

8.1 Introduction: Using existing data

There are a number of different ways to create and populate a table:

- Create the table definition from scratch and then populate the table manually with data values (as you did in [Lesson 5](#)).
- Create the table definition from scratch and then *append* data from some other electronic format to the table.
- *Import* the table definition and data from another database or application (such as MICROSOFT EXCEL or a text file).
- Create a link to an external data source. In this case, the data is not actually stored in your database file, but is accessible from within ACCESS like any other table.

The first approach (populating the tables from scratch) can be seen as a last resort. The data that you need to build an application often exists in electronic format somewhere already. For example, if you are “downsizing” an application from a mainframe environment to a desktop DBMS, the mainframe data can probably be extracted and saved as plain text files.¹ Similarly, if you are upsizing from a

chaotic spreadsheet-based system, then it may be possible to import the spreadsheets directly into the database. Obviously, getting the data in electronic format can save an enormous amount of time and avoid many keyboarding errors.



Of course, there is one important caveat: Mainframe reports and spreadsheets are seldom in normalized form. Consequently, conversion into sensible database structure may involve a certain amount of manual reorganization and transformation.

In this lesson, you will explore techniques for importing data and linking to existing data sources. The general objective of introducing these skills is to permit you to take advantage of existing sources of electronic data whenever possible.

¹ To get the data in the format you require in a mainframe environment, you often have to be very nice to the individual who controls the central computer (i.e., the geek in the glass room). This in itself is often sufficient justification for downsizing.



8.2 Learning objectives

- append data from a spreadsheet to an existing table
- import data from text file into a new table
- create a linked table using data from a non-ACCESS database

8.3 Exercises

8.3.1 Appending data from a spreadsheet

In the tutorial package, you will find a MICROSOFT EXCEL spreadsheet named **CustList.xls**. The spreadsheet contains a list of your customers.¹

- ➔ Open the **CustList.xls** spreadsheet in EXCEL.
- ➔ Compare the columns of data in the spreadsheet to the field names you used in [Section 5.3.1](#) when creating the **Customers** table (i.e., open your **Customers** table in design view).

¹ For the purpose of these lessons, we are working with a very small amount of data. For example, your entire customer base is assumed to consist of five firms. Rest assured that the theory and techniques described in the lessons are independent of the amount of data you are dealing with (within reason).

8.3.1.1 Spreadsheet preparation

Notice that the spreadsheet headings are *similar* to the field names of the **Customers** table. There are, however, two exceptions:

1. The spreadsheet does not contain any information about regions or whether the customer has been authorized to conduct on-line ordering. In contrast, the table has a **RegionCode** field and an **OnLineOrdering** field.
2. The spreadsheet contains a column named “Billing Address” (with a space between the words) whereas the database field is called “BillingAddress” (no space between the words in accordance with the field naming conventions [introduced](#) in [Section 5.4.5.1](#)).

To append records from a spreadsheet into an existing table, the first row of the spreadsheet must contain column names that match the field names of the target table exactly. The import wizard uses the column name information to ensure the data ends up in the right place.

- ➔ Edit the spreadsheet so that the heading “Billing Address” reads “BillingAddress” (one word) as shown in [Figure 8.1](#).



Since each column of data in the spreadsheet is labeled, there is no

requirement to have “dummy columns” for missing fields. For example, there is no need to insert a blank column in the spreadsheet and label it “RegionCode”.

➔ Save and close the modified spreadsheet.

8.3.1.2 Using the import wizard

- ➔ Return to ACCESS, ensure the database window is in the foreground, and select **File** → **Get External Data** → **Import** from the main menu.
- ➔ In the “Open” dialog, select the **CustList.xls** spreadsheet as the source of

FIGURE 8.1: Modify the contents of the spreadsheet that contains the customer data.

1 Open the CustList.xls file in EXCEL.

2 Edit the column name “Billing Address” so that it conforms to the naming convention you used when creating the **Customers** table in ACCESS.

3 Save and close the spreadsheet.

1	CustName	Billing Address	City	Prov	PostalCode	ContactPerson	ContactPhone
2	Sam's Stock Pot	8272 West 4th Avenue	Vancouver	BC	V7H 1K8	Sam Wong	(604) 732-1019
3	Loonie Mart #107	1929 Commer Drive	Vancouver	BC	V9G 1K0	Bill Williams	(604) 233-9101
4	Rosch Dry Goods Inc.	82 Crowfoot Trail	Calgary	AB	T2W 1J4	Alice McPorie	(403) 229-4483
5	Gadgets "R" Us	Suite 235 1523 10th Avenue	North Vancouver	BC	V9L 1U9	Leslie Cranfield-Jones	(604) 929-2829
6	The Chef's Assistant	1832 Martin Street East	Kamloops	BC	V2A 1K3	Andre Oulette	(604) 490-2928
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							



the data to be imported, as shown in [Figure 8.2](#).

- ➔ Follow the instructions provided by the wizard to complete the import process, as shown in [Figure 8.3](#) and [Figure 8.4](#).

You should get a message reporting that the import process was successful.



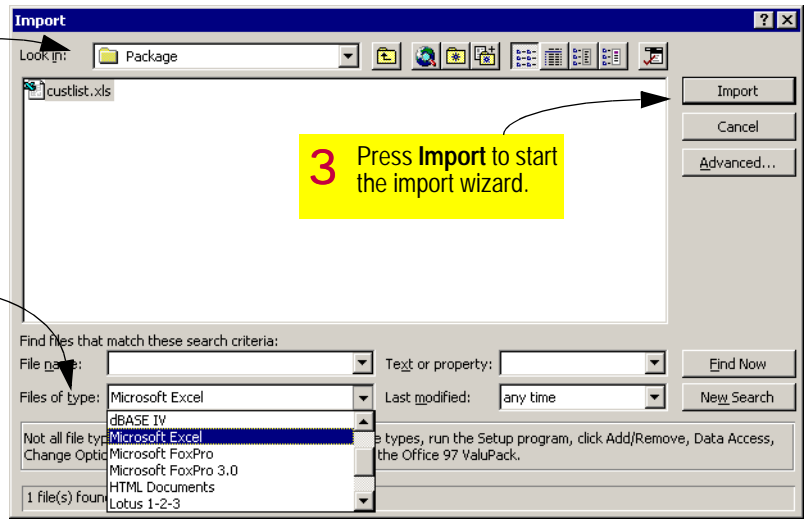
If there is a problem with the import wizard, the error message provided by ACCESS is not particularly helpful. All you can do is insure the column headings in your spreadsheet match the field names in the target table and try again.

- ➔ Open your **Customers** table in datasheet view to verify that the data has been transferred correctly.

FIGURE 8.2: Locate and open the EXCEL workbook containing the customer data you wish to import.

1 Navigate to the folder that contains the **CustList.xls** spreadsheet.

2 Set the file type to MICROSOFT EXCEL. All files of the specified type are displayed.



3 Press **Import** to start the import wizard.

By importing the data from the spreadsheet, you have created a completely independent copy of the data. Thus, if a customer address is

changed in the spreadsheet, it must also be changed in the **Customers** table. For this reason, importing works best when the

FIGURE 8.4: Use the import wizard to append records to the **Customers** table (part 2).

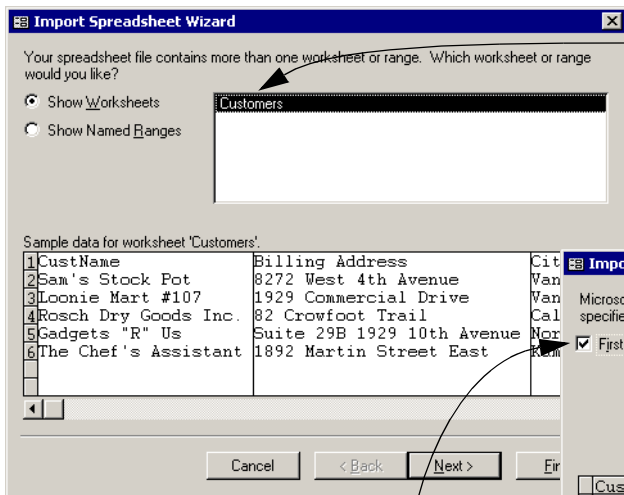
3 Indicate that the data is to be imported into an existing table (**Customers**).

4 Leave the name of the target table unchanged.

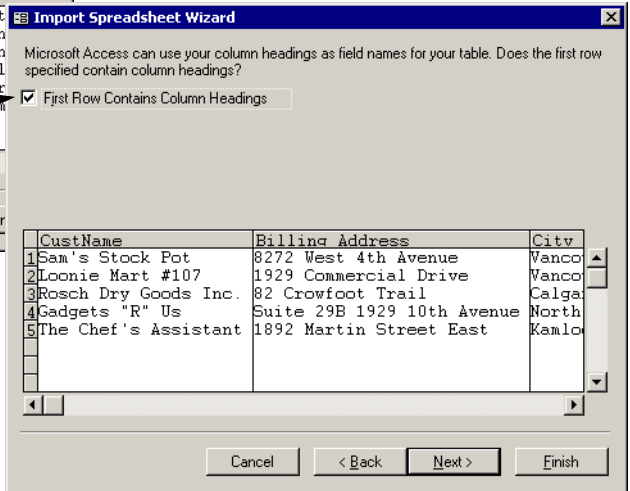
5 Click on **Finish** to start the import process.

CustName	Billing Address	City
1 Sam's Stock Pot	8272 West 4th Avenue	Van
2 Loonie Mart #107	1929 Commercial Drive	Van
3 Rosch Dry Goods Inc.	82 Crowfoot Trail	Cal
4 Gadgets "R" Us	Suite 29B 1929 10th Avenue	Nor
5 The Chef's Assistant	1892 Martin Street East	Kan

FIGURE 8.3: Use the import wizard to append records to the *Customers* table (part 1).



1 Indicate that the data you want to import is on a worksheet named "Customers".



2 Check the box to indicate that the first row in the spreadsheet contains column names, not data.

database application is meant to *replace* the spreadsheet-based application, rather than

coexist with it.

8.3.2 Importing data from a text file

In this section, we are going to assume that you have been provided with an ASCII text file called **Inventor.txt** that contains information about the inventory level of all your products. ASCII is an acronym for the American Standards Code for Information Exchange. Briefly, ASCII is a standard for converting patterns of bits and bytes into platform-independent, human-readable text. ASCII (or “plain text”) files typically have extensions such as “.txt”, “.asc”, or “.csv”.



If in doubt, an easy way to determine whether a file is in ASCII format is to open it using WINDOWS NOTEPAD. If you can read the contents of the file, then it is ASCII.¹ If you see a bunch of special characters, blocks, and smily faces, then the file is in one of many non-ASCII—or “binary”—formats (such as spreadsheet or a graphics file formats).

¹ More recent versions of NOTEPAD can also read Unicode text. Unicode was developed in response to the inadequacy of ASCII for representing languages other than English. Whereas ASCII uses seven bits to represent up to 128 different characters, Unicode currently uses 16 bits and supports 24 different languages.

8.3.2.1 Exploring the ASCII text file

- Open **Inventor.txt** in the NOTEPAD text editor that comes with WINDOWS. On most WINDOWS systems, you can start NOTEPAD using **Start** → **Programs** → **Accessories** → **Notepad**.
- Examine the format of the data. The key elements of this particular text file are shown in [Figure 8.5](#).
- Close the text file.

Although the name of the data file is **Inventor.txt**, the file contains information about products generally. Inventory level (**QtyOnHand**) is simply one of several product attributes, such as description, price, and so on.



When you are building systems, do not be fooled by the naming conventions of others. For example, in a banking environments, information about clients (name, address, etc.) can often be found in a data source called “accounts”. In many cases, some detective work is required to make sense of the data you already have.



FIGURE 8.5: Examine the format of the *Inventor.txt* ASCII file.

An ASCII text file contains plain text. In the *Inventor.txt* file, each record is on a separate line and fields are delimiters by commas. Textual values are enclosed within quotation marks.

! Although nested quotation marks often cause import programs to work incorrectly, the ACCESS import wizard appears to handle such issues very well. If the nested quotation marks do cause problems, you have to edit the text file (using search and replace) and re-run the import wizard.

```

ProductID,"Description","Unit","UnitPrice","QtyOnHand"
51 5012,"water jug, s.s. w/ice guard, 2 litre","EA",23.50,36
57 3826,"spatula, 6"" Cuisipro","EA",4.00,65
57 3828,"spatula, 8"" Cuisipro","EA",4.25,20
57 4966,"Mixing bowl, 16 qt.,"EA",12.50,0
57 551,"S.S. salad server set","2PC",3.15,32
71 12101,"S.S. soup ladle","EA",5.25,49
71 12110,"S.S. skimmer","EA",5.00,20
71 12111,"S.S. sauce ladle","EA",5.25,9
71 12114,"S.S. grave ladle with spout","EA",4.75,56
74 4042,"snail plate w/white handle","EA",3.15,36
74 4321,"Pastry brush, 1""","EA",4.00,32
74 4539,"Meat tenderizing hammer","EA",2.50,12
74 6083,"Spring form pan, 9"" non stick","EA",7.50,8
74 6084,"Spring form pan, 10"" non stick","EA",8.00,18
74 6102,"Deluxe measuring spoon set","4PC",3.50,11
74 6109,"Colander, s.s., 5 qt.,"EA",6.00,21
74 6191,"Potato ricer, tinned","EA",8.50,0
74 6245,"Pastry blender","EA",3.25,19
74 6308,"wok 14"" steel with handles","EA",10.00,78
74 6811,"Med. rubber scraper w/s.s. handle","EA",2.65,56
74 6814,"Rubber scraper 9-1/2"" x 2""","EA",0.80,32
74 6881,"Lobster set","EA",22.00,11
74 7102,"Cuisipro supreme pepper 5 oz.,"EA",4.95,8
78 6932,"Fondue fuel, 500 ml","EA",1.75,26
    
```

8.3.2.2 Importing a new table

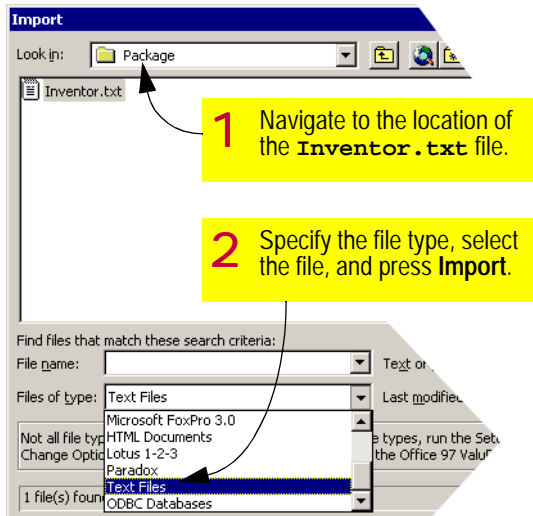
Rather than append the data to an existing table as you did in Section 8.3.1, you are going to import the text file into a completely new table.

➡ Ensure the database window is in the foreground and select **File** → **Get External Data** → **Import** from the main menu.

- ➡ In the "Open" dialog box, specify the location of the *Inventor.txt* file, as shown in Figure 8.6.
- ➡ Follow the instructions provided by the import spreadsheet wizard as shown in Figure 8.7 and Figure 8.8. Save the resulting table as new table named **Products**.
- ➡ Open the **Products** table to verify that the data was imported correctly.



FIGURE 8.6: Locate and open the text file containing the product data.



wizard, you need to open the table in design mode and change many of the design decisions made by the import wizard.

For example, the import wizard sets the length of all text fields to the maximum allowable value (255 characters). In addition, the field names are taken directly from the heading row in the text file and no captions or defaults are specified.



Once you have created a table by importing data, you must remember to switch to design mode and change the field properties (e.g., data type, size, caption, and default) to appropriate values.

8.3.3 Creating a link to a different database

Recall that your employee data is stored in an off-the-shelf payroll system that you bought many years ago. Despite its age, the payroll system still does what it is supposed to do and you have no intention of replacing it or upgrading it. After asking around a bit, you discover that the data is actually saved in dBASE IV format for DOS.

Although you could use the import wizard to *import* the employee data, doing so would create an independent copy of the employee data. Thus, every change made to employee

It might seem like less work to import a new table rather than append records to an existing table. To append, you have to first create the table structure and then edit the source file to ensure that its headings conform to the table structure.

It turns out, however, that both approaches require about the same amount of effort. Once you have imported a table using the import

FIGURE 8.7: Use the import wizard to create a *Products* table (part 1).

The wizard has decided that your data is in a 'Delimited' format. If it isn't, choose the format that more correctly describes your data.

Choose the format that best describes your data:

- Delimited - Characters such as comma or tab separate each field
- Fixed Width - Fields are aligned in columns with spaces between each field

Sample data from file: \\CELLAR\WORK\TUTORIALS\2NP\ORDERENTRY_97\FILES\SA1A

1	"ProductID", "Description", "Unit", "UnitPrice", "Qty
2	"51 5012", "Water jug, s.s. w/ice guard, 2 litre"
3	"57 3826", "Spatula, 6" "Cuisipro"", "EA", \$4.00,
4	"57 3828", "Spatula, 8" " "Cuisipro"", "EA", \$4.25,
5	"57 4966", "Mixing bowl, 16 qt.", "EA", \$12.50,0
6	"57 551", "S.S. salad server set", "2PC", \$3.15,32
7	"71 12101", "S.S. soup ladle", "EA", \$5.25,49
8	"71 12110", "S.S. skimmer", "EA", \$5.00,20

What delimiter separates your fields? Select the appropriate delimiter and see how your text is affected in the preview below.

Choose the delimiter that separates your fields:

- Tab
- Semicolon
- Comma
- Space
- Other:

First Row Contains Field Names

Text Qualifier:

ProductID	Description
51 5012	Water jug, s.s. w/ice guard, 2 litre
57 3826	Spatula, 6" Cuisipro"
57 3828	Spatula, 8" Cuisipro"
57 4966	Mixing bowl, 16 qt.
57 551	S.S. salad server set
71 12101	S.S. soup ladle
71 12110	S.S. skimmer
71 12111	S.S. sauce ladle

1 Since the fields in the text file are delimited (with quotations marks and commas), select the "Delimited" option.

2 Indicate that each value is separated by a comma.

3 As before, check to indicate that the first line in the file contains field names, not data.

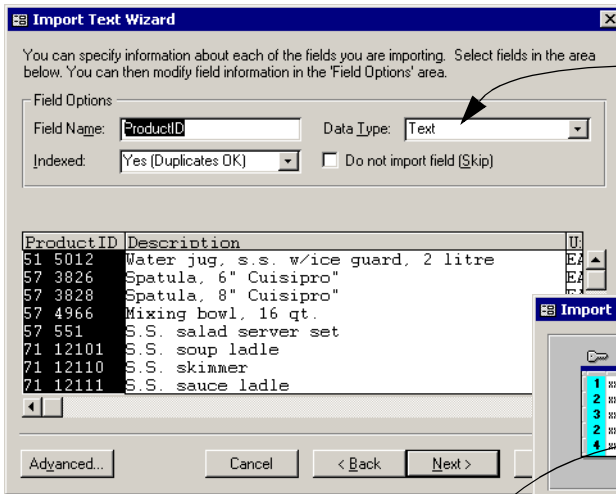
4 Indicate that textual values are denoted by double quotation marks.

information in the payroll system would also have to be made to the **Employees** table in the

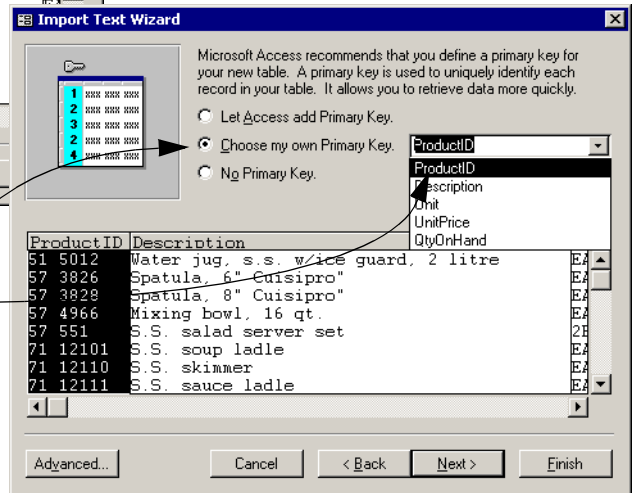
order entry system.



FIGURE 8.8: Use the import wizard to create a **Products** table (part 2).



5 In this screen, you specify field names and data types (or skip the field entirely). Since it is easier to make these changes from with the table design view, simply press Next to move to the next screen.



6 Use the existing **ProductID** field as the primary key for the new **Products** table.

7 In the final screen, press **Finish** to import the table.



Murphy's Law of Information Systems states that if two copies of the same data are stored electronically, they will become inconsistent almost immediately.¹

ACCESS has a feature called **linked tables** that permits you use a table from different databases as if it were part of your own database. Moreover, the source table does not necessarily have to be located in an ACCESS database file. ACCESS provides direct support for many popular desktop databases (such as dBASE, FOXPRO, and PARADOX) and virtually any data source can be linked through ODBC middleware (ODBC is described in much greater detail in [Lesson 9](#)).

In this section, you are going to create a link to the table used to store employee data in the payroll system. In dBASE IV, each database object (table, query, form, and even index) is stored in a separate file. Thus, the **PAY_EMPS.dbf** file contains a single table. The index file for the table, which is used to identify the primary key and speed-up searches, is named **PAY_EMPS.mdx**. Both these files can be found in the [project package](#).



It does not take long working with separate files for tables, indexes, and other database objects before you come to appreciate the single-file approach used by ACCESS.

- From the database window, select **File** → **Get External Data** from the main menu. Instead of selecting **Import** as you have done previously, select **Link Tables**.
- In the "Link" dialog, find the **PAY_EMPS.dbf** file included in the project package. The file type should be set to dBASE IV.
- Select the file and press the **Link** button, as shown on the left-hand side of [Figure 8.9](#).

If you look carefully, the name of the "Link" dialog has changed to "Select Index File" and the file type is "dBASE Index".

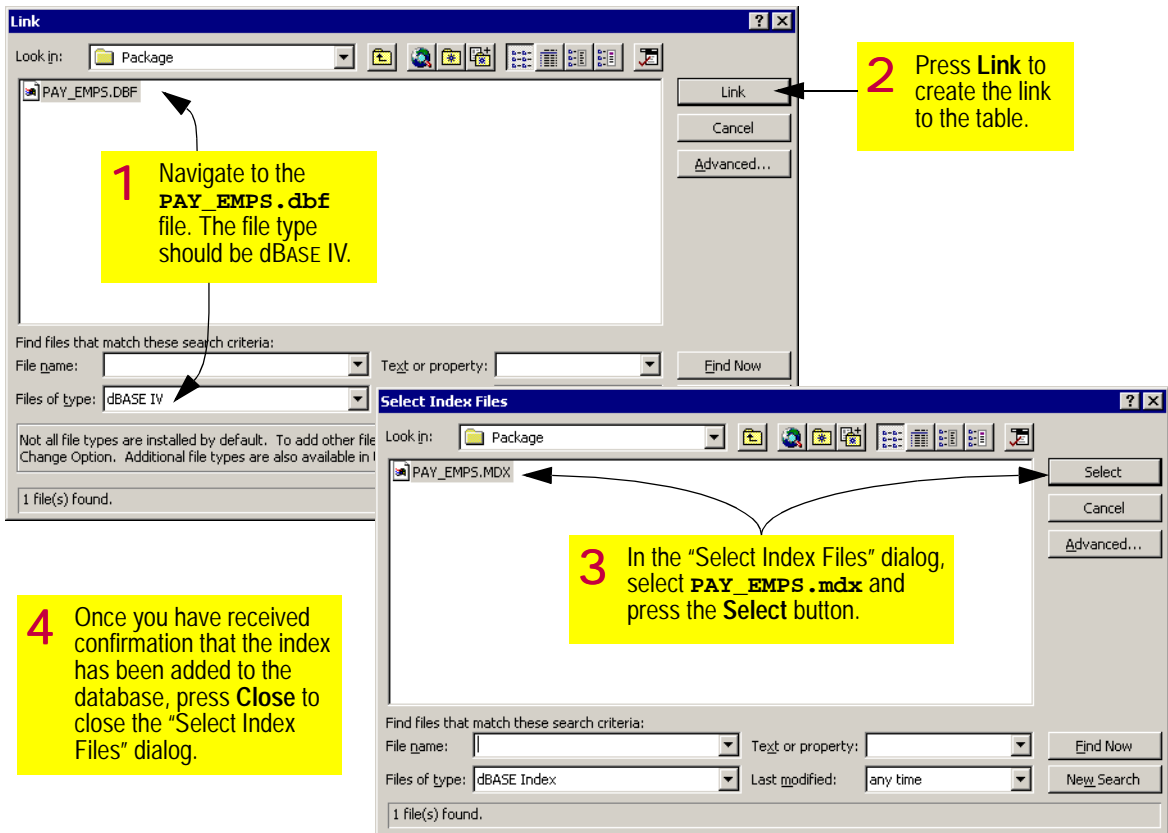


If you are using ACCESS 2000, you will not get the "Select Index File" dialog. In addition, ACCESS 2000 does not permit you to modify the table once the link to the dBASE table is created. The problem is that MICROSOFT has changed the way in which ACCESS 2000 interacts with dBASE files. To fix the problem, you can either visit the Microsoft site and download an updated copy of the JET 3.5 database

¹ This is not a *real* Murphy's Law—I just made it up.



FIGURE 8.9: Create a link to the dBASE IV file **PAY_EMPS.dbf** and its index file **PAY_EMPS.mdx**.



engine (start by searching the MICROSOFT site for knowledge base article

"Q263561"). Alternatively, you can simply ignore the problem—we will not be updating the **PAY_EMPS** table anyway.

- ➔ Ensure **PAY_EMPS.mdx** is highlighted and press the Select button, as shown on the right-hand side of Figure 8.9. You should get a message indicating that the index has been added.

Once you have selected the index, the "Select Index File" dialog does not go away. This is because it is possible to have multiple indexes per table in ACCESS.

- ➔ Press the Close button without adding any further index files.

You should now get a dialog asking you to select a unique record identifier as shown in Figure 8.10. Since only one dBASE index was specified, and since the primary key specified in the index is the field **EMP_ID**, you do not have much of a choice to make.

- ➔ Select **OK** to **EMP_ID** as the unique record identifier (key). You should get a message that the link was made successfully.
- ➔ Press **Close** to close the "Link" dialog box.

You should now have a linked table called **PAY_EMPS** in the database window, as shown in

Figure 8.11. Since the name "PAY_EMPS" does not conform to our table naming convention, you should change it to something else.

- ➔ Right-click on the **PAY_EMPS** linked table in the database window, select **Rename** from the context menu, and change the name of the link to **Employees**.
- ➔ Open the **Employees** table in datasheet mode.

Note that the linked table is simply a window into the **PAY_EMPS.dbf** file. Thus, if you make any changes to the data (e.g., change Gerard Huff's first name to Gerry), the change is made directly to the payroll system's data. Of course, there is no payroll system in this case, so there

FIGURE 8.10: Select EMP_ID as the unique record identifier for the linked table.

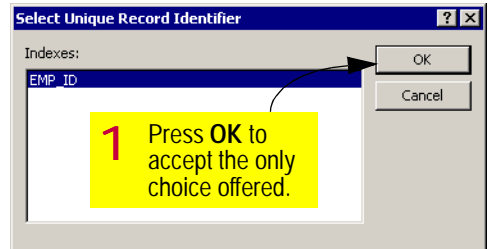
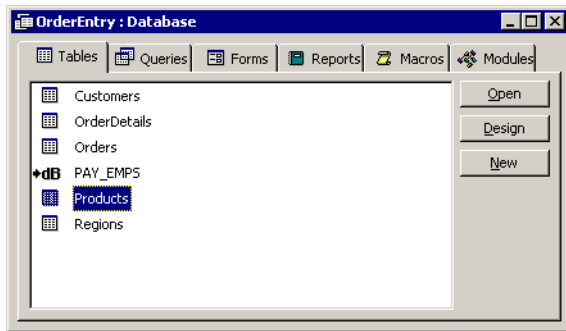




FIGURE 8.11: The database window shows a linked table named PAY_EMPS.



is very little stopping you from experimenting with the linked data.

In the context in which the employee data is used here, it is actually very important that the data from the source table be read-only and sensitive data should be excluded. The last thing you want is for an order entry clerk to accidentally change job classifications or pay levels in the payroll system.



You must be mindful of security issues when creating links between databases. In this example, anyone with access to the order entry system can now open the **Employees** linked table and look at and

change the **EMP_PAY_LE** (pay levels) field for all the employees in the company.

8.3.4 Changing the foreign key

In [Section 6.3.3](#), we made a provisional commitment to use a concatenated key consisting of **emp_fname + emp_lname** to uniquely identify employees.

Now that we have created a link to the payroll system and have had the opportunity to see the fields and data types used in the system, we can make a more informed decision about what field to use as a foreign key. Specifically, we know that a field called **EMP_ID** exists and can be used to uniquely identify employees in the organization.




Remember that ACCESS, like most databases, ignores the case of field names and other database objects. Thus, it does not matter that you named your fields using lower case whereas the fields in the dBASE IV file are named using upper case.

8.3.4.1 Concatenated foreign keys revisited

Before we change the foreign key, we are going to take the opportunity to revisit the topic of declaring relationships for concatenated keys.

- ➔ Ensure that all tables are closed. Then open the relationships window.
- ➔ Use the **Show Table** button add the new Employees table to the relationship window.
- ➔ Multi-select the foreign key on the “one” side (that is, hold down the **Control** key and click on **EMP_FNAME** and **EMP_LNAME** in the **Employees** table).
- ➔ Drag the fields on to the “many” side (the **Regions** table).
- ➔ In the “Edit Relationships” dialog, make sure the correct fields in both tables are selected, as shown in [Figure 8.12](#).

 Although ACCESS selects the correct fields when a single-field foreign key is used, it does not do a very good job of selecting fields when multiple fields are used. As a consequence, you must complete the step shown in [Figure 8.12](#) manually.

You will notice that the check boxes for specifying referential integrity and cascading deletes are disabled.


 Since Access is unable to automatically enforce referential integrity for linked

FIGURE 8.12: Ensure the correct fields are selected for the relationship.



tables, you have to write your own routines in a programming language or use other techniques (discussed in later lessons) to achieve this functionality.

8.3.4.2 Using EMP_ID as a foreign key

Changing foreign keys in a fully-implemented database application is much like changing the plumbing in an old house—it is hard, messy work and if all goes well, no one can tell the difference. On the other hand, if things do not go well...

- ➔ Right click the relationship you created in [Section 8.3.4.1](#) and select **Delete**.

Since **EMP_FNAME** and **EMP_LNAME** are no longer going to be used as the foreign key, they no longer belong in the **Regions** table.

- ➔ Switch to the database window and open the **Regions** table in design mode.
- ➔ Select **EMP_FNAME** and **EMP_LNAME** (use the field selectors to the left of the field names) and press delete. Ignore any warning messages that appear about deleting indexes.
- ➔ Create a new field called **RepID** and set its data type to Long Integer. The field will contain the **EMP_ID** of the employee representing the region.
- ❓ The pros and cons of “RepID” as a field name are discussed in [Section 8.4](#).
- ➔ Open the relationships window and create a relationship between **Employees.EMP_ID** and **Regions.RepID**.

8.4 Discussion: Naming consistency across multiple systems

In the exercises above, you created a field called **RepID** to store the employee number of the individual responsible for each sales region. In the human resources database, however,

employee numbers are stored in a field called **EMP_ID**.

You can create a relationship between the **Regions.RepID** and **Employees.EMP_ID** fields as long as the two have exactly the same data type (in this case, Long Integer). That is, fields in a relationship do not need to have the same name. The downside of using different field names is that it is not obvious that the two fields contain the same data or are related in any way. On the positive side, the name **RepID** makes it clear that the field contains the ID of the sales representative for the region.

As a general rule of thumb, it is best to use the same name for the same piece of data. But in large organizations (or even in small organizations in which application development is decentralized), it is very hard to get people to converge on a single naming scheme without making an large invest in an organization-wide data model

- ❓ Many CASE tools have a repository for attribute alias. For example, the fact that **RepID** is an alias for **EMP_ID** could be stored in the CASE tool for future reference.

8.5 Application to the project

Although you have imported customer data, recall that the spreadsheet does not contain any information about the regions to which customers have been assigned.

- ➔ Add the following information to the **Customers** table:

Customer name	Region code
Sam's Stock Pot	C
Loonie Mart #107	K
Rosch Dry Goods Inc.	E
Gadgets "R" Us	C
The Chef's Assistant	N

The situation is similar for the **Regions** table: although you have created the **RepID** field to store the **EMP_ID** values of each region's sales rep, you have not yet populated the **Regions.RepID** column.

- ➔ Add the following information to the **Regions** table:

Region	EMP_ID of sales rep
Central	9
East	3
Key	9
North	4

Region	EMP_ID of sales rep
South	NULL
West	10



Note that there is currently no sales rep assigned to the South region. To record this fact in the **Regions** table, simply leave **RepID** blank for the South region (i.e., do not try to type N-U-L-L into the field).

- ➔ Ensure you have modified the field properties of the tables you have imported. For example, the names, data types, and lengths of the fields in the **Products** table need to be changed before relationships are created.



An input mask is not recommended for **ProductID** for two reasons. First, the field is only partially structured so there is little you can do to constrain values. Second, as you will see later on, it is better to have users pick valid **ProductIDs** values from a list.



If you ignore the recommendation above and decide to create an input mask for the **ProductID** field, ensure you understand the implications of [Section 5.4.6.2](#).



At this point, all your **master tables** should be populated. Do not worry about populating your **transaction tables** yet—this is what we are building the order entry system for.



For the purpose of the assignment, the term “transaction” tables refers to tables that contain information about individual transactions (e.g., **Orders**, **OrderDetails**). “Master” tables, in contrast, are tables that contain relatively stable, non-transactional information (e.g., **Customers**, **Products**).