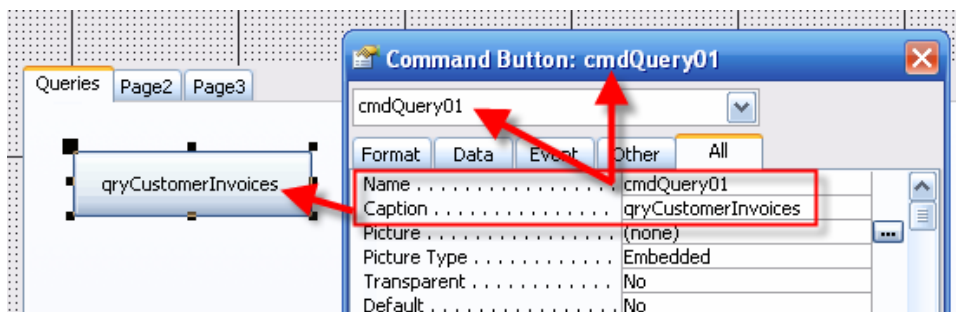


Figure 166 Clean the Menu Up



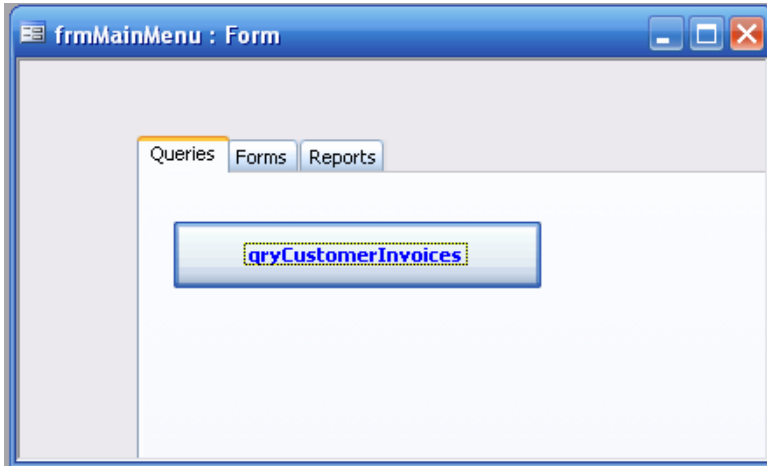
To edit the first command button's name and caption, stay in the design view, right-click on the command button to produce the properties box for that button. Change the name and caption as shown in Figure 167.

Figure 167 First Edited Query Command Button



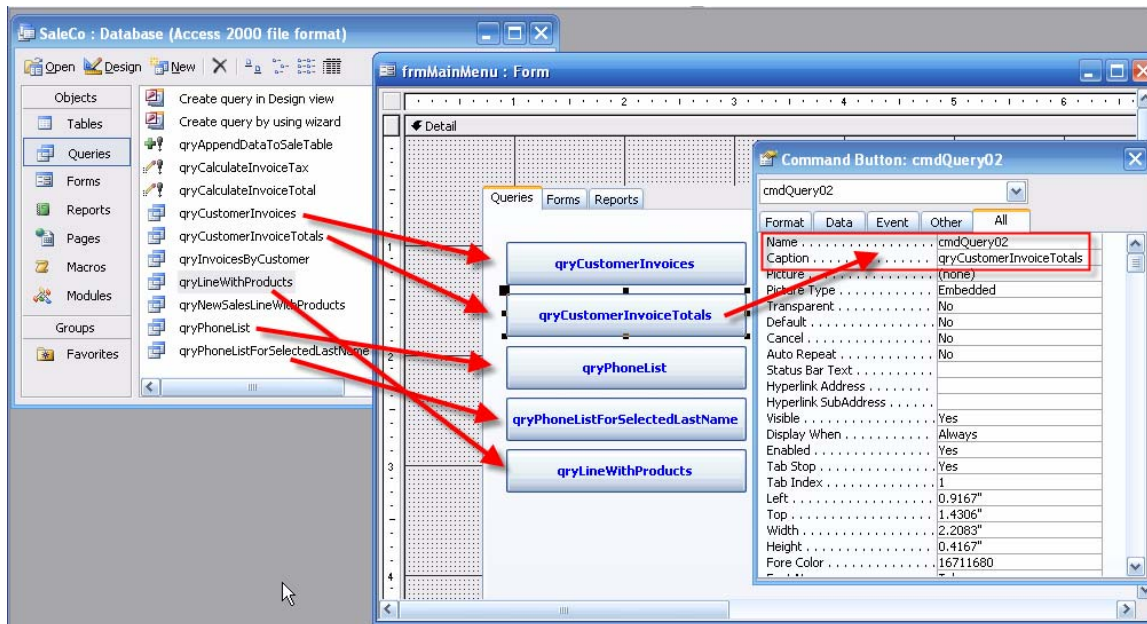
When you are done with the editing shown in Figure 167, continue the editing to produce the results in Figure 168. (You should know how to boldface the command button's text, how to change the text color, how to change the size and location of the command button by dragging its limits, and how to edit the tabs.) Note that this command button occurs on the Queries page only – the remaining two pages are still blank. (Go ahead and click from tab to tab to see the results.)

Figure 168 Additional Basic Edits



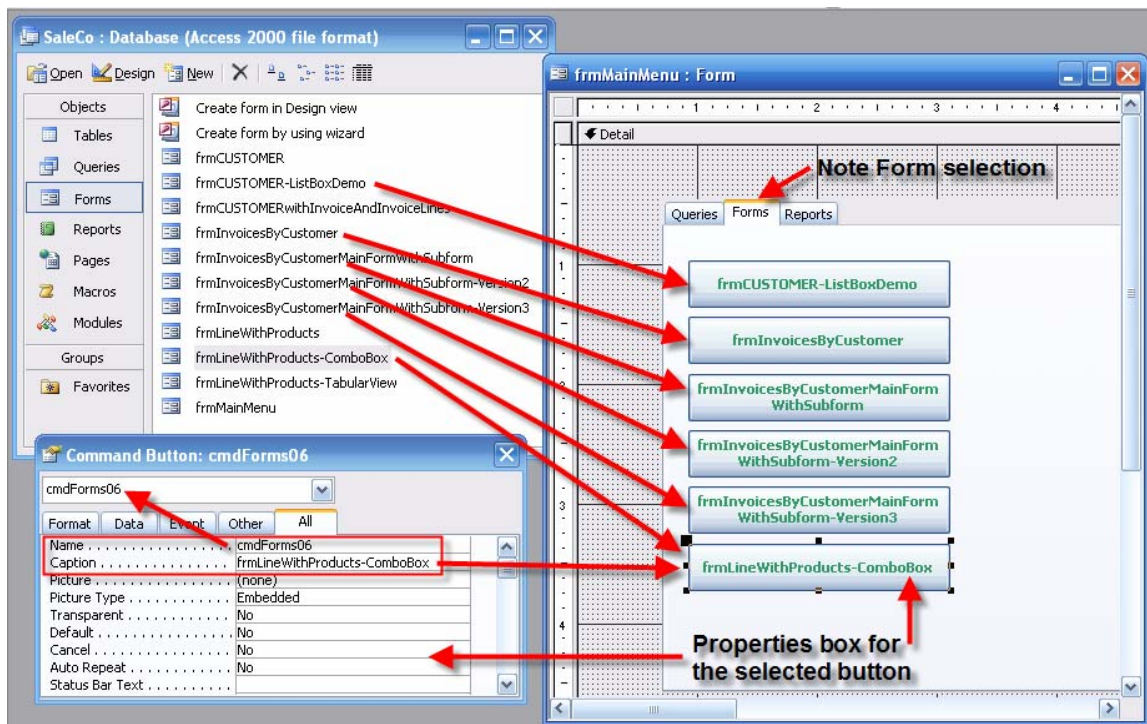
Let's create few more buttons for the **Queries** page. The easiest way to do this is to use a procedure you should know if you are familiar with Windows. That is, select the object in design view and then use the edit/copy/edit/paste routine to make and place copies. (Remember to make sure that you are on the Queries page!) Drag the buttons to their intended positions. Note that the editing routine ensures that all the buttons have the same properties. When you have made and placed the button copies, edit each to match the results in Figure 169. Don't forget to save your efforts from time to time. (Save the form as **frmMainMenu**.)

Figure 169 Multiple Queries Page Command Buttons



Next, copy the buttons you see here and paste them into the **Forms** page, then edit the buttons to match the forms that will be included. When you are done, your page – in design view – should match Figure 170. Note that the letter color has been changed to dark green and that this page has one more button than the query page.

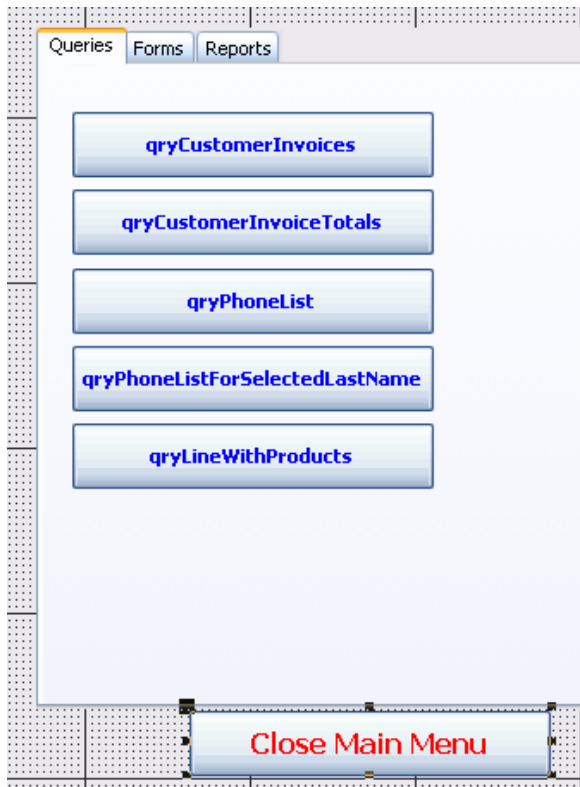
Figure 170 Form Page Command Buttons



You have not yet created any reports, so leave the **Reports** page blank. (You will learn how to create reports in Section 3.4.)

Next, let's create a **Close Menu** command button that shows up on each page. Figure 171 shows that a single button has been copied and pasted *outside* the pages.

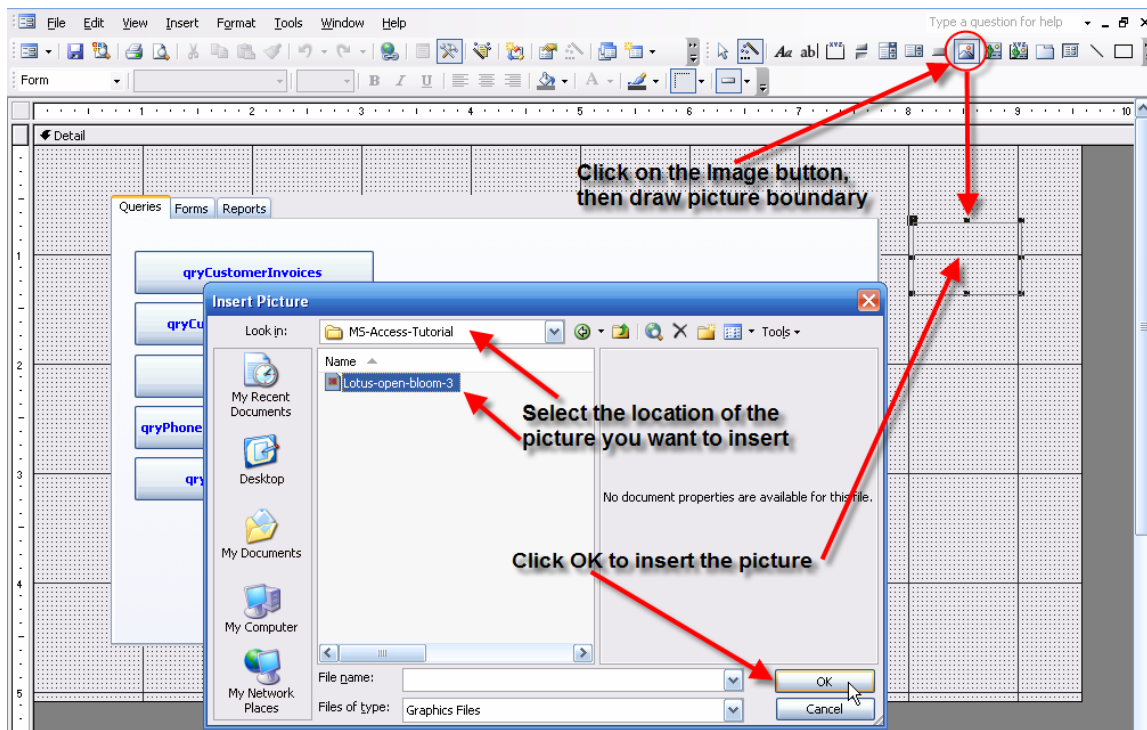
Figure 171 Close Menu Button Initial Location



When you drag the command button from its “outside the page” location and drop it on the first page, *it will show up on each page*. When this button is activated via a macro you will develop in Section 3.5, you will be able to close the menu from any page. (Go ahead and do the just-described drag-and-drop routine and then click through each pager and note that the **Close Main Menu** button is located on each page.)

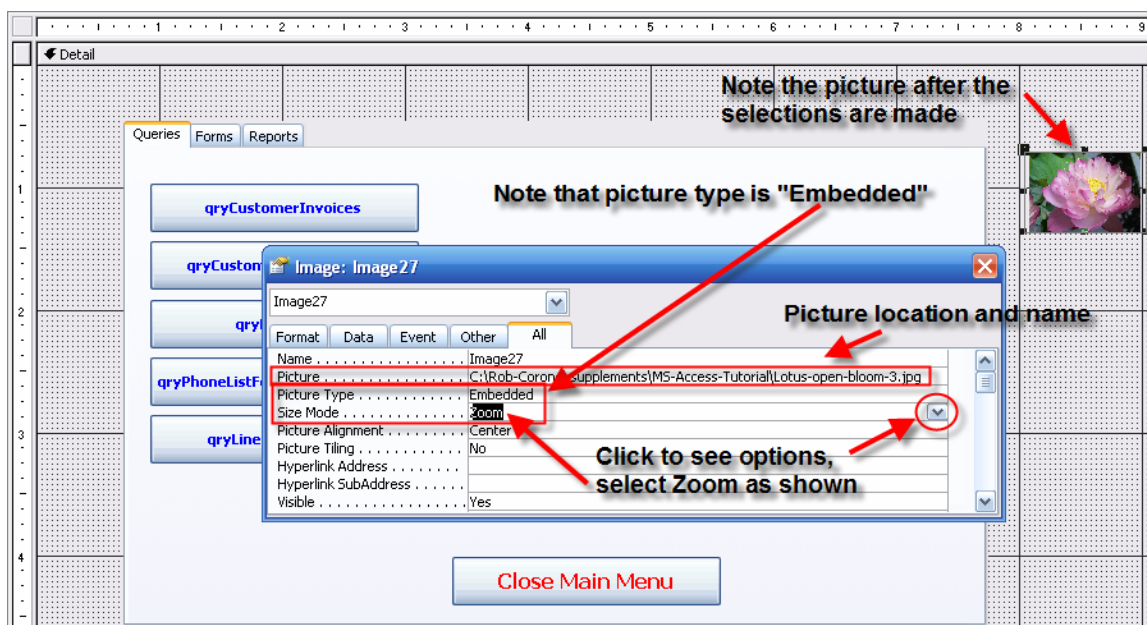
Finally, let's dress up the menu by using a picture. Any picture will do, but there happens to be a picture of a Lotus flower in the same folder as the database, so let's see how that picture may be inserted into the menu form. Figure 172 shows the procedure. Make sure that the picture frame is initially drawn outside the page limits and then drag-and-drop it on the first page to make sure that the picture shows up on all pages.

Figure 172 Insert a Picture



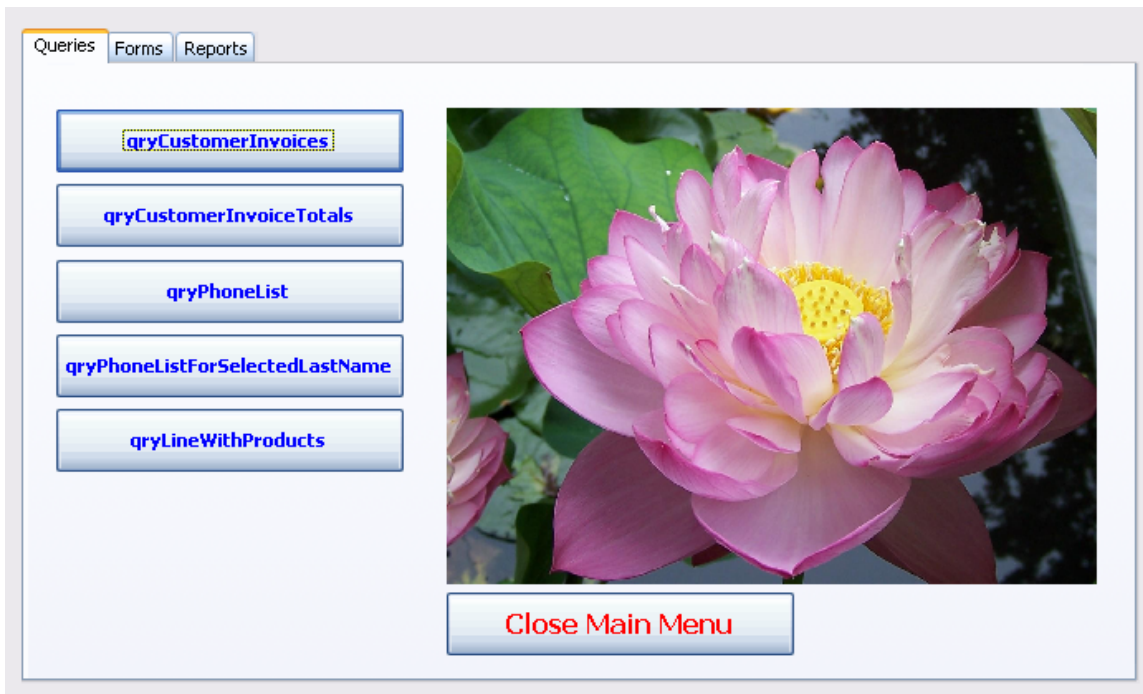
When you click **OK** as shown in Figure 172, the picture will be inserted ... but all you see is a small part of it. That's because the default setting is **Clip**. Use the picture's properties box to reset the presentation to **Zoom** as shown in Figure 173.

Figure 173 Formatted Picture



Note that the **Image** object in Figure 173 was created *outside* the menu form's detail section. **Therefore, if you drag this new object to any page, it will show up on all pages.** Go ahead and perform the drag and drop routine, then drag the image object's boundaries to enlarge the image as shown in Figure 174.

Figure 174 Completed Menu in Form View



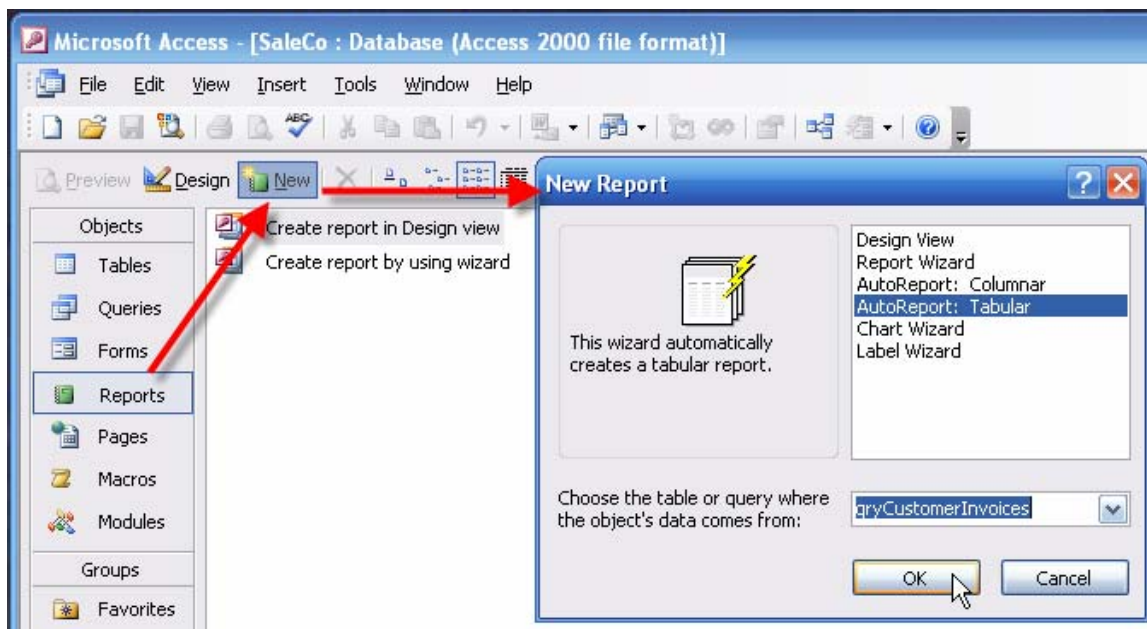
You now have an attractive menu that will become the hub for managing the Access applications. (You can move from menu page to menu page by clicking on the tabs.) Save the form again when you are satisfied with the form's layout. (The form was saved earlier as **frmMainMenu**.)

4.1 Reports

Although reports often contain the same information as forms, they do have several advantages. First, it is much easier to show multiple-record information in reports than in forms. Second, given their layout, reports are much easier to print than forms. In addition, if you have a lot of numeric data to present, the Access report format enables you to produce subtotals, totals, and grand totals as the report is generated.

The Access report wizard is excellent and it requires little effort on your part to produce a usable report. The report generating sequence – **Reports/New** -- is shown in Figure 175. Note that the data source is a query named **qryCustomerInvoices**.

Figure 175 Designing a New Report



As soon as you click the **OK** button shown in Figure 175, Access will generate the report and show its output. Only a few of the report records are shown. (See Figure 176.)

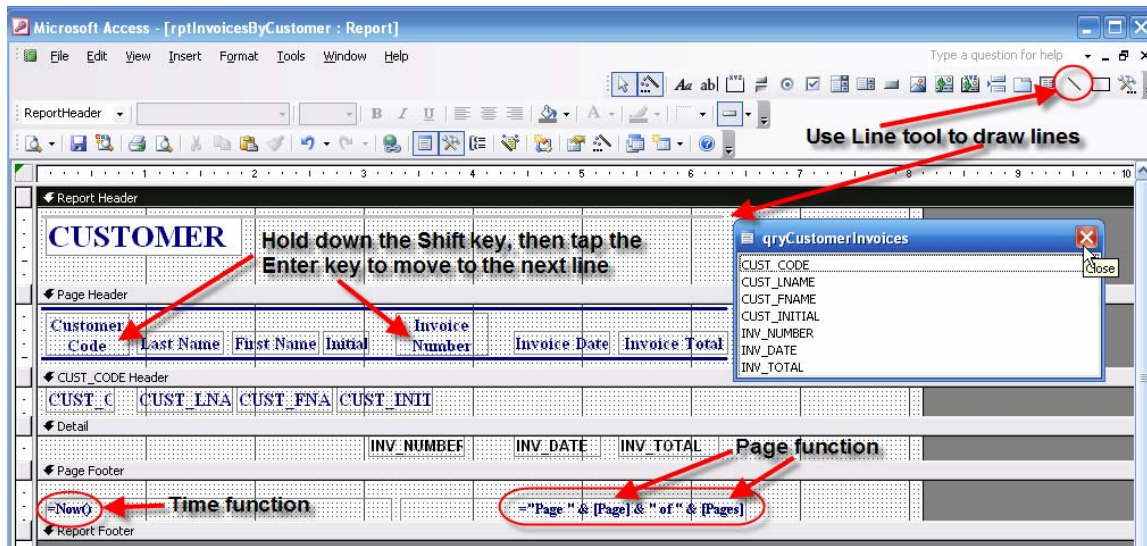
Figure 176 New Report Output

<i>CUSTOMER</i>						
<i>UST_CODE</i>	<i>CUST_LNA</i>	<i>CUST_FNA</i>	<i>CUST_INIT</i>	<i>V_NUMBER</i>	<i>VV_DATE</i>	<i>INV_TOTAL</i>
<i>10011</i>	<i>Dunne</i>	<i>Leona</i>	<i>K</i>			
				1008	17-Mar-06	\$431.08
				1004	17-Mar-06	\$37.66
				1002	16-Mar-06	\$10.78
<i>10012</i>	<i>Smith</i>	<i>Kathy</i>	<i>W</i>			
				1003	16-Mar-06	\$166.16
<i>10014</i>	<i>Orlando</i>	<i>Myron</i>				
				1006	17-Mar-06	\$429.66
				1001	16-Mar-06	\$26.94

Although the report output shown in Figure 176 is already quite usable, you should use the now-familiar design tools and practices to add functionality and eye appeal. (After all, end users are likely to “grade” your professional expertise on the basis of the presentation quality.) However, before you start editing the report, save the report as **rptCustomerInvoices**. Note that naming the report to match its source – in this case, a query named **qryCustomerInvoices** – maintains the self-documentation standard. (Note the **rpt** prefix denotes a **report**.)

Let’s begin the editing by improving the headers. Figure 177 shows that the font was changed and moved to improve the presentation format. Note also that Access has placed some handy functions at the bottom of the report ... you can move those to the top of the page later and then add the date function, **date()**. Because all the necessary fields have been included, go ahead and close the field list for the **qryCustomerInvoices**.

Figure 177 Edited Report Format



Save the changes you have just made and then open the report in Print Preview to generate the results shown in Figure 178. (Only a few records are shown. You may have to go back and forth between the design and print preview to help you line up the various text boxes.)

Figure 178 Print Preview of Edited Report

CUSTOMER						
Customer Code	Last Name	First Name	Initial	Invoice Number	Invoice Date	Invoice Total
10011	Dunne	Leona	K	1008	17-Mar-06	\$431.08
				1004	17-Mar-06	\$37.66
				1002	16-Mar-06	\$10.78
10012	Smith	Kathy	W	1003	16-Mar-06	\$166.16
				1006	17-Mar-06	\$429.66
10014	Orlando	Myron		1001	16-Mar-06	\$26.94

Let's do some additional report editing. First, move the time and page functions to the top of the page and add the date function as shown in Figure 179. Next, let's generate the invoice total *for each customer*; that is, you need a customer subtotal. Therefore, the invoices must be grouped by the CUST_CODE field and you must have a place – known as a *detail footer* -- to display the output. To create a detail footer, (right) click just outside the detail section to generate the options list for the detail section. The edits are shown in Figure 179 and the edit results are shown in Figure 180.

Figure 179 Additional Edits

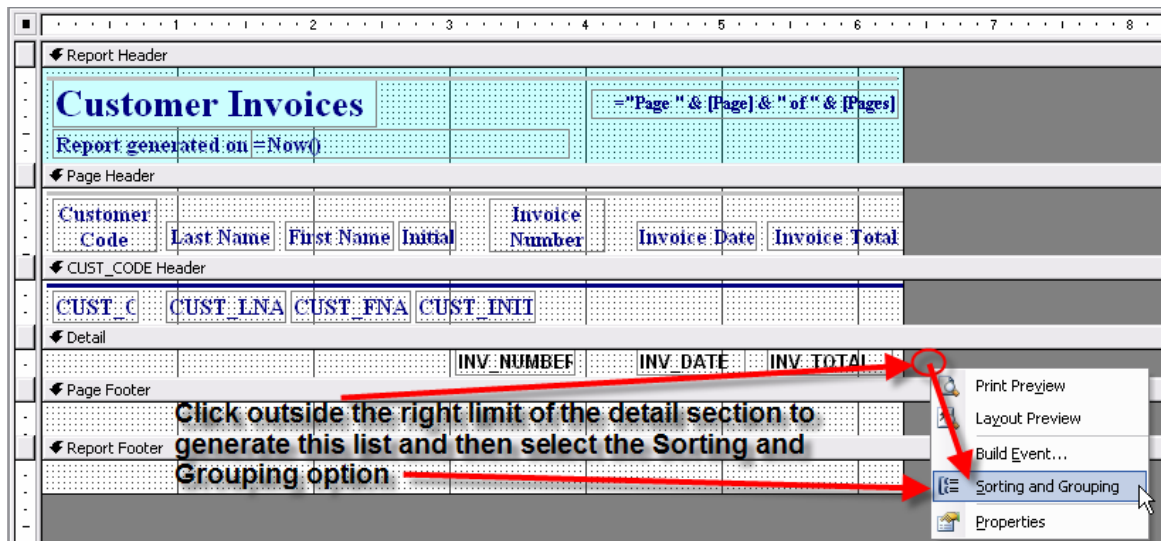
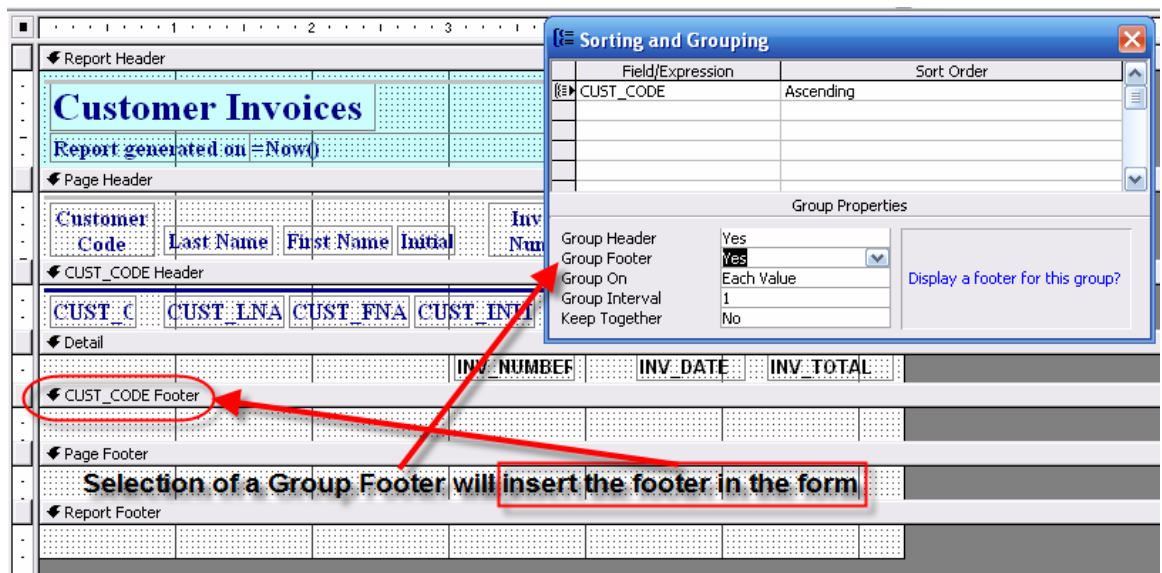


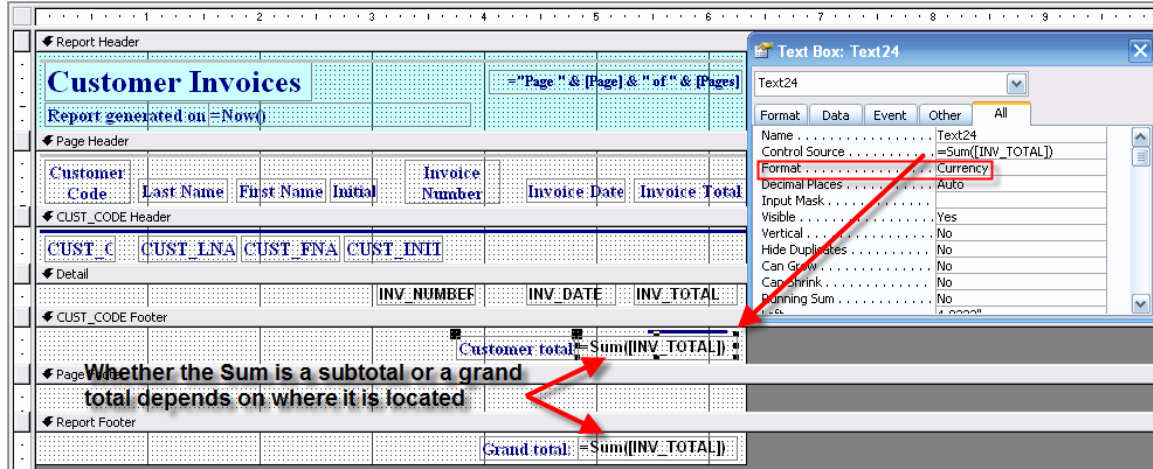
Figure 180 Addition of Section Footer



Now that you have added the section footer, you can insert subtotals for the section and then insert totals for all the customer invoices in the report footer. Note the textbox formats in Figure 181 and then look at the revised report's output. Make sure that you save the report again to preserve all the changes you

have made. Figures 181 and 182 show the various formatting changes and the final output. Save the report from time to time, using the report name **rptCustomerInvoices**.

Figure 181 Subtotals and Totals



As you examine Figure 181, note the creation of the (computed) customer total textbox and especially note its computation through the **Sum** function. *The same function is used to compute the grand total.* Aside from the fact that the two new text boxes have different labels, the important difference between the two textboxes – customer total and grand total – is their *location*. Figure 182 shows the report output.

Figure 182 Completed Report Output

Customer Invoices							Page 1 of 1
Report generated on Wednesday, March 29, 2006							
Customer Code	Last Name	First Name	Initial	Invoice Number	Invoice Date	Invoice Total	
10011	Dunne	Leona	K	1008	17-Mar-06	\$431.08	
				1004	17-Mar-06	\$37.66	
				1002	16-Mar-06	\$10.78	
				Customer total:		\$479.52	
10012	Smith	Kathy	W	1003	16-Mar-06	\$166.16	
				Customer total:		\$166.16	
10014	Orlando	Myron		1006	17-Mar-06	\$429.66	
				1001	16-Mar-06	\$26.94	
				Customer total:		\$456.60	
10015	O'Brian	Amy	B	1007	17-Mar-06	\$37.77	
				Customer total:		\$37.77	
10018	Farriss	Anne	G	1005	17-Mar-06	\$76.08	
				Customer total:		\$76.08	
10019	Smith	Olette	K	1009	27-Mar-06	\$195.20	
				Customer total:		\$195.20	
Grand total:						\$1,411.33	

4.1.1 Special Report: Labels

Printed customer address labels are a useful addition to any application set. (Such labels are useful when you mail out special promotions or billing statements.)

Access makes it easy to develop an effective and professional looking set of labels. Before creating the labels, make sure that you add the four marked fields in the CUSTOMER table as shown in Figure 183. Then create the query shown in Figure 183. This query will produce the output shown in Figure 184.

Figure 183 Customer Label Query Design View

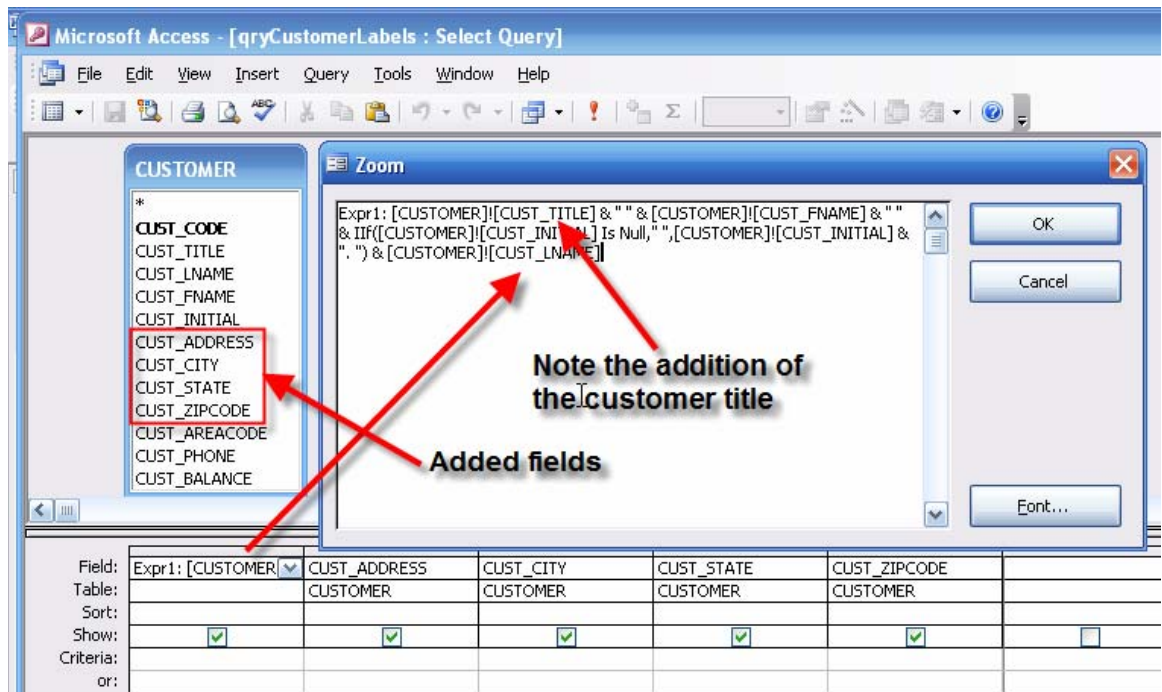
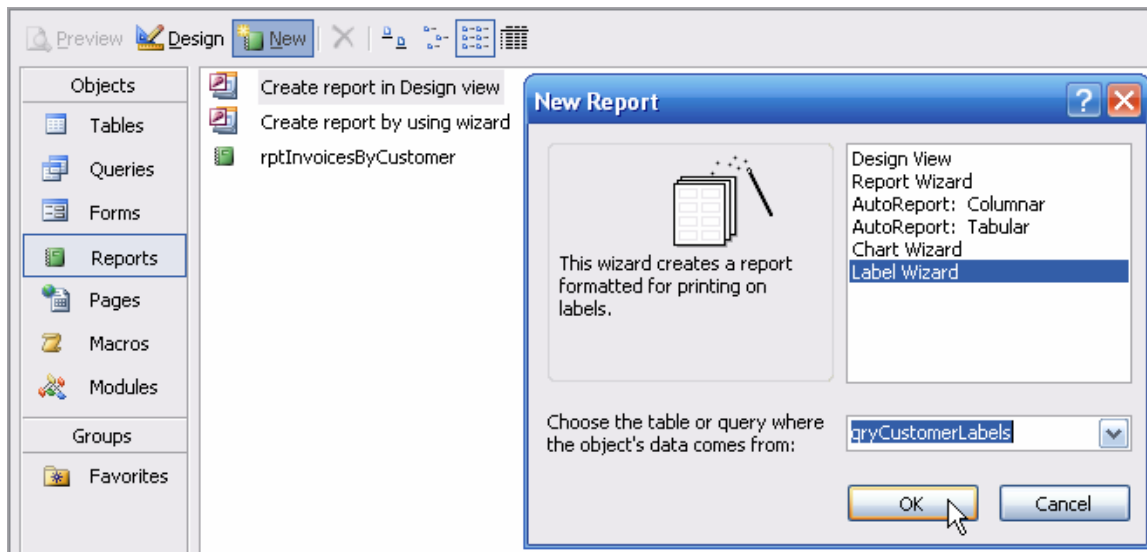


Figure 184 Customer Label Query Output

Customer Name	CUST_ADDRESS	CUST_CITY	CUST_STATE	CUST_ZIPCODE
Mr. Alfred A. Ramas	1124 Mountain View Rd.	Nashville	TN	37119
Ms. Leona K. Dunne	219 Twilight Lane	Nashville	TN	37123
Ms. Kathy W. Smith	389 Belle Glade Ct.	Beauville	KY	38976
Mr. Paul F. Olowski	1217 Main Street	Nashville	TN	37119
Col. Myron Orlando	3428 Mitchell Dr.	Nashville	TN	37123
Mrs. Amy B. O'Brian	2145 Meadow Lane	Nashville	TN	37228
Mr. James G. Brown	917 Twilight Lane	Nashville	TN	37119
Mr. George Williams	Box 2194	Smithville	TN	38003
Gen. Anne G. Farriss	453 Lotus Ct.	Carter	KY	38998
Rev. Olette K. Smith	890 Lockheed Avenue	Huntsville	AL	32892
*				

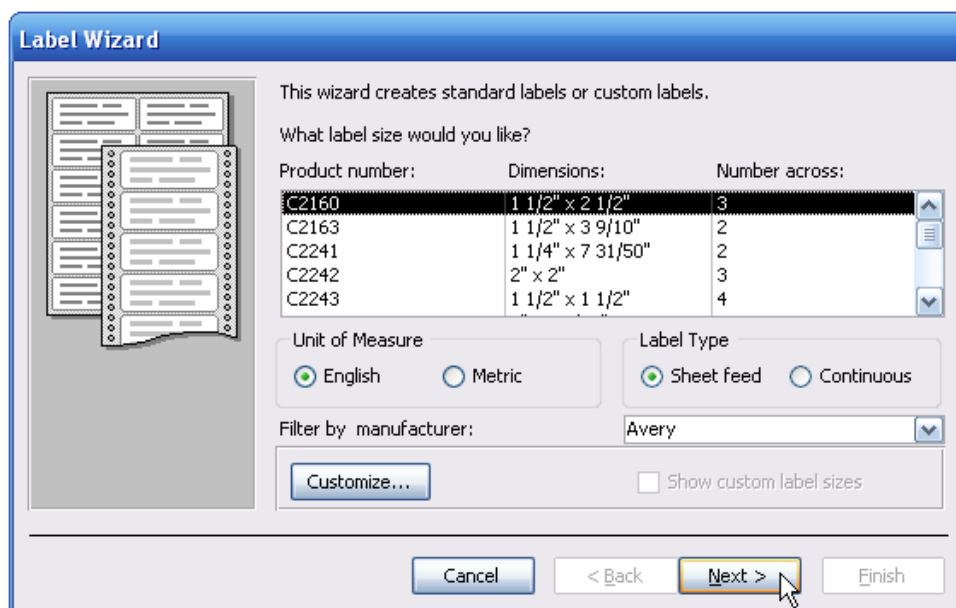
You are now ready to create the label report. In this case, the label wizard is a good place to start. (You will be able to edit the label wizard’s results later.) Figure 185 shows the start of the process. (Note that the query named **qryCustomerLabels** was selected to be the data source for the labels.)

Figure 185A Starting the Label Wizard



When you click on the OK button shown in Figure 185, you’ll see the label size selection dialog box in Figure 186. As you can tell the current selection is an **Avery** label set, three across the label page, using labels that measure 1 ½” x 2½”.

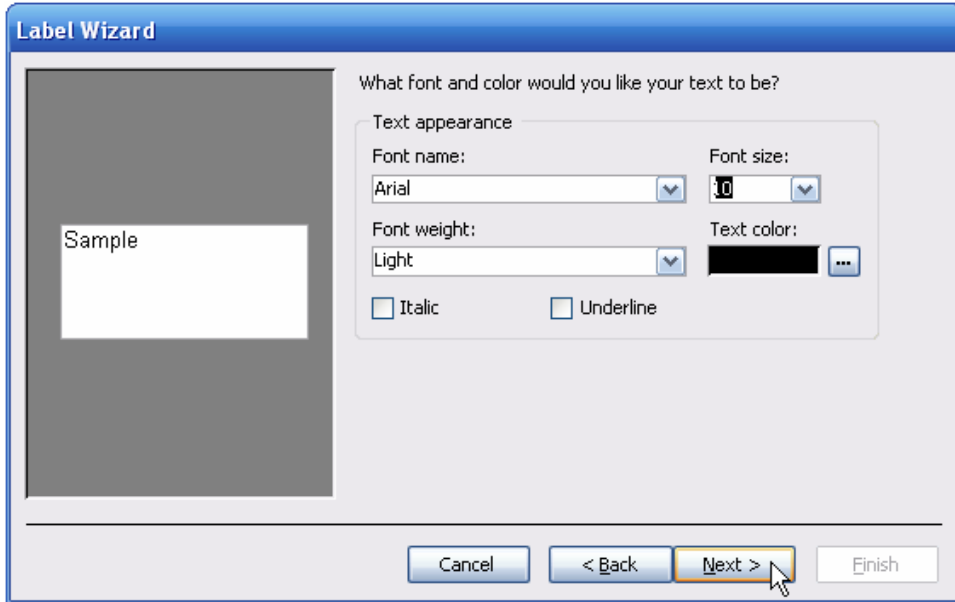
Figure 186 Label Size Selection



Because this is a very popular label format, go ahead and accept this selection by clicking on the **Next >** button in Figure 186 to generate Figure 187. (As you can tell by looking at Figure 186, there are quite a

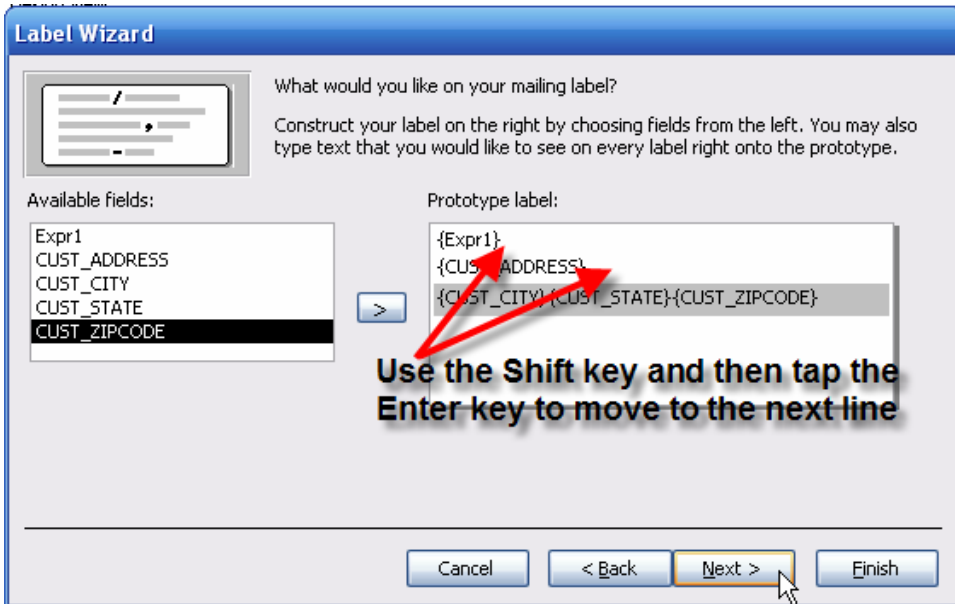
few label format options available – only five are shown here, but the scroll bar will reveal quite a few more.)

Figure 187 Font Size Selection



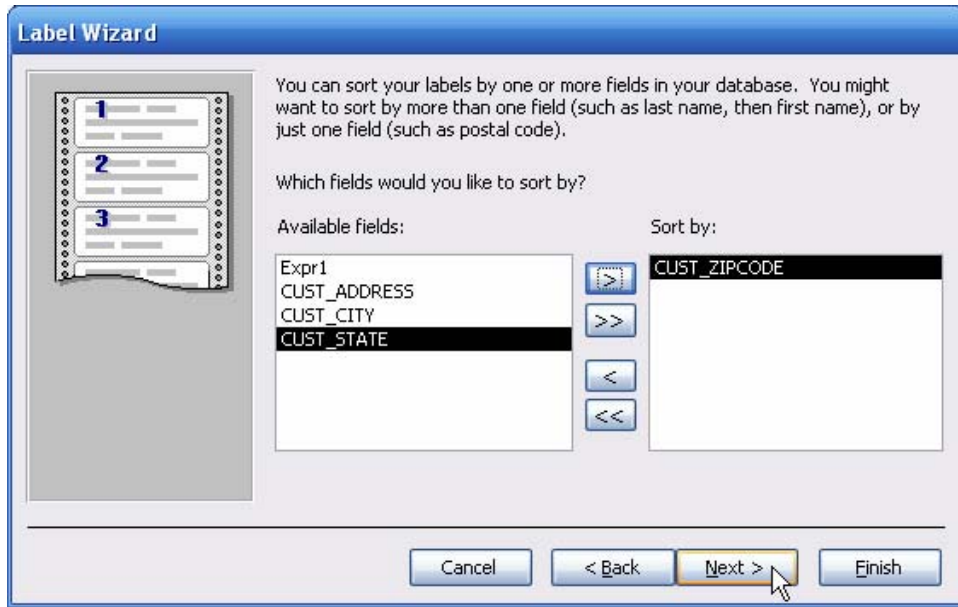
After selecting the label font in Figure 187, click on the **Next >** button in Figure 187 to generate Figure 188 to select the fields to be placed on the label.

Figure 188 Field Selection



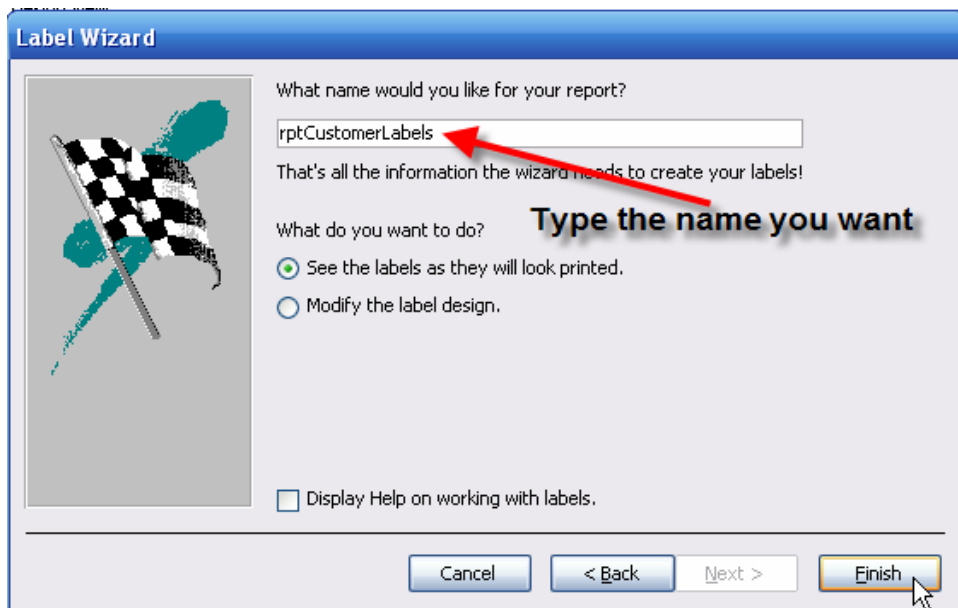
To place a field on the label, select it and then click on the > button shown in Figure 188 to move it to the **Prototype label**. The annotation in Figure 188 tells you how to get a line break so that you can move to the next label line.

Figure 189 Sort Selection



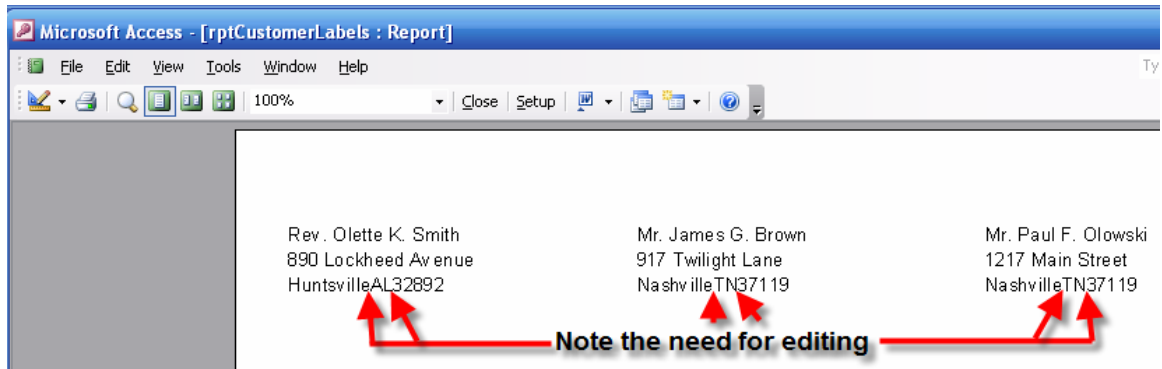
When you have moved all required fields to the **Prototype label** section, click on the **Next >** button in Figure 189 to generate Figure 190 and then type the name for the label report.

Figure 190 Type the Report Label Name



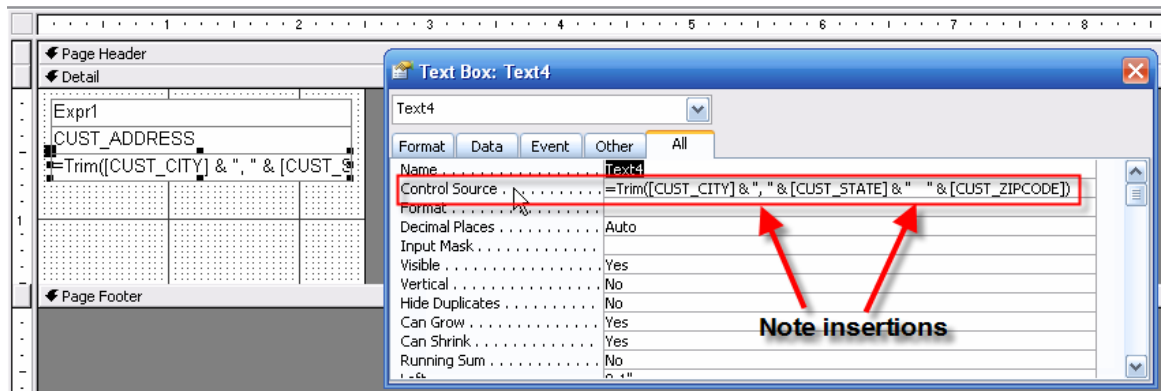
As you can tell by looking at Figure 190, the name **rptCustomerLabels** was used to reflect its use of the **qryCustomerLabels** query as the report's data source. Click on the Finish button in Figure 190 to save the report and to see its label output in Figure 191.

Figure 191 Editing Requirements



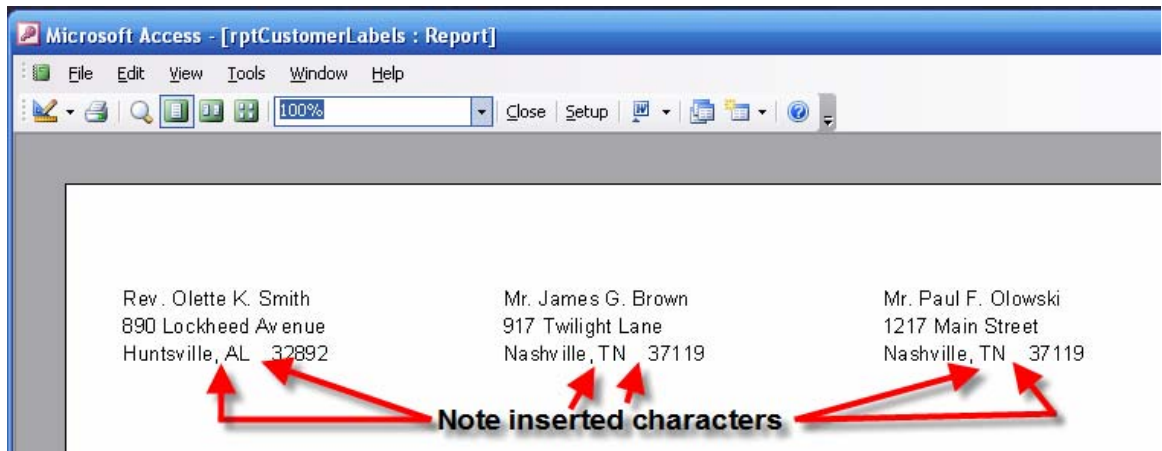
As you can tell by looking at Figure 191, the output needs some touch-up editing. The output would have a much more professional look if there were some space between the city and the state and between the state and the zip code. Also, putting a comma after the city name is standard. Therefore, open the label report in its design view and then generate the properties box for the third text box. Scroll down to the **Control Source** and make the insertions shown in Figure 192. (Use the **Expression Builder** to make the editing changes.)

Figure 192 Design View of Edited Label Report



After you have completed the edits in Figure 192, save the label report again and then open it in its **Print Preview** to see the output in Figure 193. Note that the output has a much more professional look than the one you saw earlier in Figure 191.

Figure 193 Print Preview of Edited Label Report



Using the techniques you have learned in this tutorial, you can easily modify the query to enter last and first name parameter entries and then to use this modified query as the data source for a label report that prints labels for one or more selected customers. For example, save the **qryCustomerLabels** query as **qrySelectedCustomerLabels** and then make the changes you see in Figure 193A. Next, save the **rptCustomerLabels** report as **rptSelectedCustomerLabels** and then use the new query as its data source. (See Figure 193B.)

Figure 193A Selected Customers Query

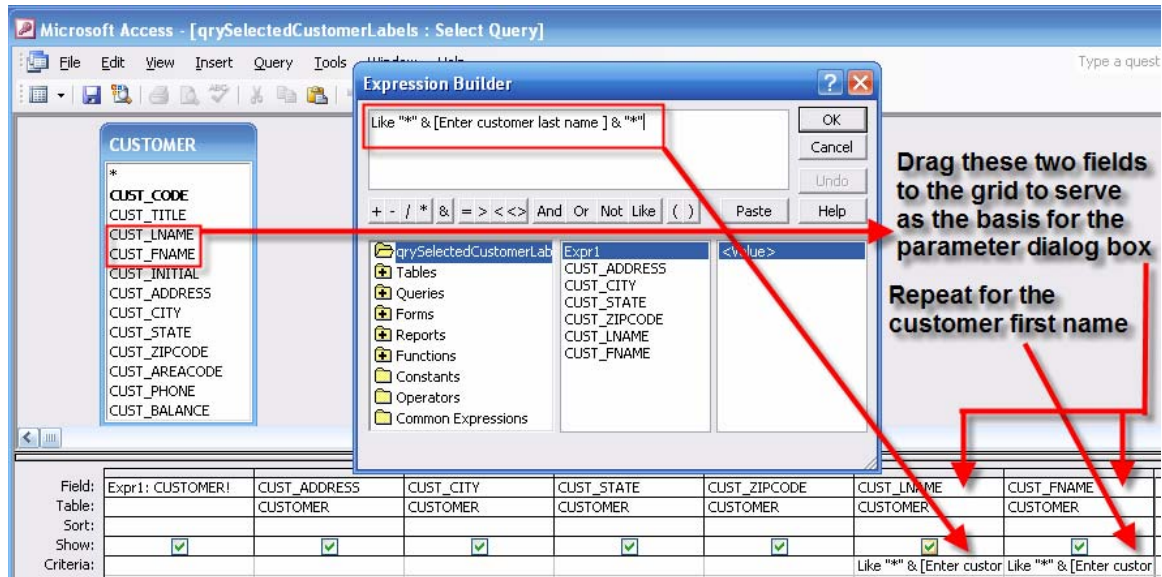
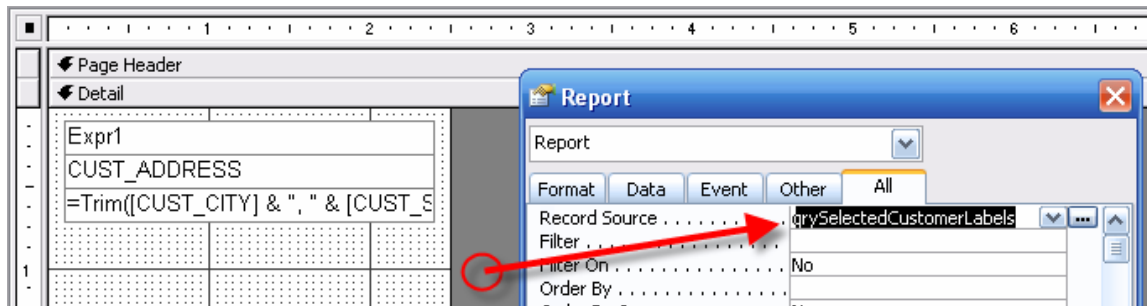


Figure 193B Selected Customers Label Report

Now add three command buttons on the main menu form's **Reports** page, one for the customer invoice report, one for the customer labels, and one for selected customer labels.

5.1 Macro Groups and the Macros within Them

Macros are code sets that let you perform a wide variety of actions that range from opening and closing forms, queries, and reports to calculating and inserting values on a form. You can even manage pictures with the help of macros. Given the wide range of actions you can take with macros, this tutorial covers only a few options. However, once you have seen how macros are created and used, you have a sufficient basis for exploring additional macro options on your own. Ultimately, the most valuable contribution of macros is that they help you tie the many database applications together to create a system. In this tutorial, that system is based on the menu form you created in Section 3.3.

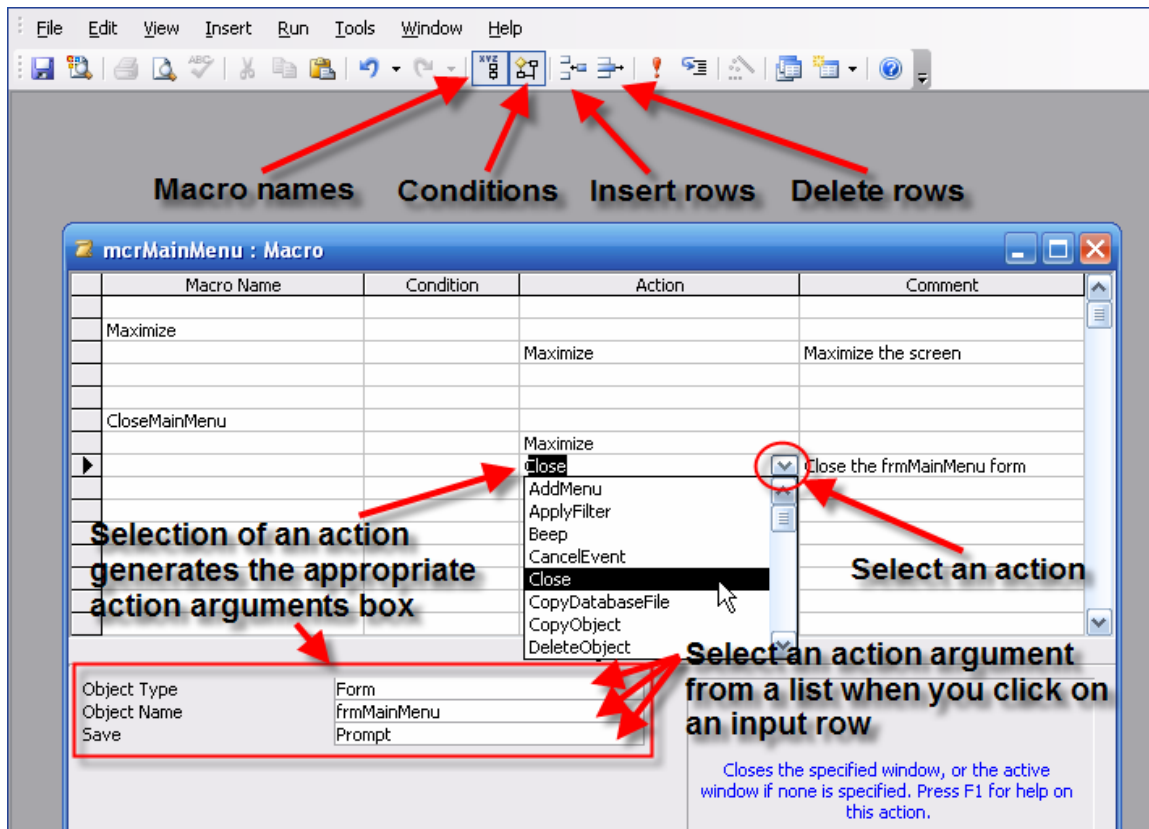
Remember that you should continue to strive to make the system self-documenting. Start with a plan to *organize the macros* you intend to create. The best way to organize macros is to make each set of related macros part of a named macro *group*. The macro group name should reflect the group's contents. In this case, you have created a menu that will help you access queries, forms, and reports. At least one of the macros will control a general menu action, such as closing the menu when you are done. Because you are about to create *macro* groups, it seems reasonable to use **mcr** as a preface to the macro group names. Therefore, it would be appropriate to begin with four macro groups: **mcrMainMenu**, **mcrMainMenuQueriesPage**, **mcrMainMenuFormsPage**, and **mcrMainMenuReportsPage**.

To create a macro group, select the **Macro** option from the now-familiar list of objects – see, for example, the list of objects in Figure 175 -- and then select the **New** option to open the macro design screen you see in Figure 194. (Actually, you will only see the **Action** and **Comment** portions when you first see the macro design screen. Click on the **Macro Names** and **Condition** buttons to show the four columns in Figure 194.)

Let's start with the general main menu form's anticipated actions. Let's suppose you want to make sure that the menu form is maximized when it opens and that you close the main menu form by clicking on the **Close Main Menu** button you created in Section 3, Figure 174. Figure 194 shows both macros, named **Maximize** and **CloseMainMenu**, respectively. In the interest of self-documentation, remember to use descriptive names that reflect the macro's activities. The macros are contained within a macro group that already has the **mcr** prefix, so the macro names within the macro group don't need to have

the prefix repeated. Access will automatically use the macro group name to label each macro within that group. Therefore, Access will reference the **Maximize** macro in the **mcrMainMenu** group as **mcrMainMenu.Maximize**.

Figure 194 Design View of Macro Group



To create the macro named **Maximize**, start by selecting a line in the **Macro Name** column and then simply type **Maximize** as shown in Figure 194. The action is to maximize the screen, so click on the line in the **Action** column to place a down arrow on that line. (Note the circled down arrow a few lines farther down in Figure 194.) Clicking one the down arrow will generate a list of all the actions that are available to you. Click on one of the options – in this case, **Maximize**, to transfer the selection from the list to the action line.

The second macro will close the main menu, so type **CloseMainMenu** in the macro name column as shown in Figure 194. This macro will include two actions. First, it will maximize the screen – just in case you decided to restore the screen manually – and, second, it will close the form. Note that **Close** is one of the action options on the list, so click on that option to transfer it to the action line. (You can also **type Close**, without selecting it from the option list. In fact, if you type the first few characters of any action command, Access will fill in the remaining characters.)

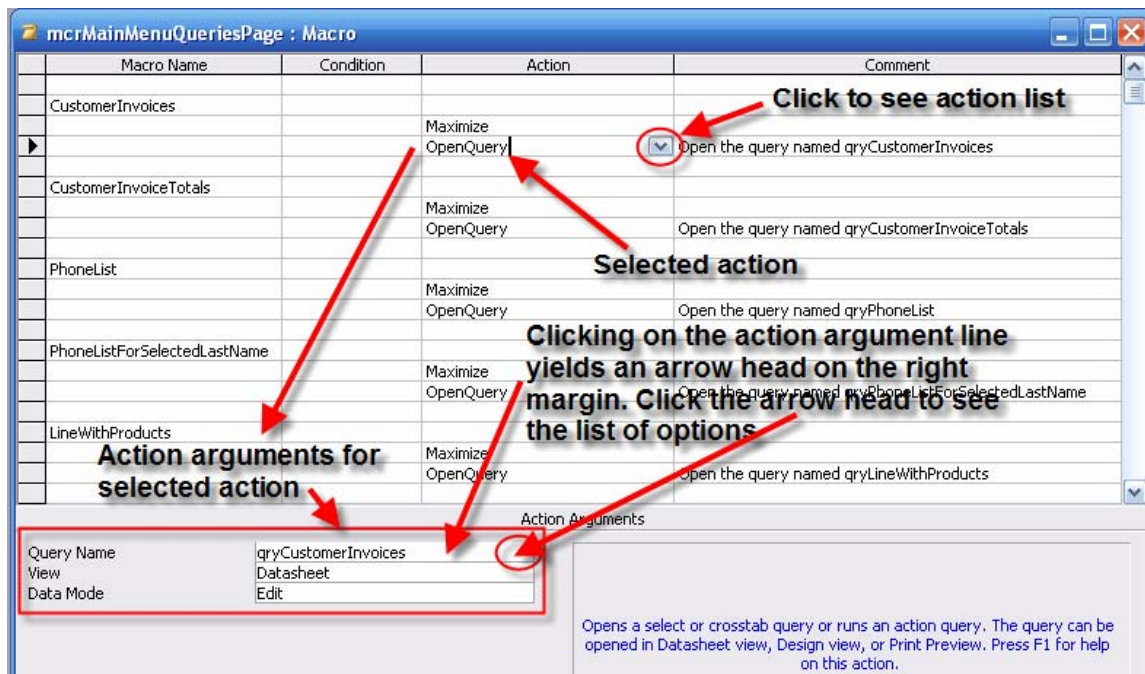
The **Close** action triggers an action argument box such as the one you see in Figure 194. In this case, the action argument has three components: an **Object Type**, an **Object Name**, and a **Save**. Clicking on the **Object Type** line will produce a down arrow again ... and clicking on that arrow will produce a list of

object types. Because this action involves the main menu *form*, select the **Form** option as shown here. Select the **Object Name** the same way – the object is a form named **frmMainMenu**, so that’s the one you select from the list. The **Save** option has a **Prompt** default selection, so go ahead and keep that default.

As you examine Figure 194 closely, note that the initial action is entered one line lower than the macro name. Although it is not necessary to do that, you will find that procedure very handy if you later decide to insert an action *before* the first one. (All you have to do then is put the cursor on the first action line below the macro name and tap the **Insert** button.) Also, note that a comment was typed in for each of the actions. Since these actions are very simple and are named to reflect their use, comments are not required here. However, it’s a good idea to use comments when the macro actions become more complex and have multiple components.

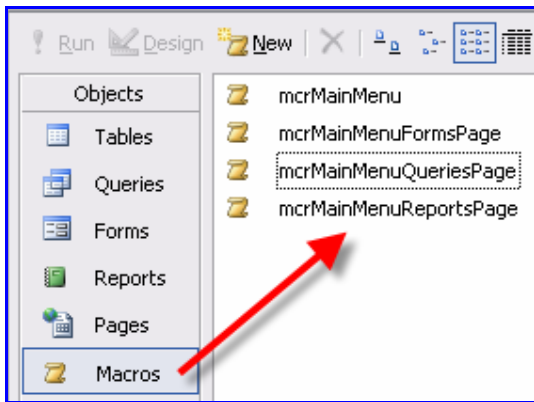
Go ahead and save the new macro group as **mcrMainMenu**. That’s it. You can now create the remaining macro groups the same way. Figure 195 shows the macros in the macro group named **mcrMainMenuQueriesPage**. Note that the **OpenQuery** action triggers an **Action Arguments** box that contains a **Query Name**, the desired **View**, and the **Data Mode**.

Figure 195 The Queries Macro Group



Repeat the process for the forms and the reports. Naturally, when you select the action **OpenForm** from an action list, the action arguments box contains references to *form* actions and when you select the action **OpenReport** from an action list, the action arguments box contains references to *report* actions. Save each macro group, using the macro group names suggested earlier in this section. When you are done, your macro groups will show up as listed in Figure 196.

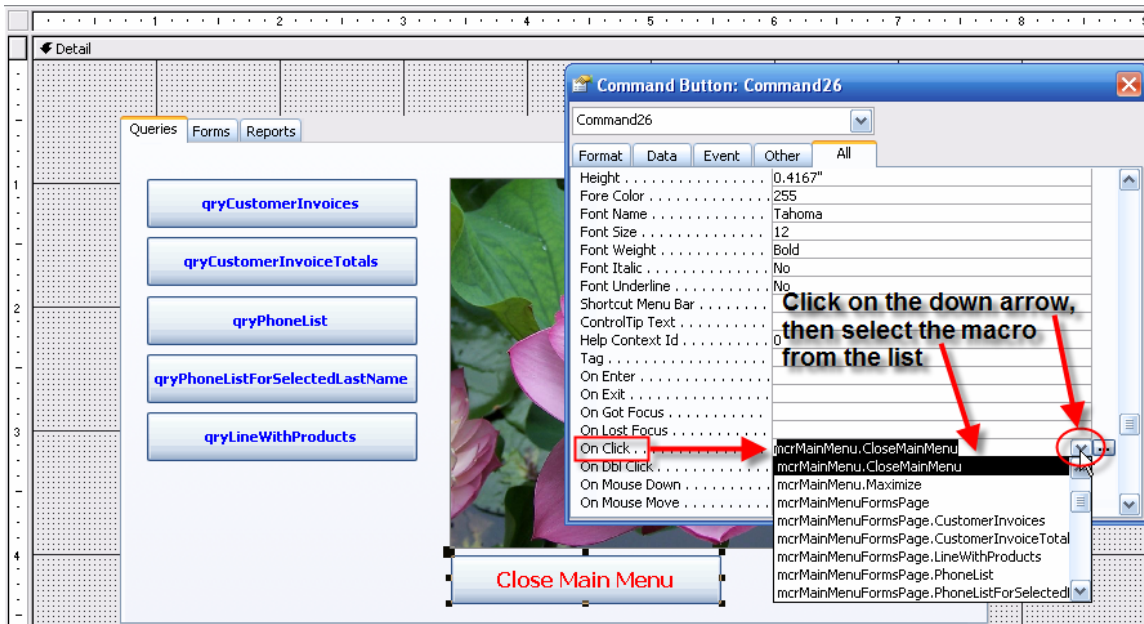
Figure 196 Completed Macro Groups



5.1.1 Attaching the Macros

To execute each macro within a macro group, it must be attached to the object that will trigger it. Let's take a look at the **Close Main Menu** button on the Main Menu form. When you click on that button, the main menu form should close. Therefore, note that the macro will be triggered via the "On Click" property in the properties box for that button. Figure 197 summarizes the process of attaching the macro to the **Close Main Menu** button on the **Main Menu** form.

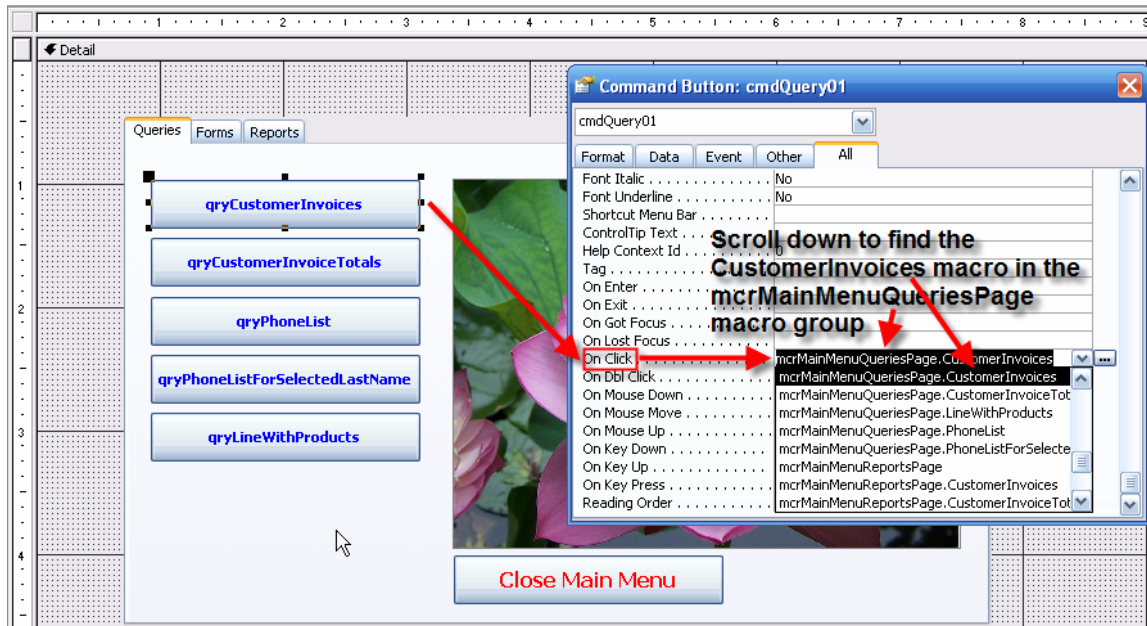
Figure 197 Attach the CloseMainMenu Macro



Now open the main menu in form view and click on the **Close Main Menu** command button and note that the macro closes the form as intended.

Use the same procedure to connect the appropriate macros in the [mcrMainMenuQueriesPage](#) and attach them to the command buttons on the main menu's query page. Figure 198 summarizes the procedure.

Figure 198 Attach the CustomerInvoices Macro



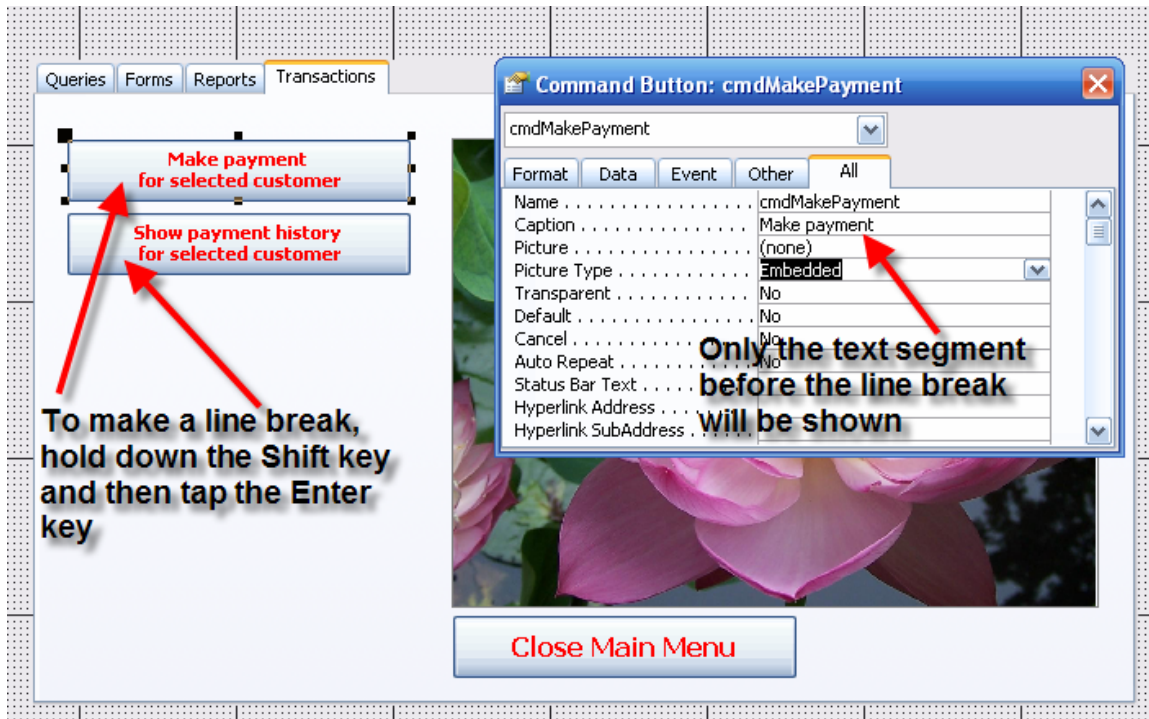
Repeat the process for the forms and the reports. You have now created a simple menu-driven system. Naturally, you can place command buttons on the forms and reports to let you close them as well as open them. (At this point, you have to close the forms and reports by using the Windows close option on the screens.)

5.1.2 A More Complex Set of Macros

Thus far, you have seen how macros can be used to perform simple actions, such as open and close queries, forms, and reports. However, macros can do a lot more than that. In this last set of examples, you will create some macros in a macro group named [mcrPayments](#) to manage customer payment on account.

Let's start by creating a new [Transactions](#) page with a [Make payment](#) command button on the main menu form. The new page and its command button are shown in Figure 199. If necessary, review Figure 162 to see how you can insert a menu page.

Figure 199 Main Menu Transactions Page

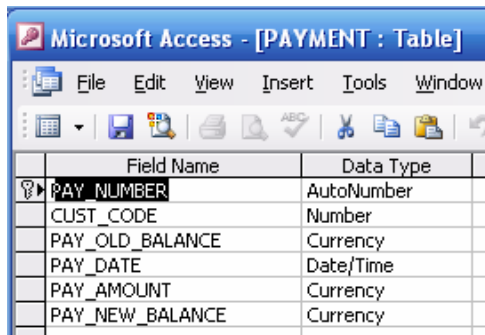


Now that you have created the new menu page, map out what you want the command buttons to do. The **Make payment** caption on the first command button shown in Figure 199 is largely self-documenting. But what precisely do you want to manage through this command option? Before you try to write the macros to accomplish the intended tasks, write a short, but precise, narrative. In this example, the narrative spells out what is to be done:

1. Find the customer who wants to make a payment.
2. When the correct customer is found, make a payment entry.
3. When the payment is made, the customer balance must be updated to reflect the payment.
4. The end user must be able to track all payments by customer, date, and amount.
5. The end user should enter only the payment amount – customer numbers, date entries, and new balance calculations must be done automatically.

Item 4 in the preceding list makes it clear that you need to create a new table in which to store all the payment transactions. Therefore, create a new PAYMENT table that includes the attributes shown in Figure 200. Note that the CUST_CODE is the foreign key in this new table.

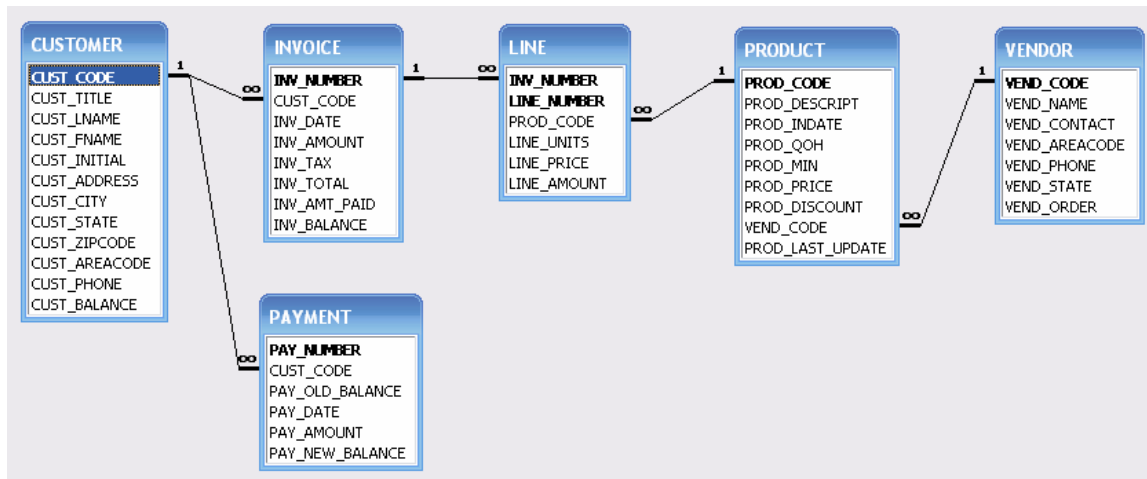
Figure 200 PAYMENT Table Structure



Field Name	Data Type
PAY_NUMBER	AutoNumber
CUST_CODE	Number
PAY_OLD_BALANCE	Currency
PAY_DATE	Date/Time
PAY_AMOUNT	Currency
PAY_NEW_BALANCE	Currency

Make sure that you relate this PAYMENT table to the CUSTOMER table, so your relationships window should look like Figure 201.

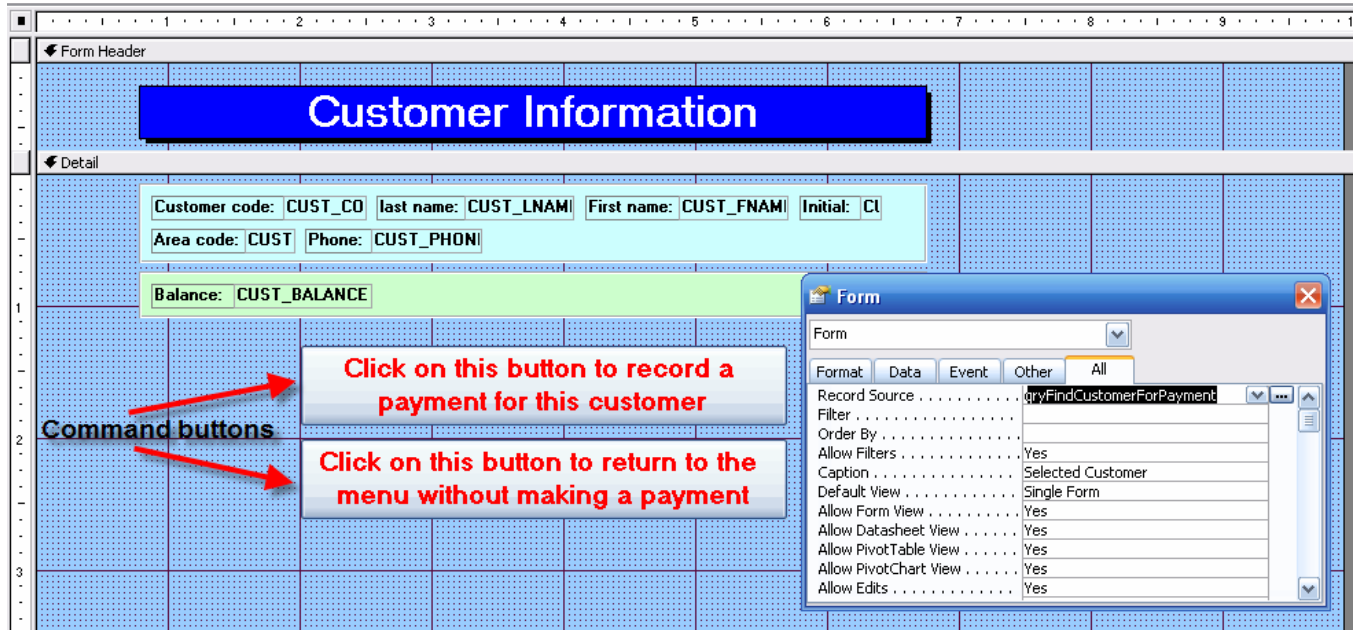
Figure 201 The Updated Relationships Window



If you click the **Make payments** command button shown in Figure 199, the first thing you will have to do is find the customer for whom the payment is to be recorded. Therefore, you must first design the **frmFindCustomerForPayment** form you see in Figure 202. This form's contents will be based on a parameter query named **qryFindCustomerForPayment**. This query uses all the CUSTOMER table fields. To enable the end user to select a customer by that customer's last name, use the following criteria statement for the CUST_LNAME field:

Like "*" & [Enter the last name] & "*".

Figure 202 frmFindCustomerForPayment Design View



Note that the form shown in Figure 202 uses two command buttons. The first command button will be used to execute a macro that lets the end user record a customer payment. (You will see how this macro works after the remaining form components are completed.) The second command button shown in Figure 202 lets the end user return to the menu without recording a payment. Clicking on this second command button merely closes the **frmFindCustomerForPayment** form and returns you to the menu.

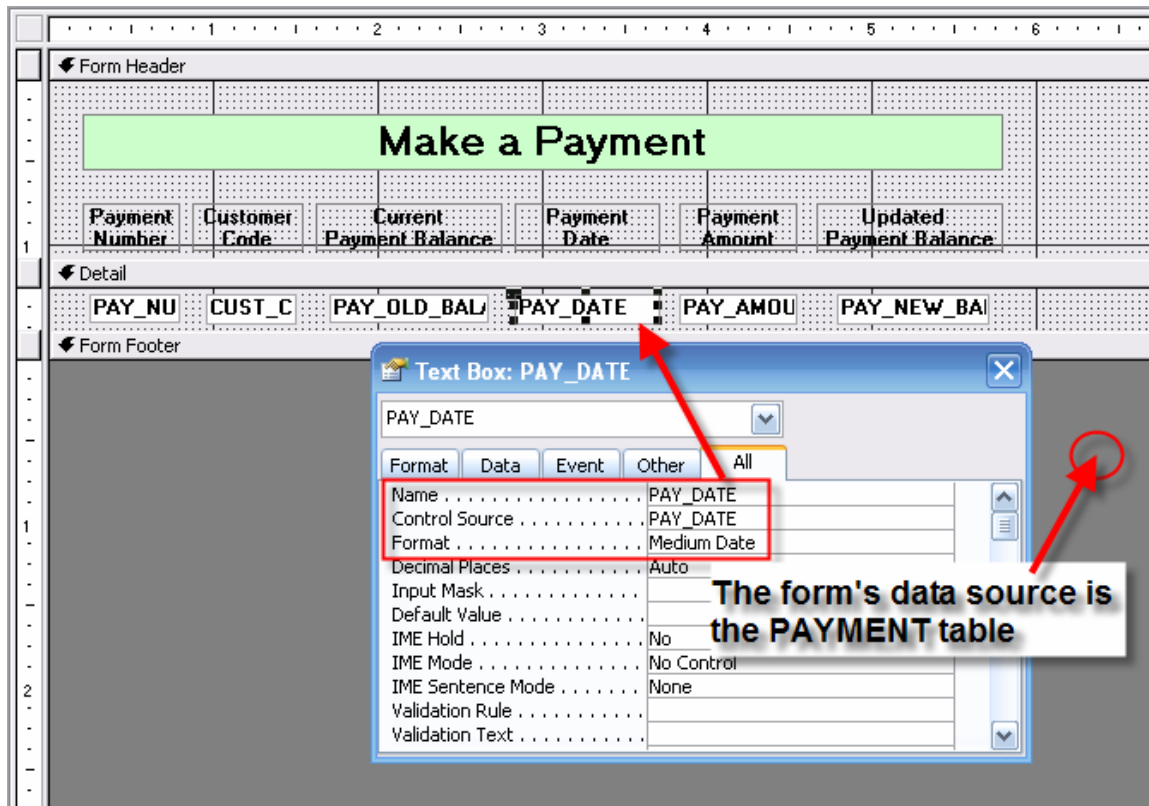
Note

If you open the **frmFindCustomerForPayment** form from the menu *without closing the menu*, that form simply covers up the menu form. (That’s why you want to maximize each form.) Therefore, if you close the **frmFindCustomerForPayment** form, the menu form becomes visible again. Because all of a form’s components, including its data components, are available in the computer’s memory, such “stacking” becomes a very handy tool to transfer values form one form to another.

If the end user clicks on the first command button shown in Figure 202, the macro must open a form that can record the payment. Therefore, make this form now. (Its structure is shown in Figure 203.) Note that the form’s data source is the PAYMENT table. (The data source for any form can be found by clicking outside its form limits.)

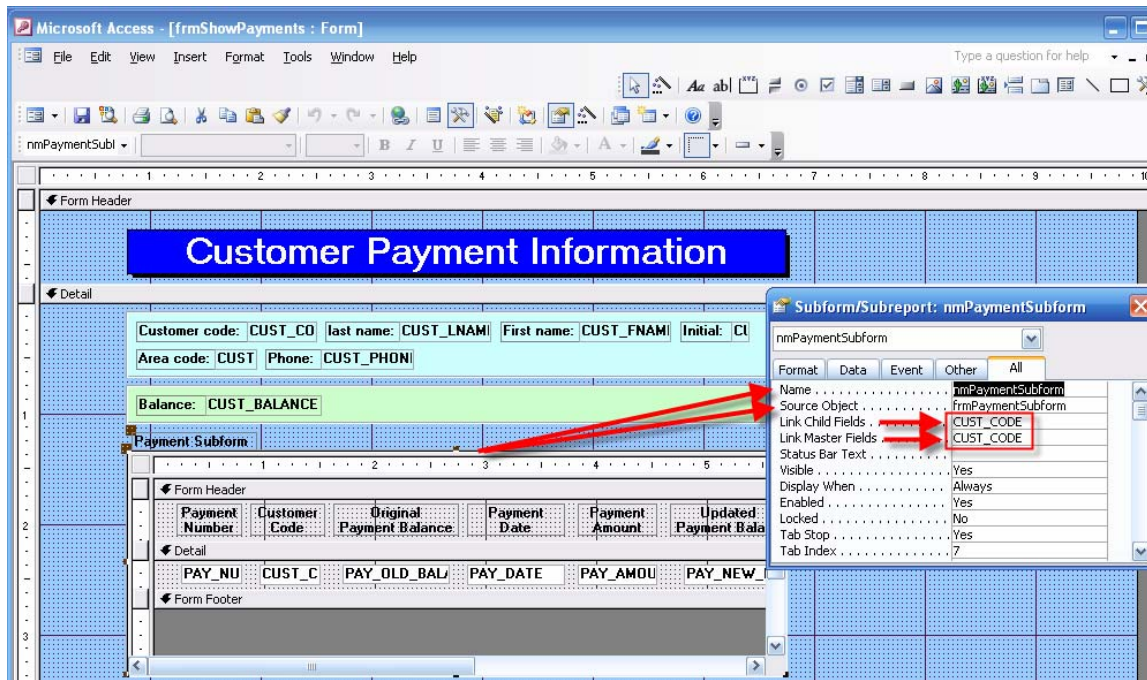
You want to ensure that the end user makes as few data entries as possible. For example, the payment number is entered through the **Autonumber** format and the payment date (PAY_DATE) uses the **Medium Date** default format. You will use a macro to automate the process of entering the customer code and the current payment balance. You will also use a macro to calculate the updated payment balance. Therefore, the only data entry will be the payment amount.

Figure 203 The frmMakePayment Form



If the end user clicks on the second command button shown in Figure 202, the customer information and the payment history must be shown. Therefore, create the appropriate main form as shown in Figure 204 and the subform as shown. Create the main form (**frmShowPayments**) first and then create the subform named **frmPaymentSubform**. Note the link between the main form and its subform in Figure 204. The data source for the main form is a query named **qryShowPayments** and the subform data source is the query named **qryPaymentSubform**.

Figure 204 The frmShowPayments Form with Subform



5.1.3 Payment Options Organization

All the macros required to process the payment transaction will be stored in the **mcrcPayments** macro group shown in Figure 205. Note that all the necessary macros are included and that the **Comments** section shows all the **Action Argument Item:** and **Expression:** information. (The details are simply copied from the **Action Arguments** boxes and pasted into the **Comment** lines to ensure that you can see all the components. To make the comments easier to read, the **Item:** and **Expression:** labels were typed in and the actual components of those elements were indented.)

Take a few moments to read the contents of all the macro components in Figure 205. Note that the **MakePayment** macro starts by opening the **frmMakePayment** form to a new record and then note that the **SetValue** action is used to write the customer code into the **frmMakePayment** form. (You should remember that the process started via the main menu and that the first job was to open the form named **frmFindCustomerForPayment** form. Because this form is still open behind the **frmMakePayment** form, you still have access to the CUST_CODE value. The **SetValue** action simply writes the CUST_CODE value from the **frmFindCustomerForPayment** form to the **frmMakePayment** form. The same process writes the CUST_BALANCE value from the **frmFindCustomerForPayment** form to the **frmMakePayment** form. The PAY_DATE is written with the help of the **Date()** function. (Note the last line in the **Make Payment** macro.)

Figure 205 The Macros in the mcrPayments Macro Group

Macro Name	Condition	Action	Comment
SelectCustomer		OpenForm	Open the frmFindCustomerForPayment form
MakePayment		OpenForm	Open the frmMakePayment form
		GoToRecord	Start a new record
		SetValue	Item: [Forms]![frmMakePayment]![CUST_CODE] Expression: [Forms]![frmFindCustomerForPayment]![CUST_CODE]
		SetValue	Item: [Forms]![frmMakePayment]![PAY_OLD_BALANCE] Expression: [Forms]![frmFindCustomerForPayment]![CUST_BALANCE]
		SetValue	Item: [Forms]![frmMakePayment]![PAY_DATE] Expression: Date()
UpdateBalance		SetValue	Item: [Forms]![frmMakePayment]![PAY_BALANCE] Expression: [Forms]![frmMakePayment]![PAY_OLD_BALANCE]-[Forms]![frmMakePayment]![PAY_AMOUNT]
		SetValue	Item: [Forms]![frmFindCustomerForPayment]![CUST_BALANCE] Expression: [Forms]![frmMakePayment]![PAY_BALANCE]
		Close	Close the frmFindCustomerForPayment form
		Close	Close the frmFindCustomerForPayment form
CloseFindCustomerForPaymentForm		Close	
CurrentCustomerBalance		OpenForm	Open the frmShowPayments form

Identify the value to be changed (points to Item and Expression lines)

Source (points to the form name in the Item line)

Copy the Item and Expression information from the Action Arguments section and paste it into the Comment section for quick reference (points to the Comment column)

Object Type: Form
Object Name: frmMakePayment
Record: New

Enter a comment in this column.

The **UpdateBalance** macro shown in Figure 205 takes the PAY_AMOUNT value from the form named **frmMakePayment** form and subtracts it from the PAY_OLD_BALANCE on the **frmMakePayment** form. (Note the first **SetValue** action's **Item:** and **Expression:** lines in the **UpdateBalance** macro.)

Once the macros in the **mcrPayments** macro group have been written, you must attach them to the command buttons you created on the main menu. For example, Figure 206 shows that the **SelectCustomer** macro was attached to the **Payments** command button, using the **On Click** format. After you have attached the macro as shown in Figure 206, open the main menu form and click on the **Payments** command button to generate the parameter entry box shown in Figure 207. Note that the macro worked as intended.

Figure 206 Macro Payment Attachment

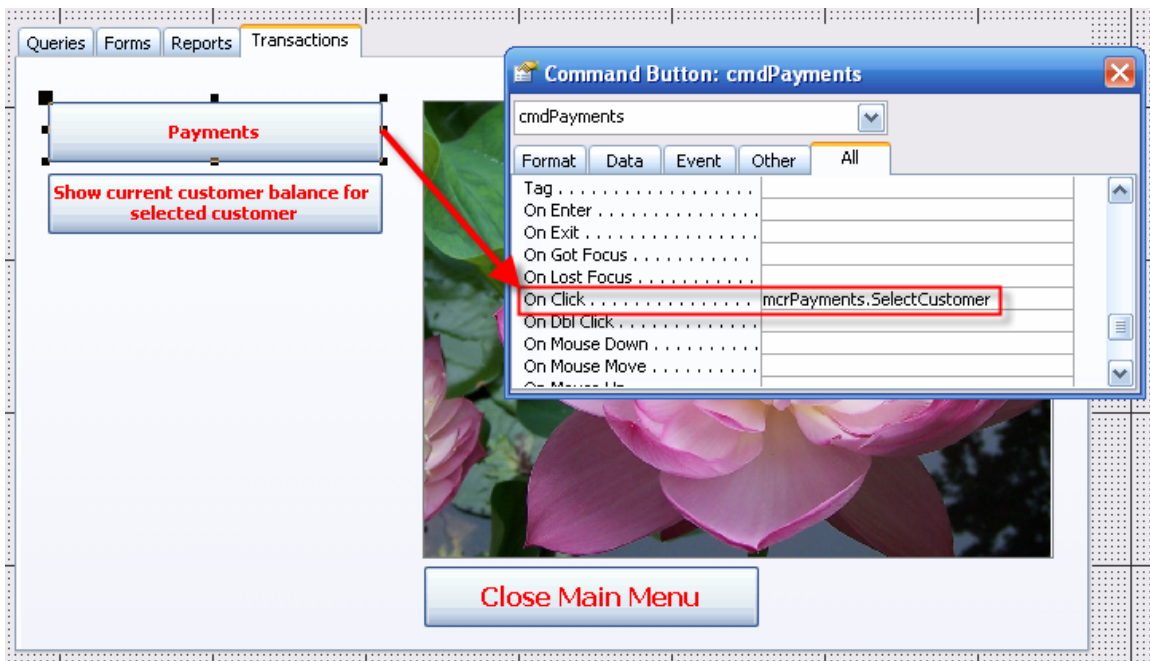
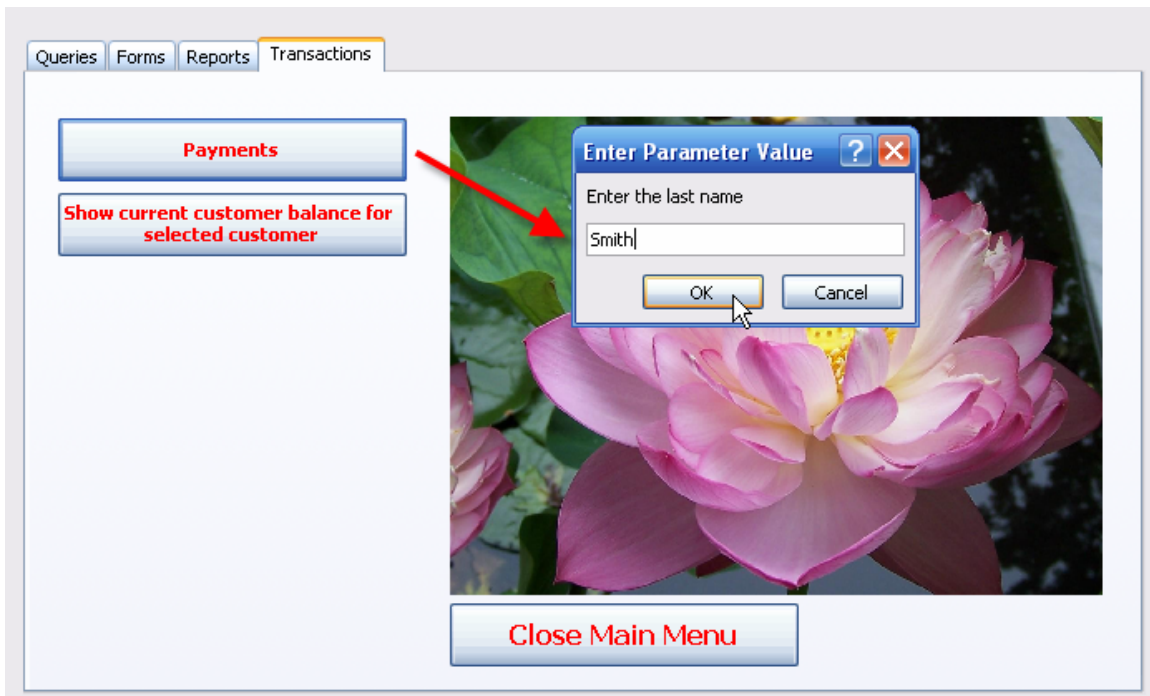
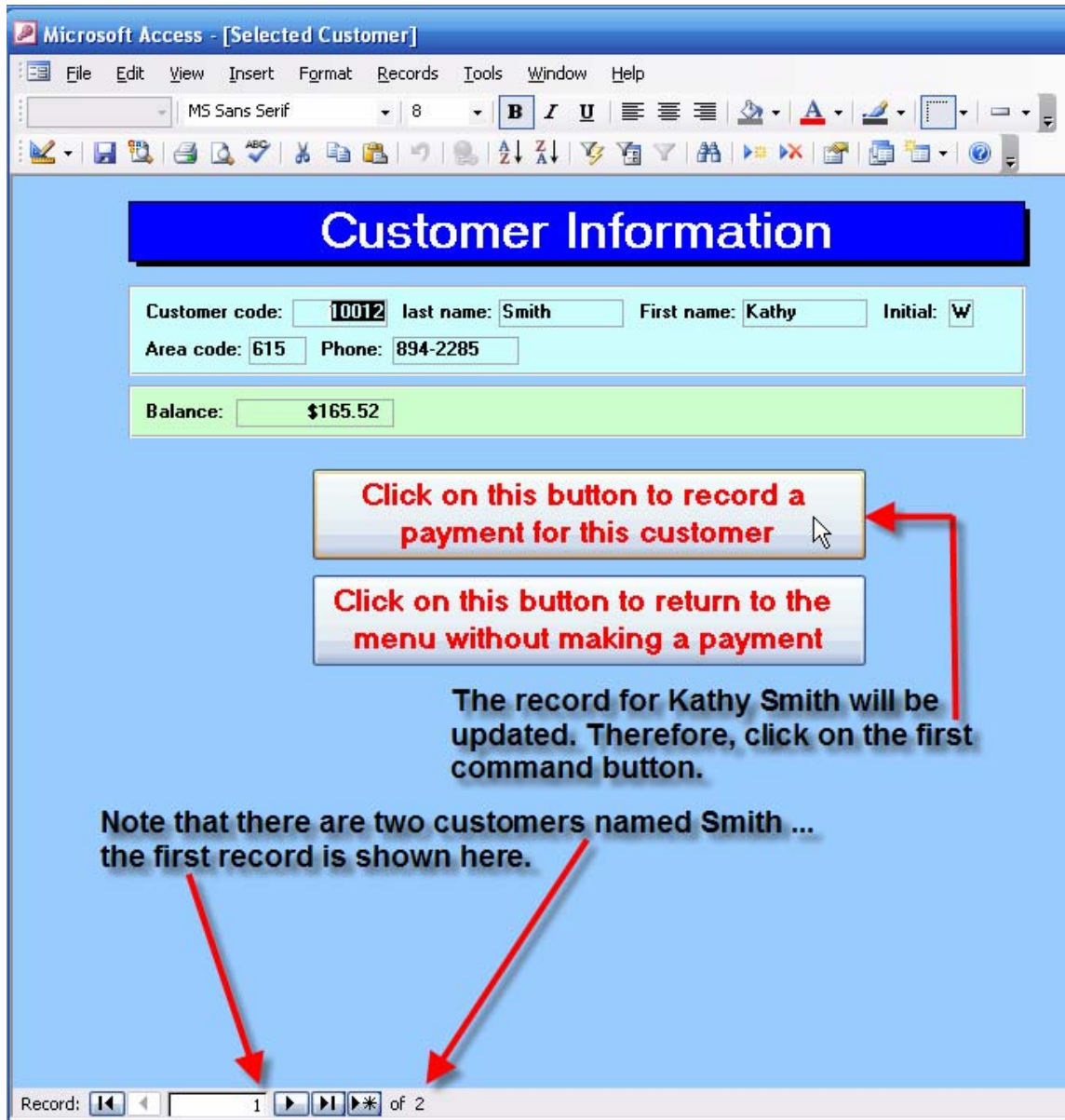


Figure 207 Payment Selection



Making the **Smith** entry shown in Figure 207 will open the form shown in Figure 208. (Note that the form shows two records, because the CUSTOMER table contains two customers whose last name is Smith.) After deciding that this first record is the one you intend to update, click on the first command button to trigger the next event, which is the payment transaction that will record a payment for customer Kathy Smith's account.

Figure 208 Selected Customer



Clicking on the first command button shown in Figure 208 will open the payment form shown in Figure 209. Review the **MakePayment** and **UpdateBalance** macros in Figure 205 to see what actions will be taken. You will discover that the macros will perform their intended tasks ... note that the payment of \$100 will update the original \$411.02 balance to yield the updated payment balance of \$311.02.

Figure 209 Payment Entry Events

Payment Number	Customer Code	Current Payment Balance	Payment Date	Payment Amount	Updated Payment Balance
	10012	\$411.02	12-Mar-06	\$100.00	\$311.02
(Number)	0	\$0.00		\$0.00	\$0.00

Entered automatically by Autnumber

Make a Payment

This value is calculated and entered by the macro as soon as the end user taps the Enter key after making the Payment Amount entry

Current (system) date is entered automatically

The only data entry made by the end user

Copied from the form named frmFindCustomerForPayment

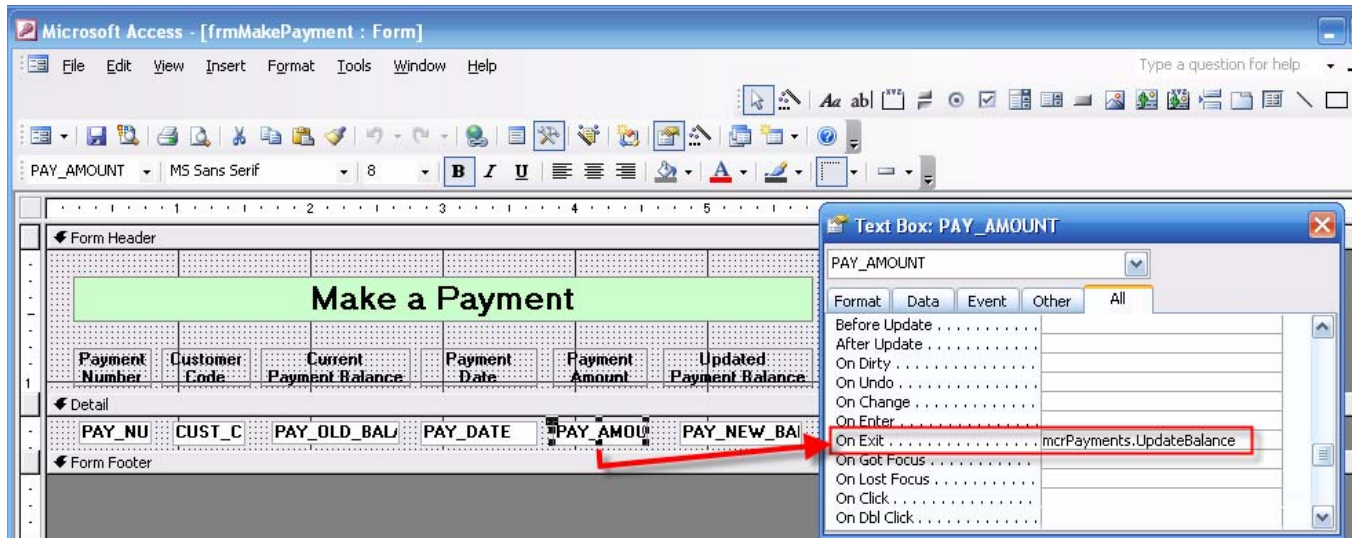
Note

The payment balance in the PAYMENT table is not updated until the form is closed. Therefore, you cannot make “single pass” multiple payments for any selected customer. If you want to make multiple payments for a selected customer, the correct process is to make the first payment, close the form, open the form again, make the next payment, and so on.

This restriction is not a serious limitation -- the “multiple payment in one pass” scenario is not likely in a real world environment. For example, if a customer comes in to make a \$100 payment by check, would it make more sense to write a single \$100 check or to write two checks for \$50 each, or even five checks for \$20 each? It is much more likely that multiple payments are made over some period of time, in which case you would have to close the form after each payment anyway. (And if you redesigned the system to let customers make payments on individual balances for each invoice, you would have to close each payment transaction as you select each of the invoices.)

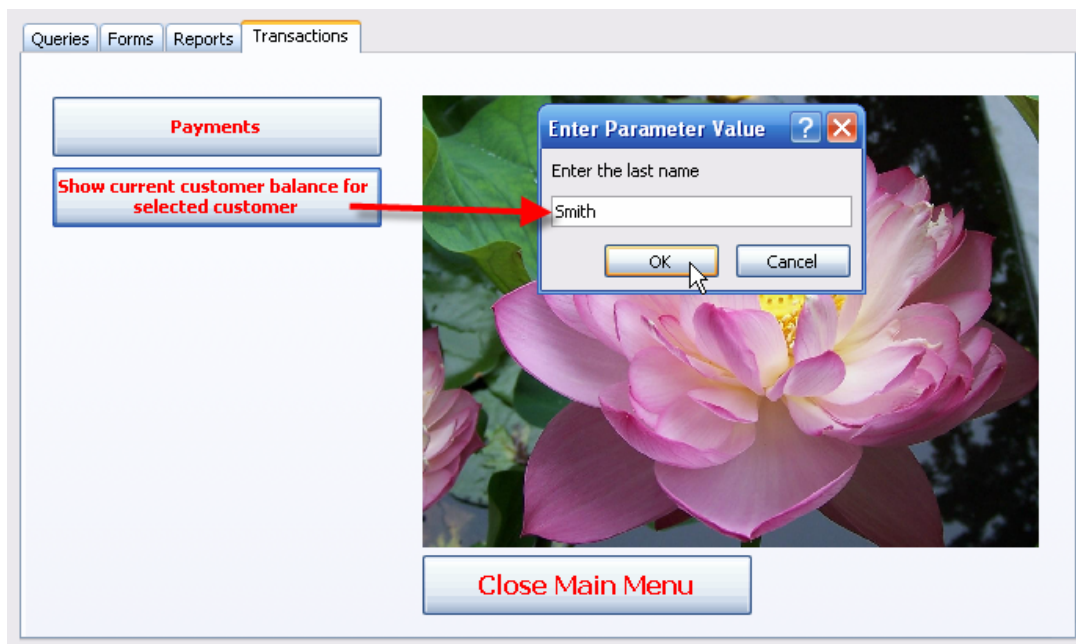
The macro attachment for the **Payment Amount** is shown in Figure 209A. Note that the update is to be made **On Exit**, because you cannot update the balance until after you have made the payment amount entry.

Figure 209A Balance Update Macro Attachment



To track the payment history for any customer, use the second command button on the main menu form shown in Figure 210. The second command button shown here uses the **CurrentCustomerBalance** macro – see Figure 205 -- to ensure that the selected customer is found and that this customer’s payment record will be shown.

Figure 210 Selection of Current Balance Option



Note that the last name entry **Smith** shown in Figure 210 yields the results shown in Figure 211. Because Kathy Smith is the only customer for whom a payment entry was made thus far, only one record in the PAYMENT table matches the **Smith** last name entry.

Figure 211 Updated Customer Payment Balance

Customer Payment Information

Customer code: last name: First name: Initial:
Area code: Phone:

Balance:

Payment Subform

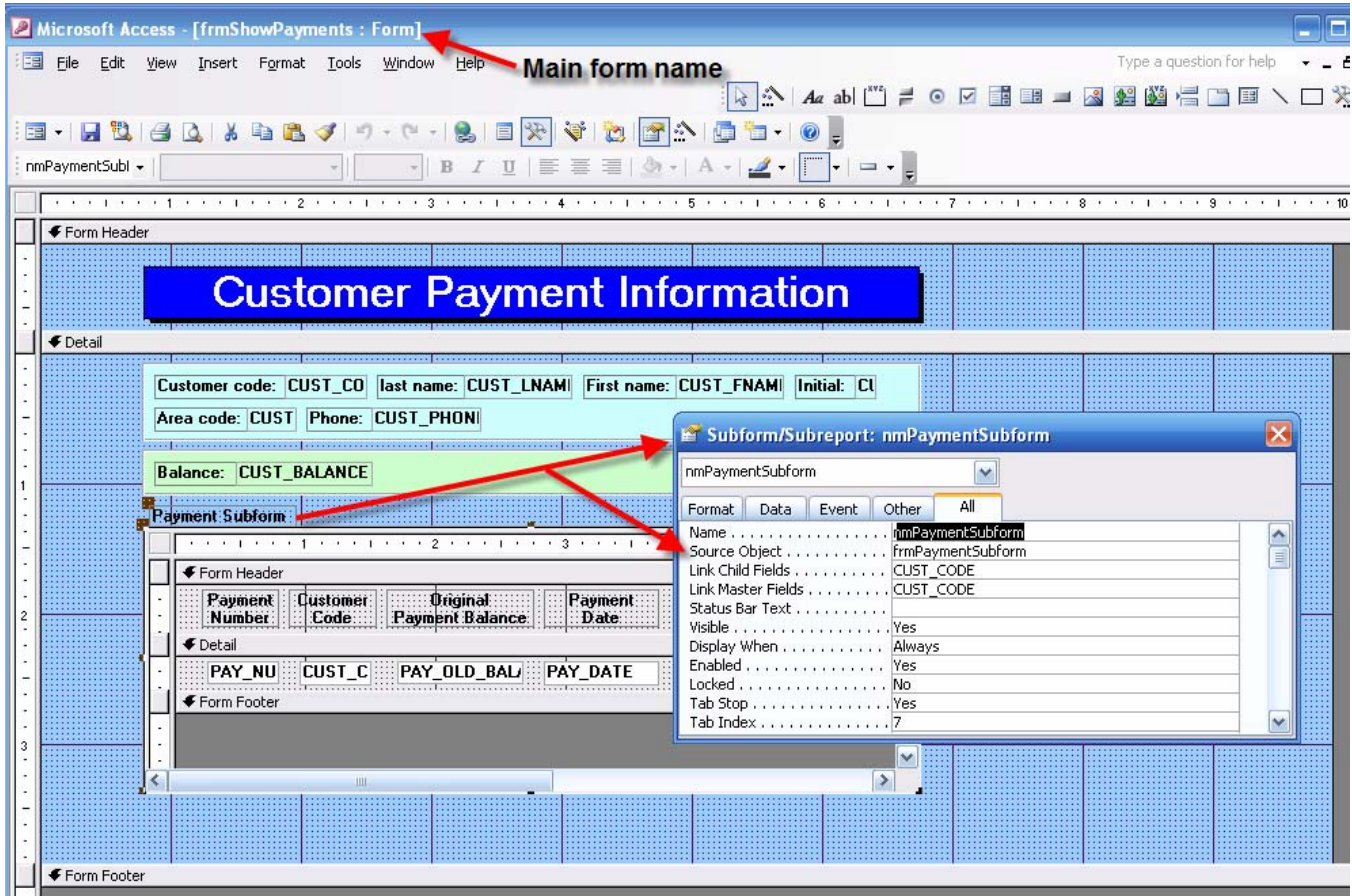
Payment Number	Customer Code	Original Payment Balance	Payment Date	Payment Amount	Updated Payment Balance
<input type="text" value="1"/>	<input type="text" value="10012"/>	<input type="text" value="\$411.02"/>	<input type="text" value="12-Mar-06"/>	<input type="text" value="\$100.00"/>	<input type="text" value="\$311.02"/>
<input type="text" value="(Number)"/>	<input type="text" value="10012"/>	<input type="text" value="\$0.00"/>	<input type="text" value=""/>	<input type="text" value="\$0.00"/>	<input type="text" value="\$0.00"/>

Note that the payment updated the original balance and that the updated balance is written in the payment and customer sources

Record: of 1

As you examine Figure 211, note that a form/subform structure was used to produce a detailed view of all the relevant information. The main form is basically a copy of the original customer form and the subform is basically a copy of the payments form you created earlier. The form/subform structure is shown in Figure 212. If you have used a copy of the payment form shown in Figure 209 that still includes the update macro, you can make payments via this form/subform, too.

Figure 212 Customer Payments



Go ahead and make a few more payment on account entries for Kathy Smith. A sample set of several payment entries is shown in Figure 213.

Figure 213 Payment History

Customer Payment Information

Customer code: last name: First name: Initial:
Area code: Phone:

Balance:

Payment Subform

Payment Number	Customer Code	Original Payment Balance	Payment Date	Payment Amount	Updated Payment Balance
<input type="text" value="1"/>	<input type="text" value="10012"/>	<input type="text" value="\$411.02"/>	<input type="text" value="12-Mar-06"/>	<input type="text" value="\$100.00"/>	<input type="text" value="\$311.02"/>
<input type="text" value="2"/>	<input type="text" value="10012"/>	<input type="text" value="\$311.02"/>	<input type="text" value="20-Mar-06"/>	<input type="text" value="\$125.00"/>	<input type="text" value="\$186.02"/>
<input type="text" value="3"/>	<input type="text" value="10012"/>	<input type="text" value="\$186.02"/>	<input type="text" value="25-Mar-06"/>	<input type="text" value="\$20.50"/>	<input type="text" value="\$165.52"/>
<input type="text" value="Number)"/>	<input type="text" value="10012"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

Record: of 3

Kathey Smith has made three payments

Kathy Smith is one of two customers whose last name is Smith.

Record: of 2

Conclusion

Only a few examples are shown in this tutorial. The objective is not to develop full-blown applications, but to show you some examples of what can be done in the Microsoft Access environment. Once you have seen those examples, you have a foundation on which to build greater expertise. Keep in mind that Access is a superb prototyping tool, but it is not capable of serving the full database and information needs of even medium-sized organizations, let alone large ones. Products such as Microsoft's SQL Server, IBM's DB2, or Oracle are better candidates for such environments. Nevertheless, given its ability to let you develop superb prototypes, Access has earned a place of honor in the ranks of database professionals.