Chapter 3. How to Request a PC File

Chapter Table of Contents

Chapter 3. How to Request a PC File

This chapter teaches you how to turn mainframe data into PC files to use in your favorite PC program. Spectrum Writer makes using mainframe data in PC programs as easy as 1-2-3.

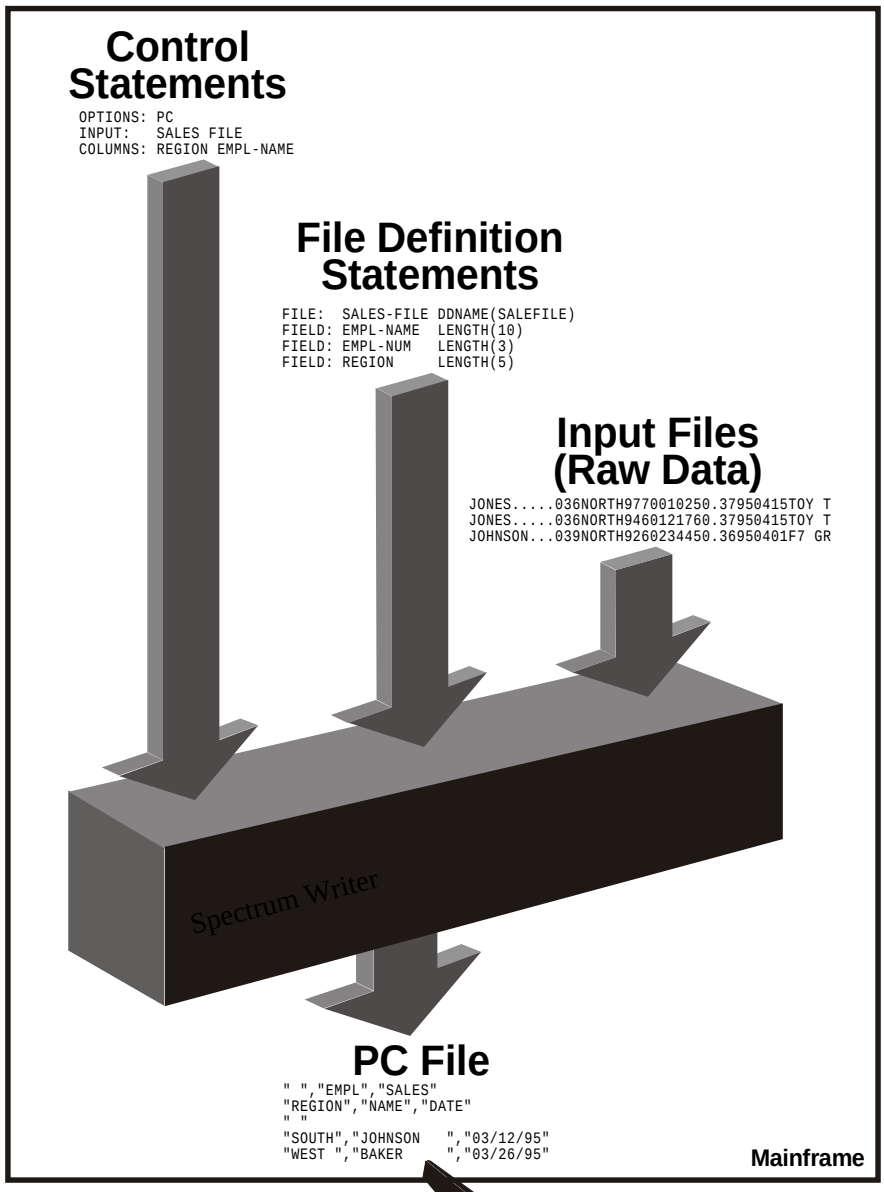
1. **Use Spectrum Writer to create a custom PC file on your mainframe.**
   Spectrum Writer's language is non–procedural, which means you just describe the *result* you want, not the programming steps needed to do it. Describe your PC file with a few simple "control statements". (These control statements are the same ones you already learned about in the previous chapter.) You can create a PC file with just three control statements. The lessons in this chapter teach you how to use the control statements.

   Once you've written the necessary control statements, submit a batch job to execute Spectrum Writer. Spectrum Writer examines the control statements describing the PC file you want. It automatically locates the appropriate "file definition" statements stored in a copy library. (These statements define your mainframe files.) Spectrum Writer then accesses the mainframe data and creates the desired PC file on your mainframe.

2. **Download the PC file to your PC.**
   Just use your shop's existing download facility to transfer the PC file to your PC.

3. **Use the PC file in your PC program.**
   Start the PC program and "open" or "import" the PC file with a few simple keystrokes. When you use the PC file in a spreadsheet program, for example, the program automatically moves the data into the correct rows and columns. Each downloaded record results in one *row* in a spreadsheet. And each field becomes a *column* in a spreadsheet.

Using mainframe data in your favorite PC program is as easy as that with Spectrum Writer!



Step 1. Create PC file on mainframe

Step 2. Download PC file to PC

Step 3. Use PC file in PC program

Figure 21 lists all of the Spectrum Writer control statements available when requesting PC files and describes the function of each one.

The remainder of this chapter is divided into seven easy lessons that explain how to use the various control statements to request PC files.

After reading just the first lesson, you will be able to produce useful PC files with Spectrum Writer. The other lessons introduce additional control statements, and explain their roles in producing increasingly sophisticated PC files. It is not necessary to read all of the other lessons initially. Nor is it necessary to read the lessons in sequential order. Read the summaries below and decide which lessons you need for the kind of PC files you want to produce.

Lesson 1. How to Produce a PC File in 5 Minutes.
This lesson shows how to produce PC files using just three simple control statements — the INPUT, COLUMNS and OPTIONS statements. You will use these three statements for every PC file you make.

Lesson 2. How to Include Only Certain Records In Your PC File.
This lesson shows how to use the INCLUDEIF statement to specify which mainframe records to include in your PC file.

Lesson 3. How to Create Your Own Fields.
This lesson shows you how to create your own fields by performing computations on existing fields. This is done with the COMPUTE statement.

Lesson 4. How to Specify the PC File Order.
This lesson shows how to sort your PC file into whatever order you want. The use of the SORT statement is explained.

Lesson 5. How to Create Control Breaks.
This lesson shows how to break a PC file up into sections, with subtotals appearing at the end of each section. The use of the BREAK statement to request such "control breaks" is explained.

Lesson 6. How to Create Summary Files.
This lesson shows you how to turn a PC file with subtotals into a small summary file that is more easily downloaded to a PC.

Lesson 7. How to Use Data from More Than One File.
This lesson shows how easy it is to read records from additional files when producing PC files. By adding a single READ statement, you automatically have access to all of the fields from an additional file.

These lessons show the most common use of each control statement. Most control statements also have additional features that are not discussed in these lessons. Additional ways to use these control statements are discussed in Chapter 4, "Beyond the Basics." The complete syntax for each control statement is shown in Chapter 10, "Control Statement Syntax."

SPECTRUM WRITER CONTROL STATEMENTS (GROUPED BY FUNCTION)

Statements that Define How Input Data Looks

FILE	Defines a file
FIELD	Defines a field within a file
ASM	Defines a file using an Assembler record layout
COBOL	Defines a file using a Cobol record layout
COMPUTE	Computes a new user-defined field

Statements that Specify the Input Files to Use to Make the PC File

INPUT	Specifies the primary input file
READ	Specifies an auxiliary input file

Statements that Describe the Body of a PC File

INCLUDEIF	Specifies which input records to include in the PC file
COLUMNS	Specifies the PC file's columns and column headings

Statements that Define the PC File Order and Control Breaks

SORT	Specifies the PC file order and, optionally, specifies control break fields
BREAK	Specifies control break processing

Miscellaneous Statements

OPTIONS	Specifies the kind of PC file needed, as well as various other special options
NEWOUT	Indicates that subsequent statements will define a new PC file
COPY	Copies additional control statements for processing

Figure 21. Spectrum Writer Control Statements for making PC Files

Lesson 1. How to Produce a PC File in 5 Minutes

This lesson teaches you how to turn your mainframe data into PC files using just three simple control statements. These statements are:

- the OPTIONS statement
- the INPUT statement
- the COLUMNS statement

Converting a Whole Mainframe File

With Spectrum Writer, it only takes three statements to convert an entire mainframe file into a comma-delimited PC file, ready to import into your favorite PC program:

```
OPTION:  PC
INPUT:   SALES-FILE
COLUMNS: SALES-FILE
```

Figure 22 shows a portion of the PC file created with the above statements. It also shows the spreadsheet obtained by importing the PC file into Excel.

Let's examine what each statement does. The **OPTION statement** above tells Spectrum Writer that you want to produce a comma-delimited PC file (instead of a report) in this run. Such PC files can be imported into virtually any PC spreadsheet, data base or word processing program.

The **INPUT statement** identifies the mainframe file that contains the data that you want to put into your PC file. In this case, we specified SALES–FILE. This is a sample "sales file" that is used in many of the examples in this manual. This file contains information about each sale made by the employees of an imaginary company.

The **COLUMNS statement** specifies what columns of data we want in our PC spreadsheet. Here you can name the individual fields from the input file that you want to use to populate the columns of the spreadsheet. However, in this example we named the input file itself. That means that we want a column in the spreadsheet for *every* field in the input file.

With just these three statements, we've given Spectrum Writer everything it needs to turn mainframe data into a PC file! That's all there is to creating custom PC files with Spectrum Writer. Three simple statements let you accomplish what would otherwise have taken an entire COBOL program to do!

> **Note:** The JCL used to create this PC file is shown on page 416 for OS/390 (page 430 for VSE).

These Control Statements:

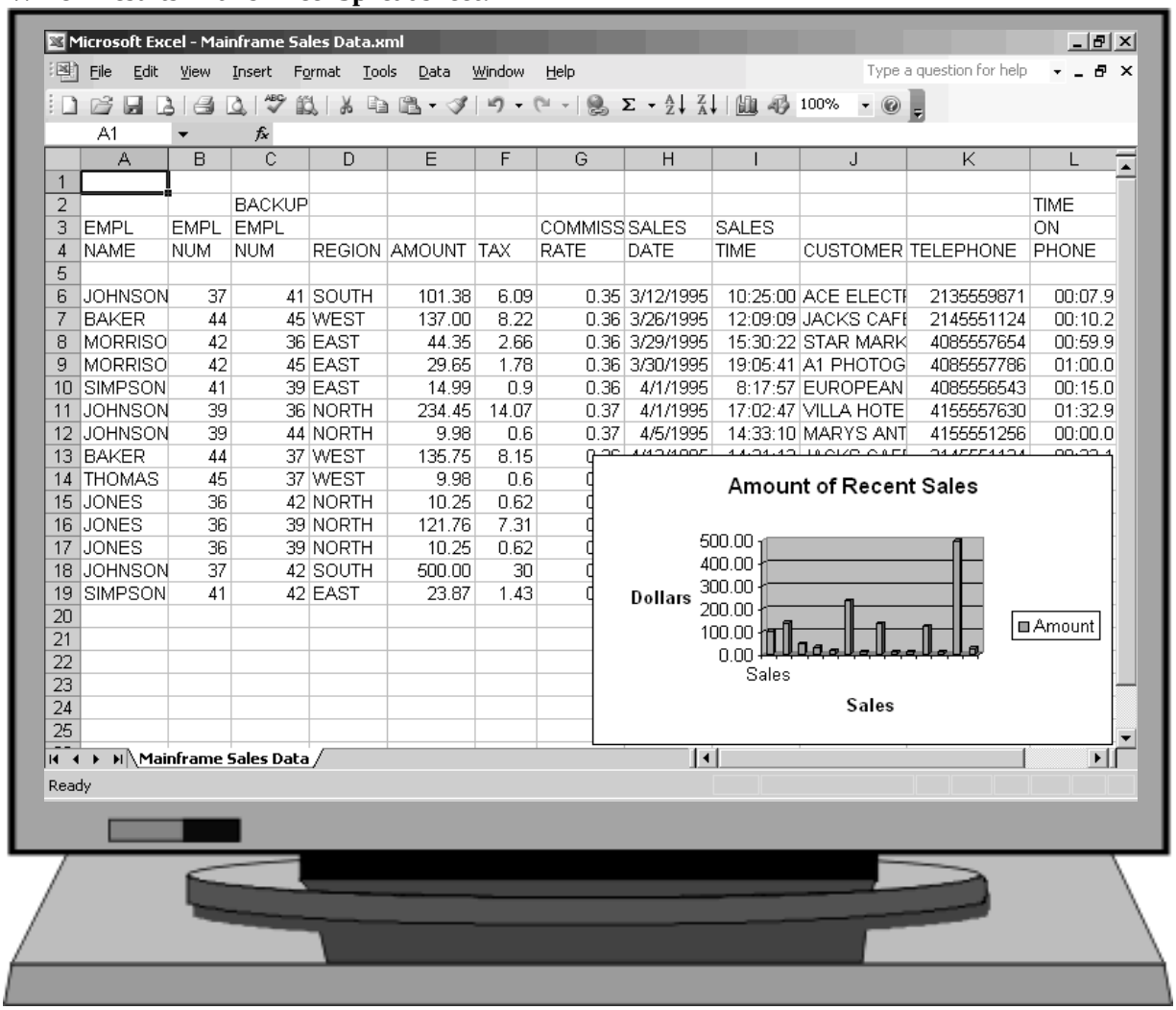
```
OPTION:  PC
INPUT:   SALES-FILE
COLUMNS: SALES-FILE
```

Produced this PC File:

```
" "," ","BACKUP"," "," "," "," "," "," "," "," "," ","TIME"
"EMPL","EMPL","EMPL"," "," "," ","COMMISSION","SALES","SALES"," "," ","ON"
"NAME","NUM","NUM","REGION","AMOUNT","TAX","RATE","DATE","TIME","CUSTOMER","TELEPHONE","PHONE"
" "," "," "," "," "," "," "," "," "," "," "," "," "," "," "," "
"JOHNSON   ","037","041","SOUTH",      101.38,      6.09,     0.350,"03/12/95","10:25:00","ACE ELECTRICAL ", 2135559871,"00:00:07.9"
"BAKER     ","044","045","WEST ",      137.00,      8.22,     0.360,"03/26/95","12:09:09","JACKS CAFE     ", 2145551124,"00:00:10.2"
"MORRISON  ","042","036","EAST ",       44.35,      2.66,     0.360,"03/29/95","15:30:22","STAR MARKET    ", 4085557654,"00:00:59.9"
"MORRISON  ","042","045","EAST ",       29.65,      1.78,     0.360,"03/30/95","19:05:41","A1 PHOTOGRAPHY ", 4085557786,"00:01:00.0"
```

(additional lines not shown)

Which Results in this Excel Spreadsheet:



	A	B	C	D	E	F	G	H	I	J	K	L
1												
2			BACKUP									TIME
3	EMPL	EMPL	EMPL				COMMISS	SALES	SALES			ON
4	NAME	NUM	NUM	REGION	AMOUNT	TAX	RATE	DATE	TIME	CUSTOMER	TELEPHONE	PHONE
5												
6	JOHNSON	37	41	SOUTH	101.38	6.09	0.35	3/12/1995	10:25:00	ACE ELECTI	2135559871	00:07.9
7	BAKER	44	45	WEST	137.00	8.22	0.36	3/26/1995	12:09:09	JACKS CAFI	2145551124	00:10.2
8	MORRISO	42	36	EAST	44.35	2.66	0.36	3/29/1995	15:30:22	STAR MARK	4085557654	00:59.9
9	MORRISO	42	45	EAST	29.65	1.78	0.36	3/30/1995	19:05:41	A1 PHOTOG	4085557786	01:00.0
10	SIMPSON	41	39	EAST	14.99	0.9	0.36	4/1/1995	8:17:57	EUROPEAN	4085556543	00:15.0
11	JOHNSON	39	36	NORTH	234.45	14.07	0.37	4/1/1995	17:02:47	VILLA HOTE	4155557630	01:32.9
12	JOHNSON	39	44	NORTH	9.98	0.6	0.37	4/5/1995	14:33:10	MARYS ANT	4155551256	00:00.0
13	BAKER	44	37	WEST	135.75	8.15						
14	THOMAS	45	37	WEST	9.98	0.6						
15	JONES	36	42	NORTH	10.25	0.62						
16	JONES	36	39	NORTH	121.76	7.31						
17	JONES	36	39	NORTH	10.25	0.62						
18	JOHNSON	37	42	SOUTH	500.00	30						
19	SIMPSON	41	42	EAST	23.87	1.43						
20												
21												
22												
23												
24												
25												

Figure 22. An Excel spreadsheet containing the entire mainframe SALES-FILE took only 3 statements

Another 5–Minute Example

Now let's make another PC file, this time using a different input file. This time we will create a Quattro Pro spreadsheet using data from the EMPL–FILE. EMPL–FILE is a sample employee file. We will create a simple employee directory from that file. In this example, we don’t need *every* field from the EMPL-FILE. We just want the spreadsheet to have columns showing: employee number, last name, first name, sex, social security number, date hired, and their city and state. We only need the following three statements:

```
OPTIONS: PC
INPUT:   EMPL-FILE
COLUMNS: EMPL-NUM  LAST-NAME  FIRST-NAME  SEX  SOCIAL-SEC-NUM
         HIRE-DATE  CITY  STATE
```

The OPTIONS statement above again specifies that we want to create a PC file rather than a report. The INPUT statement above specifies that the data we need for the PC file will come from the employee file (EMPL–FILE). The COLUMNS statement specifies the columns of data we want. Notice that we needed two lines for the COLUMNS statement in this example. You can continue a control statement onto as many lines as you need to. Just leave at least one blank space at the beginning of each continuation line.

The Quattro Pro spreadsheet resulting from the above statements is shown in **Figure 23**.

You have now seen two examples showing just how easy it is to create PC files with Spectrum Writer. That's all there is to it! You now know enough to request basic PC files from the files at your company.

Using Your Company's Files

You may be wondering how Spectrum Writer knows the names of your company's files and fields. The answer is that your company's files are defined to Spectrum Writer by other control statements that are kept in a Spectrum Writer "copy library." For example, the statements used to define the sample files used in the preceding examples are shown in Appendix F, "Files Used in Examples" (page 648).

For a list of the file names and field names available for you to use, ask your programmer. They can print that information from the Spectrum Writer Copy Library, in a format similar to that shown in Appendix F.

If you already know the name of the *file* to use, you can use a "dummy" run to easily get a list of all of its fields. Just use an INPUT statement with the SHOWFLDS(YES) parm, like this:

```
INPUT: SALES-FILE  SHOWFLDS(YES)
```

The above statement tells Spectrum Writer to print (in the control statement listing) a list of all of the fields defined for SALES–FILE.

If a file that you need to use has not yet been defined, see Chapter 6, "How to Define Your Input Files" to learn how to do that.

These Control Statements:

```
OPTIONS: PC
INPUT:   EMPL-FILE
COLUMNS: EMPL-NUM  LAST-NAME  FIRST-NAME  SEX  SOCIAL-SEC-NUM
         HIRE-DATE  CITY  STATE
```

Result in this Quattro Pro Spreadsheet:

	A	B	C	D	E	F	G	H
1					SOCIAL			
2	EMPL	LAST	FIRST		SEC	HIRE		
3	NUM	NAME	NAME	SEX	NUM	DATE	CITY	STATE
4								
5	036	JONES	JERRY	M	12098765	01/31/80	SAN FRANCISCO	CA
6	037	JOHNSON	THOMAS	M	912040334	06/21/75	SCOTTSDALE	AZ
7	039	JOHNSON	LINDA	F	4779981	11/25/79	SANTA ROSA	CA
8	040	MACDONA	RICHARD	M	889790013	07/04/82	PLEASANTON	CA
9	041	SIMPSON	TIMOTHY	M	112050456	12/01/82	ARCADIA	CA
10	042	MORRISO	MICHAEL	M	900120556	11/30/79	GLENDALE	CA
11	043	CHRISTOP	MELISSA	F	415090761	08/15/81	PHOENIX	AZ
12	044	BAKER	VIVIAN	F	878190156	06/04/82	WALNUT CREEK	CA
13	045	THOMAS	MARTIN	M	776838221	06/04/82	CONCORD	CA
14								
15								
16								
17								
18								
19								
20								
21								
22								

Figure 23. A Quattro Pro employee directory produced with just three control statements

Summary

Here is a summary of what we learned in this lesson:

- the OPTIONS statement lets you request that a PC file be produced, instead of a report
- the INPUT statement tells Spectrum Writer which mainframe input file contains the data needed in your PC file
- the COLUMNS statement tells Spectrum Writer what "columns" of data to put in your PC file
- by using just these three statements you can produce a PC file

The next lesson will teach you how to limit the records that are included in your PC file.

To Learn More

To learn more about writing control statements in general, see Chapter 9, "General Syntax Rules." In that chapter you will learn such things as:

- **how long** each line can be (page 443)
- how to **continue** control statements onto multiple lines (page 444)

There are some additional features associated with the OPTIONS, INPUT and COLUMNS statements which we have not covered in this lesson. Some of these additional features are discussed in Chapter 4, "Beyond the Basics." Examples of additional features are:

- how to specify your own **column headings** for a PC file (page 130)
- how to **suppress column headings** in your PC file (page 130)
- how to reserve **more room for numeric columns** in your PC file (page 135)
- how to create a column that contains a **literal text** (page 126)
- how to produce **multiple rows in the PC file** for each input record (page 151)
- how to turn data from **DB2 tables and views** into PC files (page 397)
- how to turn data from **existing mainframe reports** into PC files (page 258)
- how to customize your PC file, by specifying such things as: the **delimiter character** to use, the **"text qualifier"** character to use (for example, a quotation mark), the date format to use, whether to enclose dates in quotation marks, etc. (page 275)

The complete syntax for the OPTIONS, INPUT and COLUMNS statements appears in Chapter 10, "Control Statement Syntax" (pages 555, 542 and 498 respectively).

Lesson 2. How to Include Only Certain Records In Your PC File

This lesson teaches you how to select only certain records from the input file for inclusion in your PC file. The control statement discussed is:

- the INCLUDEIF statement

How to Use the INCLUDEIF Statement

In the previous lesson we saw how to select certain *fields* to be downloaded. (We used the COLUMNS statement to identify the fields that we wanted.) Now let's look at how to download only selected *records* from the mainframe file. We will use the INCLUDEIF ("include if") statement.

When no INCLUDEIF statement is specified, Spectrum Writer includes every record from the mainframe file. Use the INCLUDEIF statement to tell Spectrum Writer to "include" a record in the PC file only "if" one or more conditions are met.

This feature is very useful when you are working with large mainframe files. Downloading the entire file might take a long time (and use up lots of hard disk). Using the INCLUDEIF statement lets you download only the records that you actually need.

For example, assume that we want to download data from the SALES–FILE to a spreadsheet similar to the one in **Figure 22** (page 89). But this time let's just download the data for the employee named Jones. We simply add the following INCLUDEIF statement to our other control statements:

```
INCLUDEIF:  EMPL-NAME = 'JONES'
```

The above INCLUDEIF statement tells Spectrum Writer to "include" records from the SALES–FILE "if" the EMPL–NAME field is equal to 'JONES'. Spectrum Writer still reads through the entire SALES–FILE, just like before. But now it *examines* each record before including it in the PC file. If the record's EMPL–NAME field contains the value 'JONES', then the record is included in the PC file. If the EMPL–NAME field contains any other value, then that record is not included in the PC file.

Figure 24 shows an Access table produced using the above statement. Only the sales made by Jones appear in that table.

> **Note:** Notice that we specified an additional option in the OPTIONS statement in this example. The **NOCOLHDG option** tells Spectrum Writer that we do not want column headings in the PC file. Column headings are often desirable in file imported into *spreadsheets*. But this PC file is being imported into an Access database. The records should all contain real data. During the import process, Access let us manually specify a "field name" to use for each column.

The INCLUDEIF statement may appear anywhere after the INPUT statement. Only one INCLUDEIF statement is allowed per run, but it may contain as many conditions as you like.

These Control Statements:

```
OPTIONS:   PC NOCOLHDG
INPUT:     SALES-FILE
INCLUDEIF: EMPL-NAME = 'JONES'
COLUMNS:   REGION  EMPL-NAME  SALES-DATE  SALES-TIME  CUSTOMER  AMOUNT  TAX
```

Result in this Access Table:

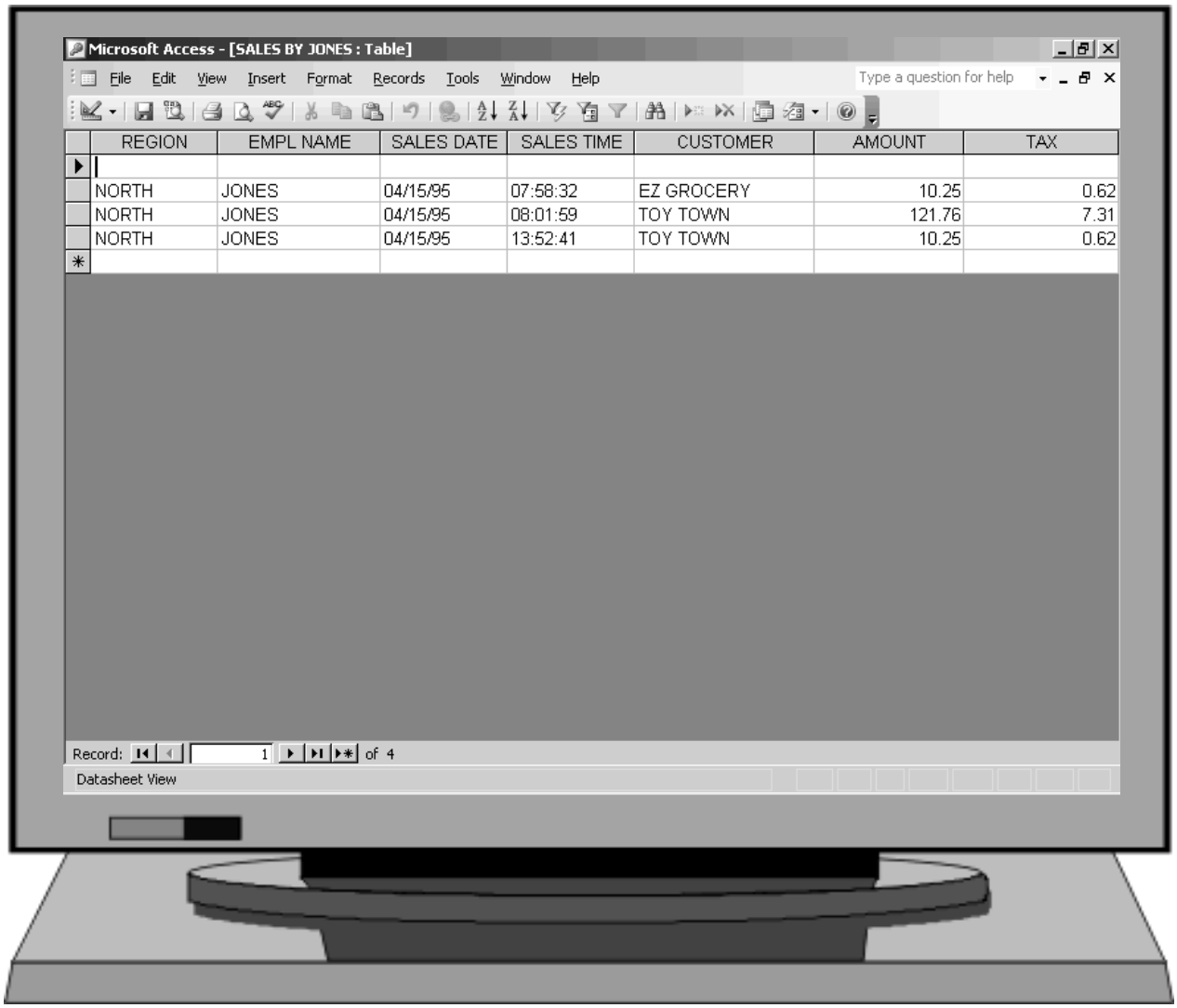


Figure 24. Using an INCLUDEIF statement to specify which records to include in a PC file

By the way, the INCLUDEIF statement can refer to any of the fields in the input file (as well as any COMPUTE field). You are not limited to just the fields that are listed in the COLUMNS statement.

How to Write Conditional Expressions

The INCLUDEIF statement simply contains a **conditional expression**. The complete rules for writing conditional expressions are explained in "Conditional Expressions" (page 459). Briefly, a conditional expression contains one or more "conditions," separated with the words AND or OR. A **condition** usually involves comparing the contents of one field with the contents of another field, or with a literal value. Let's look at some more examples of INCLUDEIF statements and their conditional expressions.

> **Note:** If you are a programmer, you will notice that the syntax for conditional expressions is very similar to the syntax used in "IF statements" in COBOL, PL/1, and BASIC. If you are familiar with any of these languages, you should find it especially easy to write INCLUDEIF statements.

You may want your PC file to include all records which **do not contain** a certain value. Do this by specifying "not equal" in your condition. For example:

```
INCLUDEIF:  EMPL-NAME ¬= 'JONES'
```

The above statement specifies that the PC file should include all records from the input file whose EMPL–NAME field is not equal to 'JONES'.

> **Note:** In addition to ¬=, you can also use <> to indicate "not equal", like this:
>
> ```
> INCLUDEIF: EMPL-NAME <> 'JONES'
> ```

You may want to include a record in your PC file if **either of two conditions** is true. To do this, use an INCLUDEIF statement with two conditions, separated by the word OR. Consider the following statement:

```
INCLUDEIF:  EMPL-NAME = 'JONES'  OR  AMOUNT > 100
```

The above statement states that a record should be included in the PC file "if the EMPL–NAME field is equal to 'JONES' or if the AMOUNT field is greater than 100." The word OR indicates that records from the input file will be included if either one (or both) of the conditions are true.

Notice in the above statement that we enclosed 'JONES' in single quotation marks, while we did not use quotation marks around the 100. That is because EMPL–NAME is a character field, while AMOUNT is a numeric field. Character literals (such as 'JONES') must be enclosed in quotation marks. You can use either single (') or double (") quotation marks. But numeric literal (such as 100), as well as date and time literals, are *not* enclosed in quotation marks. Numeric literal also must not contain commas. (The rules for writing literals are thoroughly explained in "How to Write Literals" on page 448.)

As another example, you may want to include records in your PC file when **both of two conditions** are true. For example, let's say we want a listing only of sales that were made by Jones and that were also for an amount over $100. For this PC file, two conditions must

both be true: the EMPL–NAME field must be equal to 'JONES' and the AMOUNT field must be over 100. Use the word AND to specify that both conditions must be true, like this:

```
INCLUDEIF:  EMPL-NAME = 'JONES'  AND  AMOUNT > 100
```

Now as Spectrum Writer reads each record from the input file, it will include a record in the PC file only "if the EMPL–NAME field is equal to 'JONES' and the AMOUNT field is greater than 100."

Here is an example of including records in a PC file based on the contents of a **date field**:

```
INCLUDEIF:  SALES-DATE  >  4/15/1995
```

The above statement specifies that records should be included in the PC file only if their SALES–DATE field contains a date greater than (after) April 15, 1995.

> **Note:** You may be wondering if you need to use a different format for your date literals when you know that a particular date field is stored in a record as a Julian date (YYDDD format.) The answer is no. All date literals in your control statements should be written as MM/DD/YYYY (or MM/DD/YY). Spectrum Writer automatically takes care of any date conversions that may be required. Thus, you test Julian date fields just like all other date fields:
>
> ```
> INCLUDEIF: JULIAN-START-DATE >= 1/1/2000
> ```

Here is an example of including records in a PC file based on the contents of a **time field**:

```
INCLUDEIF:  SALES-TIME  <  17:00:00
```

The above statement specifies that records should be included in the PC file only if their SALES–TIME field contains a time less than (before) 17:00:00 (which is 5 PM).

> **Note:** You are allowed to omit the seconds in your time literals, if you prefer. When the seconds are not specified, zero seconds is assumed. Thus, you could also write the above statement this way:
>
> ```
> INCLUDEIF:  SALES-TIME  <  17:00
> ```

If your INCLUDEIF statement contains both the words OR and AND, you should use parentheses to indicate the order in which to perform the comparisons. Consider the following statement:

```
INCLUDEIF: EMPL-NAME = 'JONES'  OR
           (SALES-DATE > 4/15/1995  AND  SALES-DATE < 4/30/1995)
```

In the above statement, records will be included if the EMPL–NAME field is equal to 'JONES', *or* if both of the SALES–DATE comparisons are true. The parentheses cause the two SALES–DATE comparisons to be treated as one condition. That condition is true if the SALES–DATE is greater than April 15, 1995 and is less than April 30, 1995.

> **Note:** In addition to the actual words AND and OR, you can also use the symbols "&" and "|", respectively, in your conditional expressions.

Summary

Here is a summary of what we learned in this lesson:

- use the INCLUDEIF statement when you want to include only certain records from the input file in your PC file
- the INCLUDEIF statement contains one or more conditions, separated by the words AND or OR
- groups of conditions can be enclosed in parentheses, to indicate the order in which the comparisons should be performed

The next lesson will show you how to compute your own new fields to download to your PC.

To Learn More

There are some additional features associated with the INCLUDEIF statement which we have not covered in this lesson. These additional features are discussed in "Conditional Expressions" (page 459). The additional features include:

- how to use the keyword **NOT** (or the symbol ¬) to negate a condition (page 469)
- how to **scan** a character field, to see if a certain text exists anywhere within the field (page 460)
- how to specify conditions based on **bit fields** (page 465)
- how to specify a condition based on a field's raw **hexadecimal** value (page 464)
- what to do if you prefer to specify **date literals** in DD/MM/YY or DD/MM/YYYY format (page 140), like this:

```
INCLUDEIF: SALES-DATE > 15/4/1995
```

- how the **KEYRANGE** and **STOPWHEN** parms of the INPUT statement can be used to limit the records included in your run (page 542)

The complete syntax for the INCLUDEIF statement appears in Chapter 10, "Control Statement Syntax" (page 540).

Lesson 3. How to Create Your Own Fields

This lesson teaches you how to create your own fields to include in your PC file. The control statement discussed is:

- the COMPUTE statement

Sometimes the data you need to download to your PC program is not contained in the input file. Yet the necessary data might be easily computed from one or more fields which *are* in the input file. In such cases, simply create a new field by using the COMPUTE statement.

Creating Numeric Fields

A COMPUTE statement specifies the name of the new field to create and supplies a *computational expression* to use in assigning a value to that field. The complete rules for computational expressions are discussed in "Computational Expressions" (page 472). Generally, your expression will consist of one or more arithmetic operations performed on numeric fields and/or numeric literals.

For example, the sample SALES–FILE has numeric fields named AMOUNT and TAX. We can use the COMPUTE statement to create a new field containing the total amount due just by adding those two fields together, like this:

```
COMPUTE:  TOTAL-AMOUNT = AMOUNT + TAX
```

The above statement creates a new field named TOTAL–AMOUNT. It is computed by adding the AMOUNT field and the TAX field together. Now that the TOTAL–AMOUNT field has been created, we can use that field in *any way* that other fields can be used. For example, a computed field can be used: as a column of data in your PC file; as a sort field; as a control break field; as part of a conditional expression (in the INCLUDEIF statement); even as an operand in subsequent COMPUTE statements to create other fields.

The Access table in **Figure 25** was obtained by using the above COMPUTE statement.

COMPUTE statements normally appear somewhere after the INPUT statement. The COMPUTE statement must appear before any other control statement that refers to the field being created. In **Figure 25**, the COMPUTE statement for TOTAL–AMOUNT had to come before the COLUMNS statement, since the COLUMNS statement referred to that field.

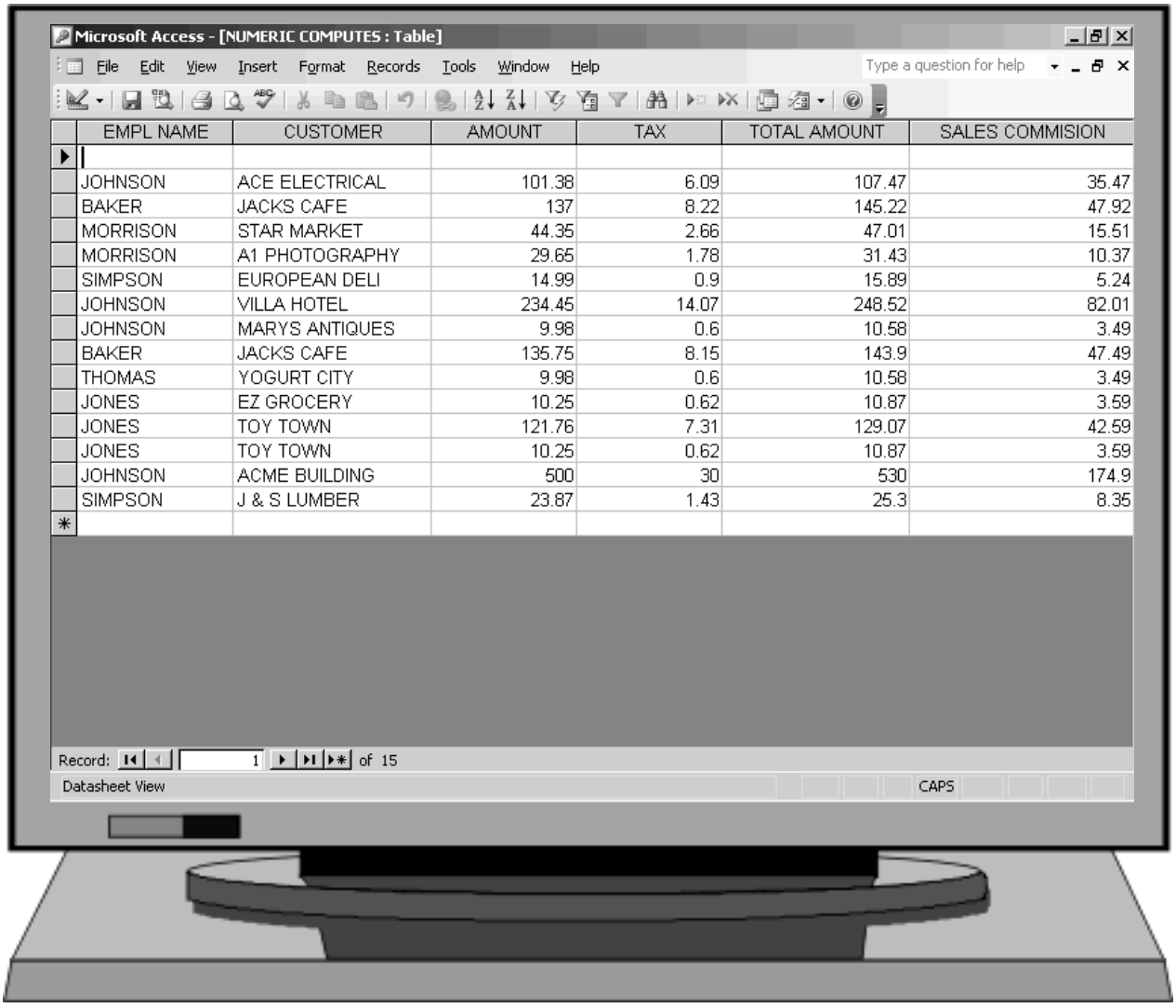
You can perform addition, subtraction, multiplication, and division in the COMPUTE statement. Use the +, –, * and / symbols, respectively. You may also use parentheses as needed to indicate the order in which the operations should be performed.

> **Note:** When performing subtraction, always put a **blank space before and after the minus sign**. Otherwise, the minus sign will appear to be part of a field name. Blanks are optional around the other arithmetic operators.

These Control Statements:

```
OPTIONS: PC NOCOLHDG
INPUT:   SALES-FILE
COMPUTE: TOTAL-AMOUNT        = AMOUNT + TAX
COMPUTE: SALES-COMMISSION(2) = TOTAL-AMOUNT * .33
COLUMNS: EMPL-NAME  CUSTOMER  AMOUNT  TAX  TOTAL-AMOUNT  SALES-COMMISSION
```

Result in this Microsoft Access table:



EMPL NAME	CUSTOMER	AMOUNT	TAX	TOTAL AMOUNT	SALES COMMISION
JOHNSON	ACE ELECTRICAL	101.38	6.09	107.47	35.47
BAKER	JACKS CAFE	137	8.22	145.22	47.92
MORRISON	STAR MARKET	44.35	2.66	47.01	15.51
MORRISON	A1 PHOTOGRAPHY	29.65	1.78	31.43	10.37
SIMPSON	EUROPEAN DELI	14.99	0.9	15.89	5.24
JOHNSON	VILLA HOTEL	234.45	14.07	248.52	82.01
JOHNSON	MARYS ANTIQUES	9.98	0.6	10.58	3.49
BAKER	JACKS CAFE	135.75	8.15	143.9	47.49
THOMAS	YOGURT CITY	9.98	0.6	10.58	3.49
JONES	EZ GROCERY	10.25	0.62	10.87	3.59
JONES	TOY TOWN	121.76	7.31	129.07	42.59
JONES	TOY TOWN	10.25	0.62	10.87	3.59
JOHNSON	ACME BUILDING	500	30	530	174.9
SIMPSON	J & S LUMBER	23.87	1.43	25.3	8.35

Figure 25. Using the COMPUTE statement to create numeric fields for a PC file

As another example of a creating a numeric field, let's say we wanted to compute a sales commission for each sale. The commission will be 33% of the total value of the sale, including the tax. We could compute the sales commission with the following statement:

```
COMPUTE:  SALES-COMMISSION(2) = TOTAL-AMOUNT * .33
```

This statement creates a new field called SALES–COMMISSION which is computed by multiplying TOTAL–AMOUNT by .33. Notice that we used the result of our previous COMPUTE statement to perform the computation in this statement.

The "(2)" after SALES-COMMISSION tells Spectrum Writer that we want that field to have 2 decimal places. Without that parm, Spectrum Writer would have defaulted to keeping 4 decimal places (in this example).

The Access table in **Figure 25** (page 99) uses the above statement.

In addition to the basic arithmetic operations, there are also a large number of built–in functions that you can use in the COMPUTE statement. These built–in functions allow you to perform more complex mathematical operations on numeric operands. A complete list of built–in functions is found in Appendix D, "Built-In Functions" (page 628).

Creating Character Fields

So far we have been creating numeric fields. Now let's consider how to create your own *character* fields. There is only one operation used in computing character fields. It is the **concatenation** operation. (Don't let that word scare you if it is new to you. "Concatenating" simply means "stringing together" two or more character fields.) The plus sign (+) is used as the symbol for concatenation. For example:

```
COMPUTE: WHOLE-NAME = LAST-NAME + FIRST-NAME
```

The above statement creates a new field named WHOLE–NAME. It is created by concatenating the contents of the LAST–NAME field and the contents of the FIRST–NAME field. The result is a single field which now contains both the last and first names of the employee. The new field will be 30 bytes long — the combined length of the two operands.

You can also concatenate more than two fields together. For example:

```
COMPUTE: MAILING-CODE = STATE + '–' + EMPL-NUM
```

This example creates a new field called MAILING–CODE which consists of the contents of the STATE field, followed by a dash, followed by the contents of the EMPL–NUM field.

In addition to the concatenation operation, there are also a number of built–in functions that can be used when creating character fields. For example, the #LEFT function can be used to extract the leftmost *n* bytes of a character field. Here is an example of how to use the #LEFT built–in function:

```
COMPUTE: FIRST-INITIAL = #LEFT(FIRST-NAME,1)
```

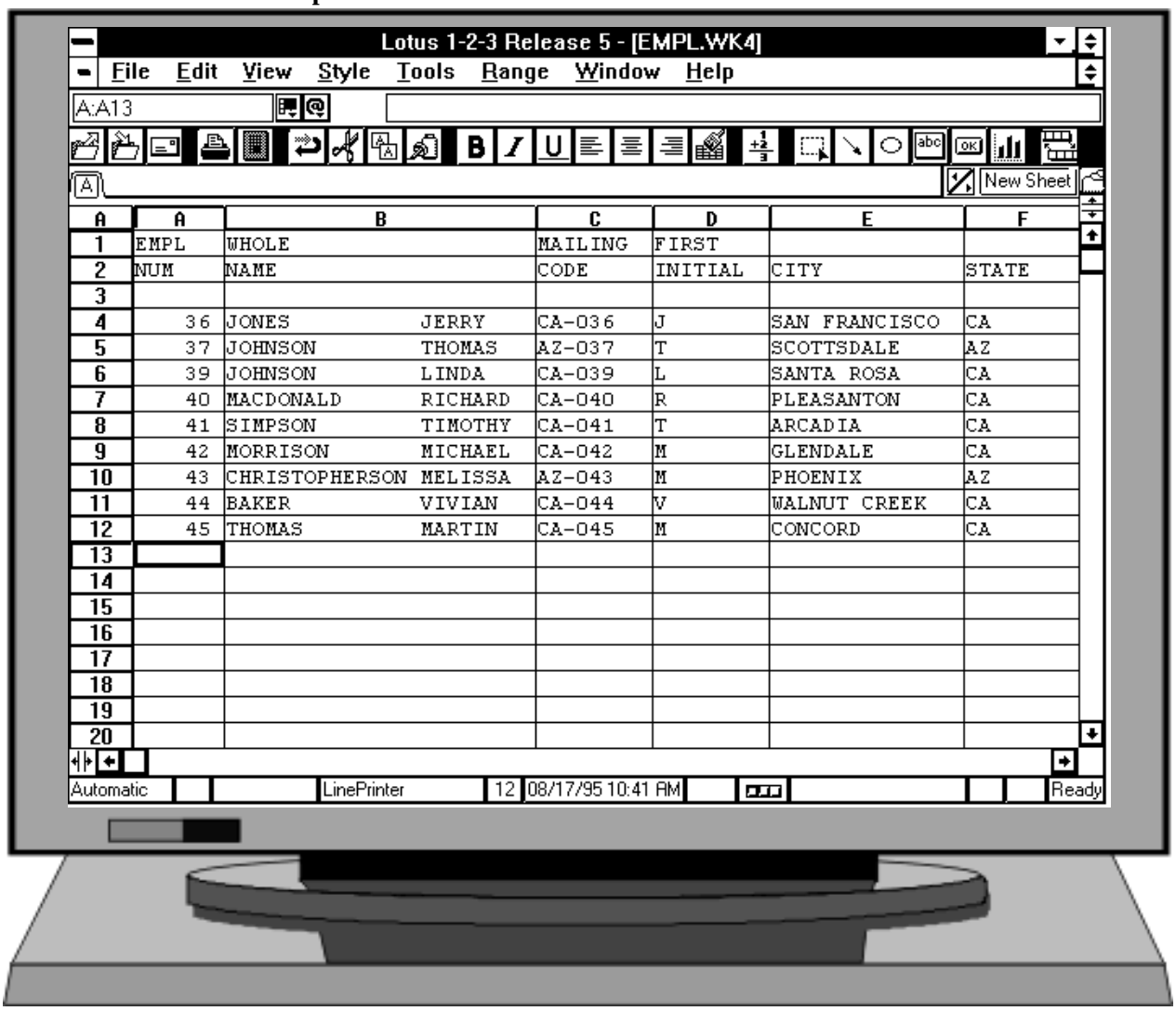
This statement creates a new character field which consists of only the first character (that is, the leftmost 1 byte) of the FIRST–NAME field.

Figure 26 shows a spreadsheet that uses each of the above COMPUTE statements.

These Control Statements:

```
OPTIONS: PC
INPUT:   EMPL-FILE
COMPUTE: WHOLE-NAME = LAST-NAME + FIRST-NAME
COMPUTE: MAILING-CODE = STATE + '-' + EMPL-NUM
COMPUTE: FIRST-INITIAL = #LEFT(FIRST-NAME,1)
COLUMNS: EMPL-NUM WHOLE-NAME MAILING-CODE FIRST-INITIAL CITY STATE
```

Result in this Lotus 1-2-3 spreadsheet:



	A	B	C	D	E	F
1	EMPL	WHOLE	MAILING	FIRST		
2	NUM	NAME	CODE	INITIAL	CITY	STATE
3						
4	36	JONES JERRY	CA-036	J	SAN FRANCISCO	CA
5	37	JOHNSON THOMAS	AZ-037	T	SCOTTSDALE	AZ
6	39	JOHNSON LINDA	CA-039	L	SANTA ROSA	CA
7	40	MACDONALD RICHARD	CA-040	R	PLEASANTON	CA
8	41	SIMPSON TIMOTHY	CA-041	T	ARCADIA	CA
9	42	MORRISON MICHAEL	CA-042	M	GLENDALE	CA
10	43	CHRISTOPHERSON MELISSA	AZ-043	M	PHOENIX	AZ
11	44	BAKER VIVIAN	CA-044	V	WALNUT CREEK	CA
12	45	THOMAS MARTIN	CA-045	M	CONCORD	CA
13						
14						
15						
16						
17						
18						
19						
20						

Figure 26. Using the COMPUTE statement to create character fields for a PC file

Assigning Values to Fields Based on Conditions

Up until now we have been using "simple" COMPUTE statements. In a **simple COMPUTE statement**, the value of the new field is defined by a single computational expression.

But it is also possible to use conditional logic in a COMPUTE statement. In **conditional COMPUTE statements**, one of several different expressions will be used to assign a value to the new field. The expression that is used will depend on one or more conditions that you specify. Conditional COMPUTE statements can be very powerful tools in producing PC files. Here is an example of a conditional COMPUTE statement:

```
COMPUTE: BONUS = WHEN(HIRE-DATE <  1/1/1980)  ASSIGN(TOTAL-SALES * .08)
                 WHEN(HIRE-DATE >= 1/1/1980)  ASSIGN(TOTAL-SALES * .05)
```

The above statement creates a field named BONUS. However, in this example the BONUS field can be computed in one of two ways: for employees hired before January 1, 1980, the bonus is 8 percent of total sales (TOTAL–SALES * .08). But, for employees hired on or after January 1, 1980, the bonus is only 5 percent of total sales (TOTAL–SALES * .05).

When assigning a value to the BONUS field, Spectrum Writer evaluates the conditional expression in each WHEN parm. As soon as a WHEN expression is found that is true, the computational expression from the corresponding ASSIGN parm is used to assign a value to BONUS. (Any remaining WHEN parms are not evaluated.)

You may have as many pairs of WHEN and ASSIGN parms as you like in a COMPUTE statement. If none of the WHEN expressions are true, a value of zero will be assigned to the field. To assign some other value when none of the WHEN parms are true, you may use the ELSE parm. For example:

```
COMPUTE: BONUS = WHEN(HIRE-DATE <  1/1/1980)  ASSIGN(TOTAL-SALES * .08)
                 ELSE                         ASSIGN(TOTAL-SALES * .05)
```

The above statement has the same effect as the previous example, but is a little simpler. It has only one WHEN expression. For employees whose hire date is before January 1, 1980, the bonus will be computed based on 8 percent. For all other cases, the bonus will be computed based on 5 percent.

You may also use conditional COMPUTE statements to create *character* fields. For example:

```
COMPUTE: TITLE = WHEN(SEX = 'M')  ASSIGN('MR')
                 ELSE             ASSIGN('MS')
```

The above statement creates a new field called TITLE. The contents of TITLE will be "MR" if the SEX field contains an "M", and "MS" otherwise.

Figure 27 shows a Lotus spreadsheet obtained by using some of the conditional COMPUTE statements just discussed.

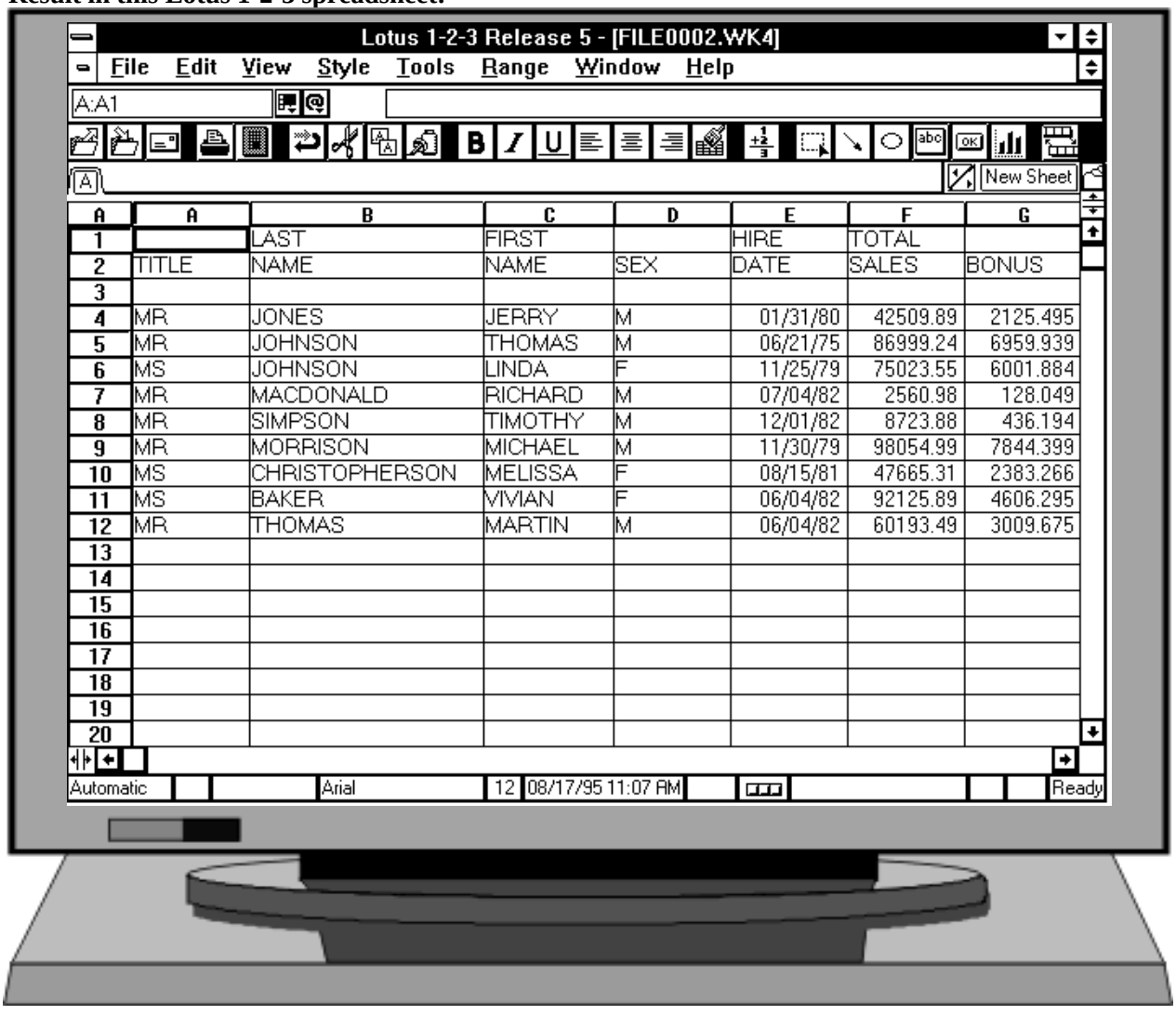
When defining character fields with a conditional COMPUTE statement, a value of **spaces** is assigned if none of the WHEN expressions are true and no ELSE parm is specified.

All of our examples so far have used just a single condition within the WHEN parm. You can, however, use any valid conditional expression within the WHEN parm. The conditional expression can contain as many different conditions as you like, separated with the words AND and OR, and optionally grouped with parentheses. (A conditional expression is the sort of expression that is allowed in the INCLUDEIF statement, as was described in "How to Write

These Control Statements:

```
OPTIONS:  PC
INPUT:    EMPL-FILE
COMPUTE:  BONUS = WHEN(HIRE-DATE <  1/1/1980)  ASSIGN(TOTAL-SALES * .08)
                  WHEN(HIRE-DATE >= 1/1/1980)  ASSIGN(TOTAL-SALES * .05)
COMPUTE:  TITLE = WHEN(SEX = 'M')  ASSIGN('MR')
                  ELSE             ASSIGN('MS')
COLUMNS:  TITLE  LAST-NAME  FIRST-NAME  SEX  HIRE-DATE  TOTAL-SALES  BONUS
```

Result in this Lotus 1-2-3 spreadsheet:



	A	B	C	D	E	F	G
1		LAST	FIRST		HIRE	TOTAL	
2	TITLE	NAME	NAME	SEX	DATE	SALES	BONUS
3							
4	MR	JONES	JERRY	M	01/31/80	42509.89	2125.495
5	MR	JOHNSON	THOMAS	M	06/21/75	86999.24	6959.939
6	MS	JOHNSON	LINDA	F	11/25/79	75023.55	6001.884
7	MR	MACDONALD	RICHARD	M	07/04/82	2560.98	128.049
8	MR	SIMPSON	TIMOTHY	M	12/01/82	8723.88	436.194
9	MR	MORRISON	MICHAEL	M	11/30/79	98054.99	7844.399
10	MS	CHRISTOPHERSON	MELISSA	F	08/15/81	47665.31	2383.266
11	MS	BAKER	VIVIAN	F	06/04/82	92125.89	4606.295
12	MR	THOMAS	MARTIN	M	06/04/82	60193.49	3009.675
13							
14							
15							
16							
17							
18							
19							
20							

Figure 27. Assigning values to computed fields based on conditions

Conditional Expressions" on page 95.) The complete rules for writing conditional expressions are given in "Conditional Expressions" (page 459). Additional examples of COMPUTE statements are shown beginning on page 513.

Summary

Here is a summary of what we learned in this lesson:

- the COMPUTE statement is used to create new fields
- a **simple COMPUTE statement** assigns the result of a single computational expression to a new field
- a **conditional COMPUTE statement** uses one of several different computational expressions, depending on the conditions that you specify

The next lesson will teach you how to sort your PC file into whatever order you want.

To Learn More

There are some additional features associated with the COMPUTE statement which we have not covered in this lesson. Some of these additional features are discussed as topics in Chapter 4, "Beyond the Basics." Examples of additional topics include:

- how to create **date** fields (page 514)
- how to create **time** fields (page 272)
- how to create **bit** (boolean) fields (page 514)
- how to specify how many **decimal places** a numeric or time field should contain (page 511)
- how to specify **column headings** for the fields you create (page 511)
- how to specify how your field should be **formatted** when it is printed in a report (page 510)
- how to specify whether a numeric or time field should be totalled in the **Grand Totals** line at the end of the report (page 148)
- how to **retain the previous value** of a COMPUTE field in certain cases (page 234)

The complete syntax for the COMPUTE statement appears in Chapter 10, "Control Statement Syntax" (page 506).

Lesson 4. How to Specify the PC File Order

This lesson teaches you how to sort your PC file into any order you want. The control statement discussed is:

- the SORT statement

How to Use the SORT Statement

When no SORT statement is specified, Spectrum Writer defaults to writing out the PC file records in their original input file order. For example, the records in the sample SALES–FILE are stored in sales date order. Therefore, the sales spreadsheets in the previous lessons (for example, page 89) all appeared in sales date order. The EMPL–FILE sample file is a VSAM file stored in EMPL–NUM order. Therefore, the earlier spreadsheets from that file were in employee number order (see page 101 for an example).

To create a PC file in a different order, just add a SORT statement. The SORT statement can appear anywhere after the INPUT statement. Only one SORT statement is allowed per run, but it may contain as many "sort fields" as you like. Spectrum Writer will sort your PC file on all of the sort fields.

For example, let's request a PC file from the SALES–FILE and sort it on three fields:

```
SORT:  REGION  EMPL-NAME  SALES-DATE
```

To begin with, the PC file will be sorted according to the first sort field — REGION. If there are multiple records for the same REGION, then those records will be further sorted using the second sort field, EMPL–NAME. Records having the same value for both the REGION and the EMPL–NAME fields will be further sorted on the third sort field — SALES–DATE.

Figure 28 shows a Microsoft Works spreadsheet obtained by using the above statement.

By the way, the SORT statement can refer to any of the fields in the input file (as well as any COMPUTE field). You are not limited to just the fields that are listed in the COLUMNS statement.

By default, Spectrum Writer sorts PC files into **ascending order** on each sort field. If you want to sort the PC file into **descending order** for a field, put the DESCENDING parm (or just DESC) in parentheses immediately after the field name. For example, to sort a PC file into reverse employee number order, you could use this SORT statement:

```
SORT:  EMPL-NUM(DESC)
```

Automatic Sorting

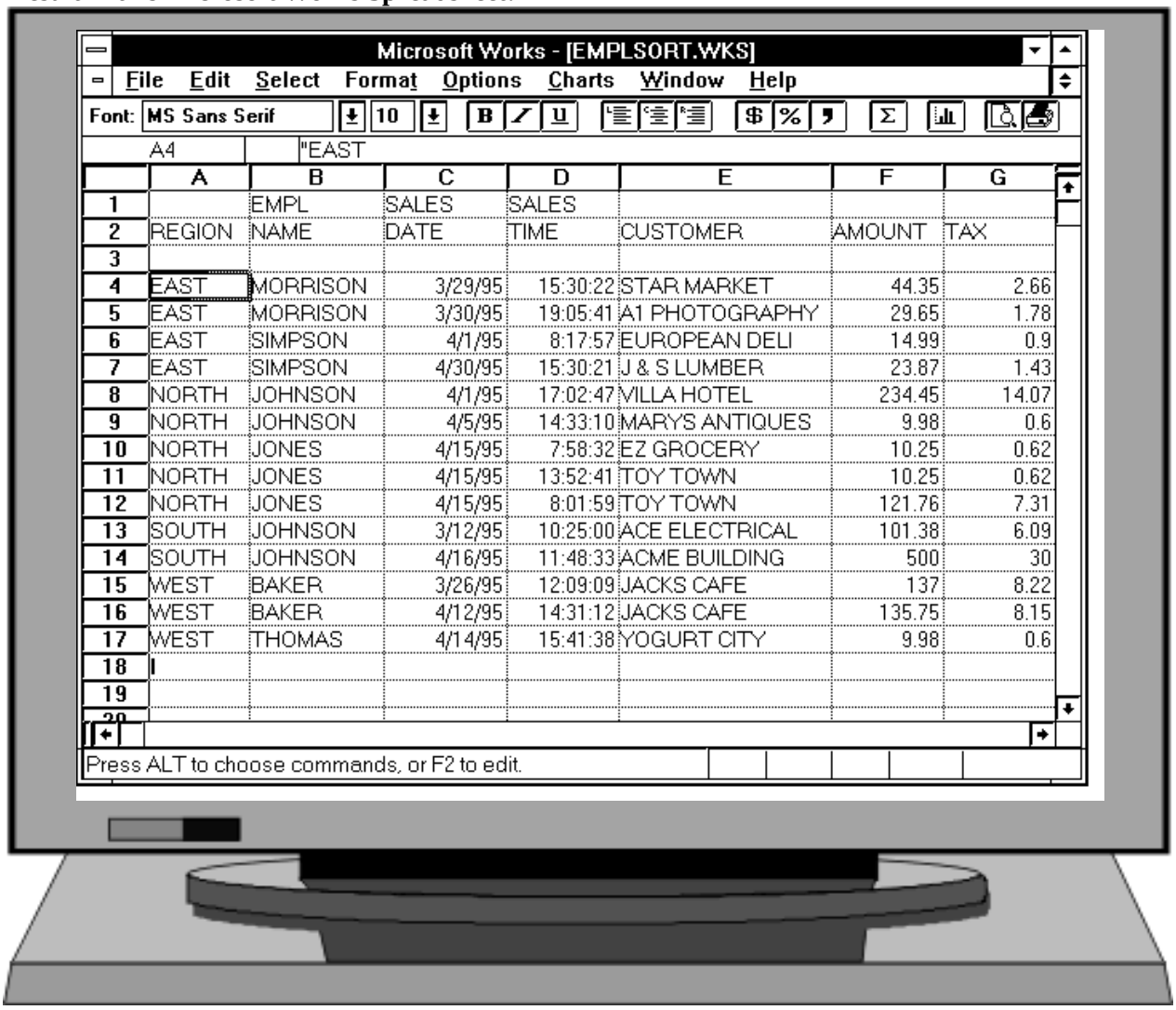
If you prefer, you can let Spectrum Writer *automatically* sort your PC file for you. To have your PC file automatically sorted on its first 5 columns of data, simply specify the AUTOSORT option, like this:

```
OPTIONS: AUTOSORT
```

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
SORT:    REGION  EMPL-NAME  SALES-DATE
COLUMNS: REGION  EMPL-NAME  SALES-DATE  SALES-TIME  CUSTOMER  AMOUNT  TAX
```

Result in this Microsoft Works Spreadsheet:



	A	B	C	D	E	F	G
1		EMPL	SALES	SALES			
2	REGION	NAME	DATE	TIME	CUSTOMER	AMOUNT	TAX
3							
4	EAST	MORRISON	3/29/95	15:30:22	STAR MARKET	44.35	2.66
5	EAST	MORRISON	3/30/95	19:05:41	A1 PHOTOGRAPHY	29.65	1.78
6	EAST	SIMPSON	4/1/95	8:17:57	EUROPEAN DELI	14.99	0.9
7	EAST	SIMPSON	4/30/95	15:30:21	J & S LUMBER	23.87	1.43
8	NORTH	JOHNSON	4/1/95	17:02:47	VILLA HOTEL	234.45	14.07
9	NORTH	JOHNSON	4/5/95	14:33:10	MARYS ANTIQUES	9.98	0.6
10	NORTH	JONES	4/15/95	7:58:32	EZ GROCERY	10.25	0.62
11	NORTH	JONES	4/15/95	13:52:41	TOY TOWN	10.25	0.62
12	NORTH	JONES	4/15/95	8:01:59	TOY TOWN	121.76	7.31
13	SOUTH	JOHNSON	3/12/95	10:25:00	ACE ELECTRICAL	101.38	6.09
14	SOUTH	JOHNSON	4/16/95	11:48:33	ACME BUILDING	500	30
15	WEST	BAKER	3/26/95	12:09:09	JACKS CAFE	137	8.22
16	WEST	BAKER	4/12/95	14:31:12	JACKS CAFE	135.75	8.15
17	WEST	THOMAS	4/14/95	15:41:38	YOGURT CITY	9.98	0.6
18							
19							

Figure 28. Using a SORT statement to specify the sort order of a PC file

Summary

Here is a summary of what we learned in this lesson:

- use the SORT statement to **sort your PC files**
- you can sort on **multiple sort fields**
- you can sort in either **ascending or descending** order

The next lesson will teach you how to create control breaks and include subtotals in your PC file.

To Learn More

There are some additional features associated with the SORT statement which we have not covered in this lesson. Some of these additional features are discussed as topics in Chapter 4, "Beyond the Basics." Some additional features include:

- creating a **control break** with the SORT statement (page 177)
- requesting **totals and statistics** with the SORT statement (page 186)

The complete syntax for the SORT statement is given in Chapter 10, "Control Statement Syntax" (page 595).

Lesson 5. How to Create Control Breaks

This lesson teaches you what control breaks are, and shows how to request them for your PC file. This lesson also shows how to include subtotals and other statistics in your PC file. The control statement discussed is:

- the BREAK statement

How to Use the BREAK Statement

If you are not a programmer the term "control break" may be new to you. But it is a very simple concept. And as you will see, control breaks can make your PC files much more useful.

Consider the result of sorting a PC file on some field. By sorting on a field, we *group together* all the rows that contain a particular value for that field. For example, in the spreadsheet on page 106 we sorted first of all on the REGION field. As you can see, this caused the spreadsheet rows to be grouped together by region. All of the rows for the East region appear together at the beginning of the spreadsheet. Next come all of the rows for the North region, and so on. By sorting on the REGION field, we grouped together all of the rows for each region.

Often it is desirable to perform special processing whenever one such group of rows ends and another group is about to begin. For example, you might want to have a row of totals for the group that just ended. You might also want a few blank rows after the totals to separate the different groups. Such processing is called **control break processing**. A **control break** is said to occur whenever one group of rows ends and another group is about to begin. The field that is being grouped (for example, REGION) is called the **control break field** (or often just the **break field**). A control break field *must* also be a sort field, since it is by being sorted that rows are grouped together in the first place.

You may designate any sort field as a control break field. Just name the field in a BREAK statement:

```
SORT:  REGION  EMPL-NAME  SALES-DATE
BREAK: REGION
```

The above statement makes REGION a control break field. Now we will get REGION totals in the resulting spreadsheet whenever one region ends and another region is about to begin.

Figure 29 shows an Excel spreadsheet obtained by using the BREAK statement above to produce a control break.

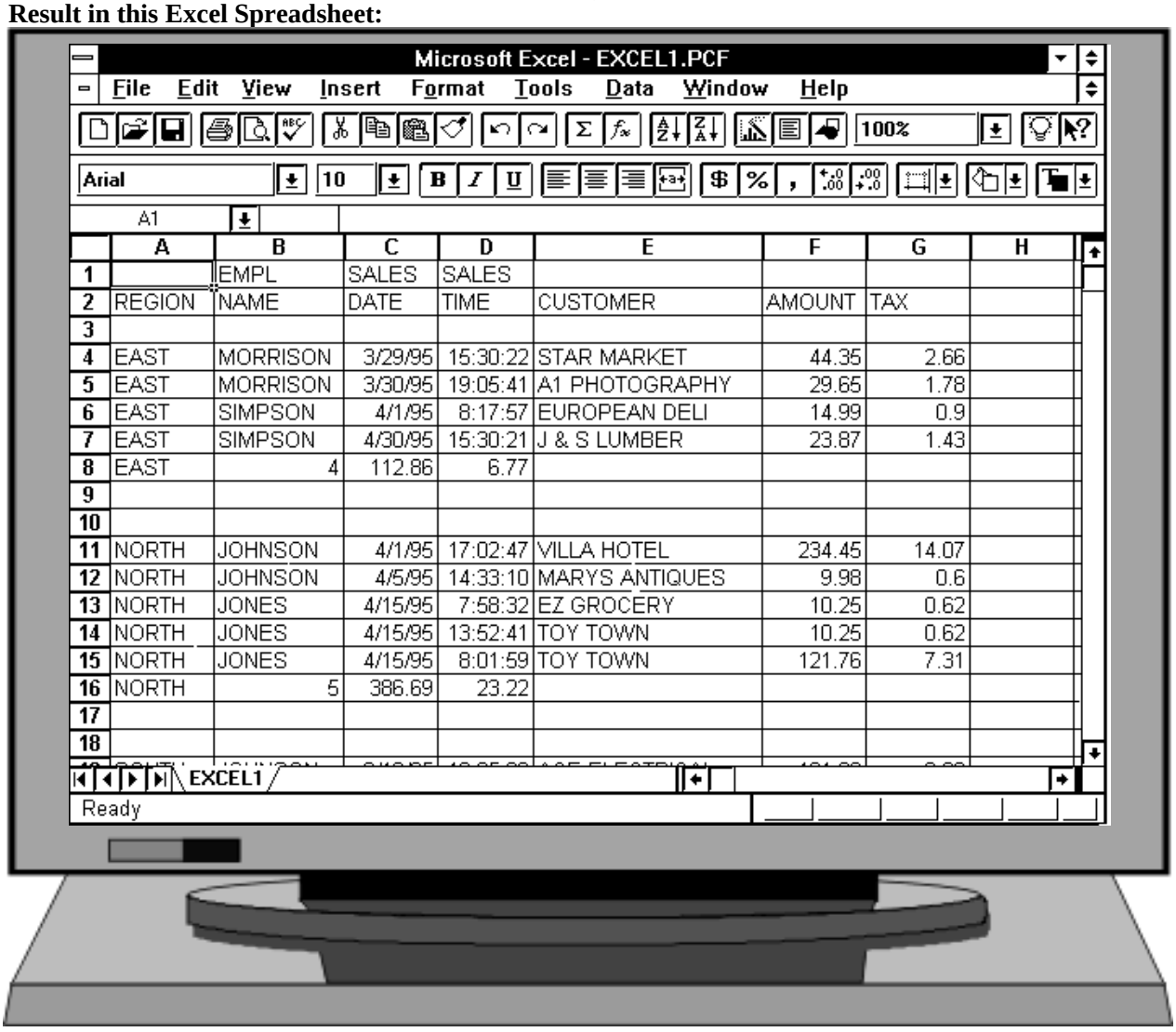
Customizing the Control Break

The Excel spreadsheet in **Figure 29** shows what Spectrum Writer's default total row looks like. It begins with the value of the break field just ended (the REGION field, in this example). The next column contains the number of items in the control group just ended. (For example, there are 4 items in the control group for the East region.) Following this are

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
SORT:    REGION  EMPL-NAME  SALES-DATE
BREAK:   REGION
COLUMNS: REGION  EMPL-NAME  SALES-DATE  SALES-TIME  CUSTOMER  AMOUNT  TAX
```

Result in this Excel Spreadsheet:



	A	B	C	D	E	F	G	H
1		EMPL	SALES	SALES				
2	REGION	NAME	DATE	TIME	CUSTOMER	AMOUNT	TAX	
3								
4	EAST	MORRISON	3/29/95	15:30:22	STAR MARKET	44.35	2.66	
5	EAST	MORRISON	3/30/95	19:05:41	A1 PHOTOGRAPHY	29.65	1.78	
6	EAST	SIMPSON	4/1/95	8:17:57	EUROPEAN DELI	14.99	0.9	
7	EAST	SIMPSON	4/30/95	15:30:21	J & S LUMBER	23.87	1.43	
8	EAST	4	112.86	6.77				
9								
10								
11	NORTH	JOHNSON	4/1/95	17:02:47	VILLA HOTEL	234.45	14.07	
12	NORTH	JOHNSON	4/5/95	14:33:10	MARYS ANTIQUES	9.98	0.6	
13	NORTH	JONES	4/15/95	7:58:32	EZ GROCERY	10.25	0.62	
14	NORTH	JONES	4/15/95	13:52:41	TOY TOWN	10.25	0.62	
15	NORTH	JONES	4/15/95	8:01:59	TOY TOWN	121.76	7.31	
16	NORTH	5	386.69	23.22				
17								
18								

Figure 29. Using the BREAK statement to create a control break with subtotals in a PC file

columns containing the total values for each numeric column in the spreadsheet (the AMOUNT and TAX fields, in this example).

The most common use of total rows is in "summary files" where the detail rows are suppressed, leaving just the total rows (see next lesson). Therefore, this default total row is designed to contain just the significant information for a control group. It does not contain any empty columns. If you are producing a spreadsheet that contains both detail rows and total rows, however, you may want to insert some blank columns in the total row. That lets you put the numeric totals in the same spreadsheet column as the corresponding detail values.

You can customize the total line at a control break by using the FOOTING parm in the BREAK statement. Consider this BREAK statement:

```
BREAK: REGION  NOTOTALS
       FOOTING(REGION ' '  ' '  ' '  ' '  AMOUNT(TOTAL)  TAX(TOTAL))
```

The above statement does two new things:

- the NOTOTALS parm suppresses Spectrum Writer's default total row at the control break
- the FOOTING parm describes a custom row to replace the default total row at each control break

The FOOTING parm works very much like the COLUMNS statement. You remember that the COLUMNS statement tells Spectrum Writer which columns are wanted in the *detail rows*. The FOOTING parm tell Spectrum Writer what columns are wanted in the *control break row*. The FOOTING parm above specifies that the contents of the REGION field should go in the first column. Then there will be four blank columns. (Each ' ' is a blank literal which results in a column that just contains a blank.) After the blank columns, the FOOTING parm specifies a column containing the total value of the AMOUNT field. And the last column contains the total value of the TAX field. By inserting four blank columns, the total AMOUNT and total TAX values line up with the detail rows. You can have as many FOOTING parms as you want in a BREAK statement. Each FOOTING parm describes one row to insert into the PC file at the control break.

You can also control the number of blank rows that appear at control breaks. By default, Spectrum Writer puts two blank rows after the total row at a control break (see page 109). Use the SPACE parm in your BREAK statement to request a different number of blank lines. For example:

```
BREAK: REGION SPACE(1)
```

The above statement requests just one blank row at the REGION control break. You may also specify SPACE(0) if you want *no* blank rows in your spreadsheet.

Figure 30 uses the FOOTING, NOTOTALS and SPACE parms to customize the control break.

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
SORT:    REGION  EMPL-NAME  SALES-DATE
BREAK:   REGION  NOTOTALS
         SPACE(1)
         FOOTING(REGION ' '  ' '  ' '  ' '  AMOUNT(TOTAL)  TAX(TOTAL))
COLUMNS: REGION  EMPL-NAME  SALES-DATE  SALES-TIME  CUSTOMER  AMOUNT  TAX
```

Result in this Excel Spreadsheet:

Microsoft Excel - CUSTBRK.TXT

File Edit View Insert Format Tools Data Window Help

	A	B	C	D	E	F	G
1		EMPL	SALES	SALES			
2	REGION	NAME	DATE	TIME	CUSTOMER	AMOUNT	TAX
3							
4	EAST	MORRISON	3/29/95	15:30:22	STAR MARKET	44.35	2.66
5	EAST	MORRISON	3/30/95	19:05:41	A1 PHOTOGRAPHY	29.65	1.78
6	EAST	SIMPSON	4/1/95	8:17:57	EUROPEAN DELI	14.99	0.9
7	EAST	SIMPSON	4/30/95	15:30:21	J & S LUMBER	23.87	1.43
8	EAST					112.86	6.77
9							
10	NORTH	JOHNSON	4/1/95	17:02:47	VILLA HOTEL	234.45	14.07
11	NORTH	JOHNSON	4/5/95	14:33:10	MARYS ANTIQUES	9.98	0.6
12	NORTH	JONES	4/15/95	7:58:32	EZ GROCERY	10.25	0.62
13	NORTH	JONES	4/15/95	13:52:41	TOY TOWN	10.25	0.62
14	NORTH	JONES	4/15/95	8:01:59	TOY TOWN	121.76	7.31
15	NORTH					386.69	23.22
16							
17	SOUTH	JOHNSON	3/12/95	10:25:00	ACE ELECTRICAL	101.38	6.09
18	SOUTH	JOHNSON	4/16/95	11:48:33	ACME BUILDING	500	30

CUSTBRK

Ready

Figure 30. Using FOOTING parms to customize the total row and create blank rows

Summary

Here is a summary of what we learned in this lesson:

- use the BREAK statement to specify a **control break field**
- control break fields must also be **sort fields**
- use the FOOTING parm to **customize the total row** at a control break
- use the SPACE parm to specify the number of **blank rows** at a control break

The next lesson will show you how to turn PC files with control breaks into "summary files."

To Learn More

There are some additional features associated with the BREAK statement which we have not covered in this lesson. Some of these additional features are discussed as topics in Chapter 4, "Beyond the Basics." Some additional topics include:

- how to write one or more customized rows at the **beginning of a control break** (page 200)
- how to **customize the total row**, and the other statistical rows (page 182 and page 186)
- how to **suppress the total row** at a control break (page 185)
- how to show various **statistics** at control breaks (page 193)
- how to compute **percentages and ratios** that apply to an entire control group (page 202)
- how to have **multiple levels** of control breaks (page 204)

The complete syntax for the BREAK statement is given Chapter 10, "Control Statement Syntax" (page 481).

Lesson 6. How to Create Summary Files

This lesson teaches you how to produce summary files. The control statement discussed is:

- the OPTIONS statement

How to Create a Summary File

Sometimes you only need summarized data in your PC— not the detail data for each individual record. Why download the entire mainframe file and summarize the data on your PC. Instead, let Spectrum Writer perform the summarization for you on the mainframe. Then just download the small summary file to your PC.

Summarizing a mainframe file with Spectrum Writer is very easy.

For example, consider the Excel spreadsheet we created back on page 109. It is a detail spreadsheet that lists every sale made in every region. The control break on REGION causes a total row to appear after the detail rows for each region.

Now let's say we only want to download the total sales amount and tax amount for each region rather than the amounts for each individual sale. To do that, we need to summarize the file by region.

By adding the following statement, we can suppress the detail rows and retain just the region totals:

```
OPTIONS: SUMMARY
```

Figure 31 shows a Paradox table obtained by using the above statement. As you can see, the table has just four rows of actual data — one for each region in the mainframe file. The first column in each row contains a region name. The second column shows the number of records that were summarized in order to create that region's total. The last two columns are the total sales amount and tax amount for the sales in that region.

Using Spectrum Writer's summarization feature can be a tremendous benefit when working with very large mainframe files (perhaps containing millions of records). The summarization is done at mainframe speed, and you end up with a much smaller PC file to download to your PC.

These Control Statements:

```
OPTIONS: PC SUMMARY NOCOLHDG
INPUT:   SALES-FILE
SORT:    REGION  EMPL-NAME  SALES-DATE
BREAK:   REGION
COLUMNS: REGION  EMPL-NAME  SALES-DATE  SALES-TIME  CUSTOMER  AMOUNT  TAX
```

Result in this Paradox Table:

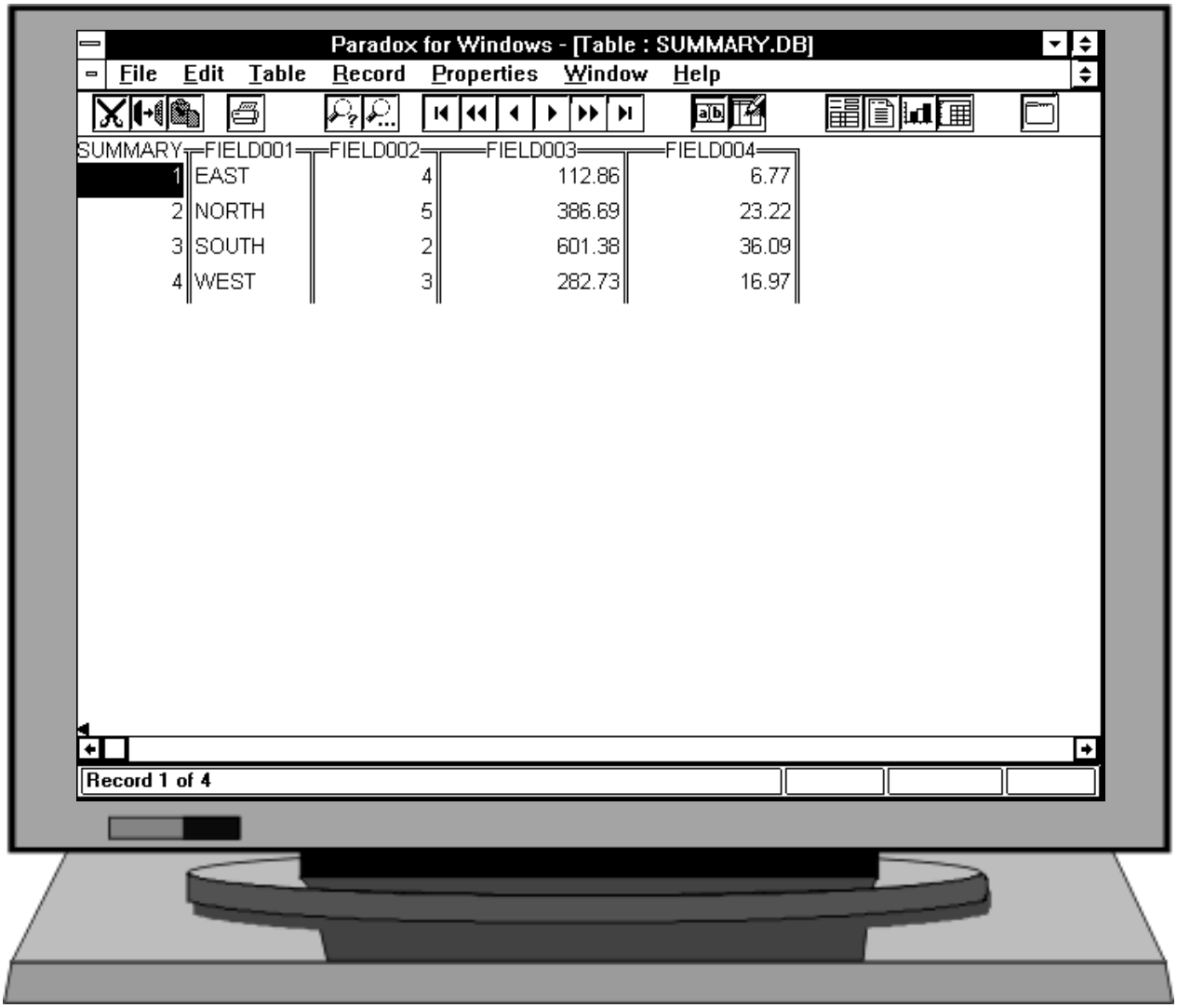


Figure 31. A Paradox table containing only summary data

Summary

Here is a summary of what we learned in this lesson:

- use the SUMMARY option (in the OPTIONS statement) to **create a summary file**
- a summary run **must have at least one control break field**

The next lesson will show you how to use data from more than one input file in a PC file.

To Learn More

There are some additional features associated with summary files which we have not covered in this lesson. Some of these additional features are discussed as topics in Chapter 4, "Beyond the Basics." Examples of additional features include:

- **customizing** the summary rows in your PC file (page 182)
- **obtaining statistics** (such as averages, maximums and minimums) in your summary file (page 186)
- creating **multiple levels** of summarization (page 204)
- including a **limited number of detail records** in each control group, creating spreadsheets such as "The Top 3 Sales in Each Region" (page 212)

Lesson 7. How to Use Data from More Than One File

This lesson teaches you how to read records from additional input files for use in your PC file. The control statement discussed is:

- the READ statement

All of the sample PC files produced so far have used data from only one input file. The data has come from the file specified in the INPUT statement, called the **primary input file**. There are times when all of the data needed for a particular PC file will not be found in just a single file. One of Spectrum Writer's most powerful features is its ability to link to *any number* of additional files to produce a PC file.

How Auxiliary Input Files Are Processed

Each PC file is allowed to have only one primary input file, specified in the INPUT statement. When data from additional input files is required, a READ statement is used. The READ statement causes a record to be read from another input file, called an **auxiliary input file**. You may have as many READ statements as you like in a single run.

By simply adding a READ statement to your request, you automatically make all of the fields from another whole file available for use in producing your PC file.

Here is how Spectrum Writer processes the primary and auxiliary input files. Spectrum Writer first reads a single record from the primary input file. (This file is always read *sequentially*, beginning with the first record in the file.) Next, if any auxiliary input files were specified, Spectrum Writer also reads one record from each of those files. (These files are always read *randomly*, using a key.) At this point, Spectrum Writer will have read one record from each of the input files. The fields from all of these records are now available for use in producing the PC file. These fields can be used:

- as columns of data
- as sort fields
- as control break fields
- in conditional expressions
- in calculations
- and in any other way that fields from the primary input file are used

After processing this set of records, Spectrum Writer then repeats the process. Another record is read sequentially from the primary input file. Then random reads are performed to each of the auxiliary input files. This next group of records is then used in making the PC file, and so on. This process is repeated until there are no more records left in the primary input file.

By simply adding a READ statement to your request, you automatically make all of the data fields from another file available for use in producing your PC file.

There is one important thing about auxiliary input files to keep in mind. Since these files are ready randomly, they *must* be keyed files or DB2 tables. (VSAM files are often keyed files.)

In a keyed file, each record has a unique "key" value associated with it. When a random read is made to such a file, a **read key** must be specified to identify which record to read. What read key should you tell Spectrum Writer to use when reading a record from an auxiliary input file? In order to be useful, the auxiliary input record should be somehow related to the primary input record. Usually, the record from the primary input file will contain the key of a corresponding record in the auxiliary input file. That key from the primary input file will be used as the read key.

> **Note:** If you are not familiar with such terms as "keyed files" and "read keys," ask your programmer to help you determine whether a particular file is keyed or not, and also to help you decide what read key to use.

How to Use the READ Statement

Now let's look at a concrete example of how to use the READ statement. Begin by considering **Figure 32**, which shows a spreadsheet that uses only a primary input file (the SALES–FILE). This spreadsheet shows information about each sale made by an employee. This spreadsheet includes columns for two fields that we haven't used in previous examples, so we'll explain them. They are the EMPL–NUM field and the PRODUCT–CODE field. The EMPL–NUM is the employee number of the employee who made the sale. The PRODUCT–CODE is a code that identifies the product that was sold to the customer.

Now, let's assume that we want this spreadsheet to also show each employee's social security number. The social security number is not available in the SALES–FILE. But it is a field in the EMPL–FILE. (See page 91.) In order to produce such a spreadsheet, we need data from a second input file — the EMPL–FILE.

The EMPL–FILE is a keyed VSAM file. Its key is the 3–byte employee number. The records in the SALES–FILE also contain an employee number, so we can use that field as the "read key" to use when we read the EMPL–FILE. That means we can make the EMPL–FILE an auxiliary input file by simply adding this statement:

```
READ:  EMPL-FILE   READKEY(EMPL-NUM)
```

This READ statement tells Spectrum Writer to use the EMPL–NUM field from the records in the SALES–FILE as a key to read an auxiliary record from the EMPL–FILE. All control statements after this READ statement may now refer to the fields in the EMPL–FILE, as well as to those in the SALES–FILE. So, we can now add the SOCIAL–SEC–NUM field from the EMPL–FILE to our COLUMNS statement:

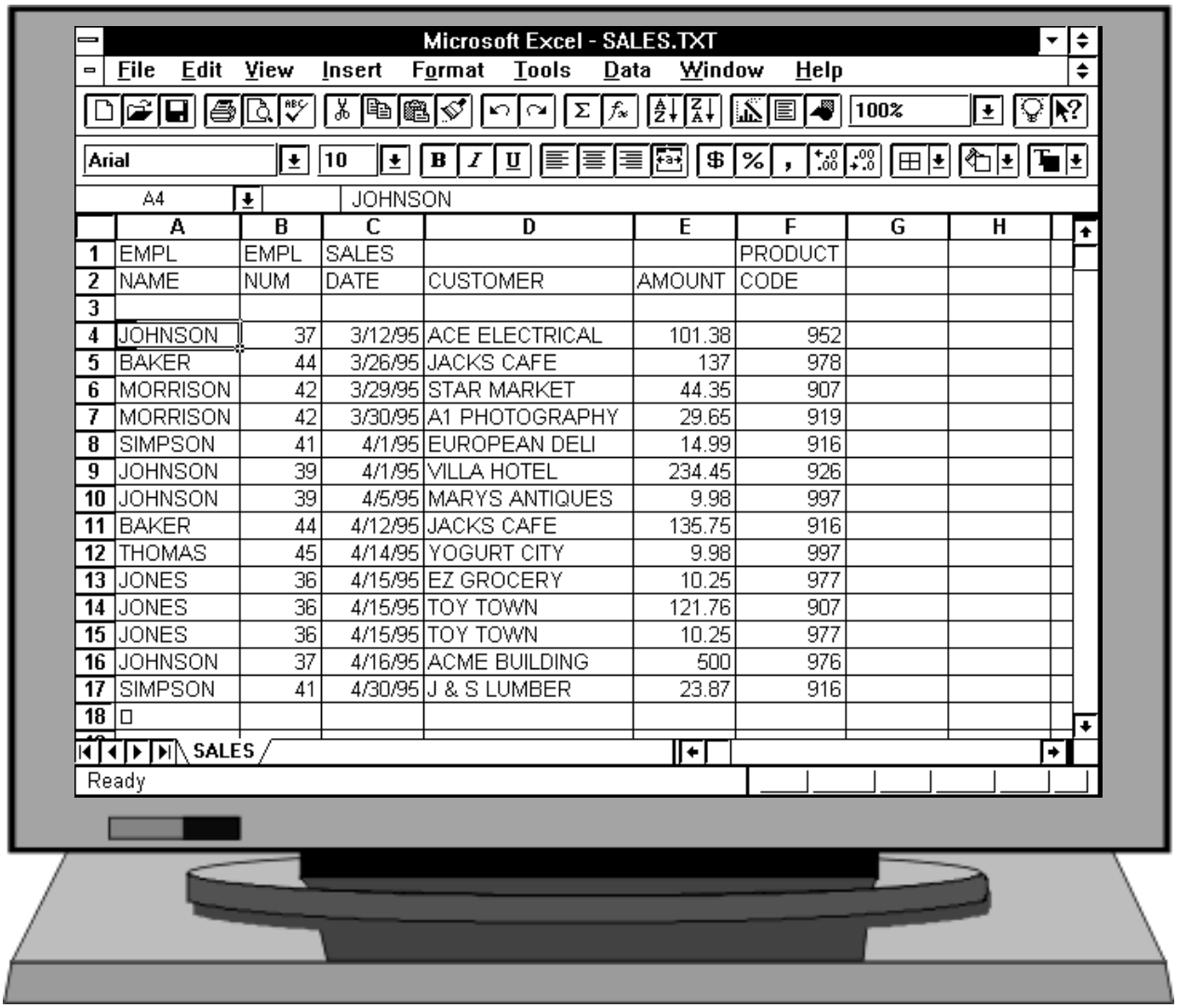
```
COLUMNS: EMPL-NAME  SALES-FILE.EMPL-NUM  SOCIAL-SEC-NUM
         SALES-DATE  CUSTOMER  AMOUNT  PRODUCT-CODE
```

Notice that in the above COLUMNS statement we must now prefix the EMPL–NUM field with a record name (like this: SALES–FILE.EMPL–NUM). This is because after the READ statement, EMPL–NUM is no longer a unique field name. A field by that name exists in both the SALES–FILE and the EMPL–FILE. (See Appendix F, "Files Used in Examples" on page 648.) Since the EMPL–NUM will have the same value in both of the records, it doesn't really matter which one we specify in the COLUMNS statement, but we do have to specify a unique name.

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
COLUMNS: EMPL-NAME  EMPL-NUM  SALES-DATE  CUSTOMER  AMOUNT  PRODUCT-CODE
```

Result in this Excel Spreadsheet:



	A	B	C	D	E	F	G	H
1	EMPL	EMPL	SALES			PRODUCT		
2	NAME	NUM	DATE	CUSTOMER	AMOUNT	CODE		
3								
4	JOHNSON	37	3/12/95	ACE ELECTRICAL	101.38	952		
5	BAKER	44	3/26/95	JACKS CAFE	137	978		
6	MORRISON	42	3/29/95	STAR MARKET	44.35	907		
7	MORRISON	42	3/30/95	A1 PHOTOGRAPHY	29.65	919		
8	SIMPSON	41	4/1/95	EUROPEAN DELI	14.99	916		
9	JOHNSON	39	4/1/95	VILLA HOTEL	234.45	926		
10	JOHNSON	39	4/5/95	MARYS ANTIQUES	9.98	997		
11	BAKER	44	4/12/95	JACKS CAFE	135.75	916		
12	THOMAS	45	4/14/95	YOGURT CITY	9.98	997		
13	JONES	36	4/15/95	EZ GROCERY	10.25	977		
14	JONES	36	4/15/95	TOY TOWN	121.76	907		
15	JONES	36	4/15/95	TOY TOWN	10.25	977		
16	JOHNSON	37	4/16/95	ACME BUILDING	500	976		
17	SIMPSON	41	4/30/95	J & S LUMBER	23.87	916		
18								

Figure 32. A spreadsheet that uses only the primary input file

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
READ:    EMPL-FILE  READKEY(EMPL-NUM)
SORT:    SOCIAL-SEC-NUM
COLUMNS: EMPL-NAME  SALES-FILE.EMPL-NUM  SOCIAL-SEC-NUM
         SALES-DATE  CUSTOMER  AMOUNT  PRODUCT-CODE
```

Result in this Excel spreadsheet:

Microsoft Excel - SALEEMPL.XLS

File Edit View Insert Format Tools Data Window Help

Arial 10 100%

A6 JOHNSON

	A	B	C	D	E	F	G	H
1		SALES						
2		FILE	SOCIAL					
3	EMPL	EMPL	SEC	SALES			PRODUCT	
4	NAME	NUM	NUM	DATE	CUSTOMER	AMOUNT	CODE	
5								
6	JOHNSON	39	4779981	4/5/95	MARYS ANTIQUES	9.98	997	
7	JOHNSON	39	4779981	4/1/95	VILLA HOTEL	234.45	926	
8	JONES	36	12098765	4/15/95	EZ GROCERY	10.25	977	
9	JONES	36	12098765	4/15/95	TOY TOWN	10.25	977	
10	JONES	36	12098765	4/15/95	TOY TOWN	121.76	907	
11	SIMPSON	41	112050456	4/30/95	J & S LUMBER	23.87	916	
12	SIMPSON	41	112050456	4/1/95	EUROPEAN DELI	14.99	916	
13	THOMAS	45	776838221	4/14/95	YOGURT CITY	9.98	997	
14	BAKER	44	878190156	4/12/95	JACKS CAFE	135.75	916	
15	BAKER	44	878190156	3/26/95	JACKS CAFE	137	978	
16	MORRISON	42	900120556	3/30/95	A1 PHOTOGRAPHY	29.65	919	
17	MORRISON	42	900120556	3/29/95	STAR MARKET	44.35	907	
18	JOHNSON	37	912040334	3/12/95	ACE ELECTRICAL	101.38	952	

DOCA313

Ready

Figure 33. A spreadsheet that uses a READ statement to specify an auxiliary input file

In this case we specified the EMPL–NUM field from the SALES–FILE. (For more information on using "record names" to qualify field names, see "How to Name the Input File Records" on page 228.)

Figure 33 (page 119) shows an Excel spreadsheet obtained by using the above statements. The spreadsheet now has the desired new column showing each employee's social security number. Notice that we also *sorted* the PC file on SOCIAL–SEC–NUM. Remember that you can use fields from auxiliary input files in any way that you can use fields from the primary input file.

"One-to-Many" Random Reads

Normally, Spectrum Writer reads just a single record from your auxiliary input file — the record whose key field matches the READKEY value. If you want to use *all of the records* which match the READKEY (or partial READKEY), add the MULTI parm to your READ statement. When the READ statement has the MULTI parm, Spectrum Writer creates and processes "logical input records" by matching the primary input file row with *each* qualifying record from the auxiliary input file. For more information on how the MULTI parm works, see "How to Perform "One–to–Many" Reads" on page 232.

How to Use Multiple READ Statements

You may use as many READ statements as you like in a run. For example, the Excel spreadsheet in **Figure 34** uses two READ statements.

The primary input file is once again the SALES–FILE, which contains one record for each sale made by an employee. It is specified in the INPUT statement:

```
INPUT: SALES-FILE
```

To obtain additional data about the employee who made each sale, we use a READ statement for the EMPL–FILE (just like in the preceding example). The EMPL–NUM field in the SALES–FILE contains the key necessary to read the correct EMPL–FILE record.

```
READ:  EMPL-FILE  READKEY(EMPL-NUM)
```

To obtain additional information about each *product* sold, a second READ statement names the PRODUCT–FILE as another auxiliary input file. (The PRODUCT–FILE is also described in Appendix F, "Files Used in Examples" on page 648.)

However, there is one minor complication in reading records from this file. The key in the PRODUCT–FILE records is 4 bytes long. It consists of the letter "P" followed by a 3–byte product code. The SALES–FILE does not contain a field which can be used *directly* as the read key to the PRODUCT–FILE. But it does contain the 3–byte PRODUCT–CODE field, which we can use to build the 4–byte read key. A COMPUTE statement is therefore used to create a new field (called PKEY) which consists of the letter "P" followed by the product code. This computed field is then used as the read key in the READ statement for the PRODUCT–FILE:

```
COMPUTE: PKEY = 'P' + PRODUCT-CODE
READ:    PRODUCT-FILE READKEY(PKEY)
```

By having two READ statements in addition to the INPUT statement, the PC file now uses data from three input files. Data from all of these files can be used in any of the subsequent

These Control Statements:

```
OPTIONS: PC
INPUT:   SALES-FILE
READ:    EMPL-FILE     READKEY(EMPL-NUM)
COMPUTE: PKEY = 'P' +  PRODUCT-CODE
READ:    PRODUCT-FILE  READKEY(PKEY)
SORT:    SOCIAL-SEC-NUM
COLUMNS: EMPL-NAME  SALES-FILE.EMPL-NUM  SOCIAL-SEC-NUM
         SALES-DATE  CUSTOMER  AMOUNT
         PRODUCT-CODE  PRODUCT-DESC
```

Result in this Excel Spreadsheet:

Microsoft Excel - PRODUCT.XLS

File Edit View Insert Format Tools Data Window Help

Arial 10 100%

H6 MAILING LABELS

	A	B	C	D	E	F	G	H
1		SALES						
2		FILE	SOCIAL					
3	EMPL	EMPL	SEC	SALES			PRODUCT	PRODUCT
4	NAME	NUM	NUM	DATE	CUSTOMER	AMOUNT	CODE	DESC
5								
6	JOHNSON	39	4779981	4/5/95	MARYS ANTIQU	9.98	997	MAILING LABELS
7	JOHNSON	39	4779981	4/1/95	VILLA HOTEL	234.45	926	DESK CALENDARS
8	JONES	36	12098765	4/15/95	EZ GROCERY	10.25	977	PAPER CLIPS
9	JONES	36	12098765	4/15/95	TOY TOWN	10.25	977	PAPER CLIPS
10	JONES	36	12098765	4/15/95	TOY TOWN	121.76	907	INKPADS
11	SIMPSON	41	112050456	4/30/95	J & S LUMBER	23.87	916	RED PENS
12	SIMPSON	41	112050456	4/1/95	EUROPEAN DEL	14.99	916	RED PENS
13	THOMAS	45	776838221	4/14/95	YOGURT CITY	9.98	997	MAILING LABELS
14	BAKER	44	878190156	4/12/95	JACKS CAFE	135.75	916	RED PENS
15	BAKER	44	878190156	3/26/95	JACKS CAFE	137	978	HOLE PUNCHERS
16	MORRISO	42	900120556	3/30/95	A1 PHOTOGRAF	29.65	919	GREEN PENS
17	MORRISO	42	900120556	3/29/95	STAR MARKET	44.35	907	INKPADS
18	JOHNSON	37	912040334	3/12/95	ACE ELECTRICA	101.38	952	PENCILS (NO. 1)

DOCA314

Ready

Figure 34. A spreadsheet that uses two READ statements to specify two auxiliary input files

control statements. In the Excel spreadsheet in **Figure 34**, the COLUMNS statement uses one field from each of the auxiliary input files. It uses the SOCIAL–SEC–NUM field from the EMPL–FILE and the PRODUCT–DESC field from the PRODUCT–FILE:

```
COLUMNS: EMPL-NAME
         SALES-FILE.EMPL-NUM
         SOCIAL-SEC-NUM
         SALES-DATE
         CUSTOMER
         PRODUCT-CODE
         PRODUCT-DESC
```

Summary

Here is a summary of what we learned in this lesson:

- the READ statement is used to read records from **auxiliary input files**
- the file named in a READ statement must be a **keyed file** (or a DB2 table)
- you may have as **many READ statements as you like** in a single run

To Learn More

There are some additional features associated with the READ statement which we have not covered in this lesson. Some of these additional features are discussed as topics in Chapter 4, "Beyond the Basics." Some additional features include:

- how to assign a **record name** to the records read from auxiliary input files (page 228)
- how to read multiple records (containing **different keys**) from the same auxiliary input file (page 224)
- how to use **data from one auxiliary input file as the read key** to another auxiliary input file (page 226)
- how to specify **generic keys** and **"KGE" keys** in the READ statement (page 230)
- how to read multiple records (with the **same key or partial key**) from the auxiliary input file (page 232)
- what happens when **no record is found** for a particular read key (page 229)
- how to determine whether the read for a particular key was **successful** or not (page 230)
- how to use the READ statement to obtain data from a **DB2 table or view** (page 397)
- how to use the ONIOERROR option to **increase the severity** of I/O errors on an input file (page 586)

The complete **syntax** for the READ statement, as well as a more detailed **narrative** of how Spectrum Writer assembles input records during the report process, is given in Chapter 10, "Control Statement Syntax" (page 578).