R S I

ORACLE

REPORT WRITER/TEXT FORMATTER

USER'S GUIDE

Oracle Users Guide - Version 2.3

"ORACLE"

REPORT-WRITER

USER'S GUIDE

TABLE OF CONTENTS

R E P O R T - W R I T E R

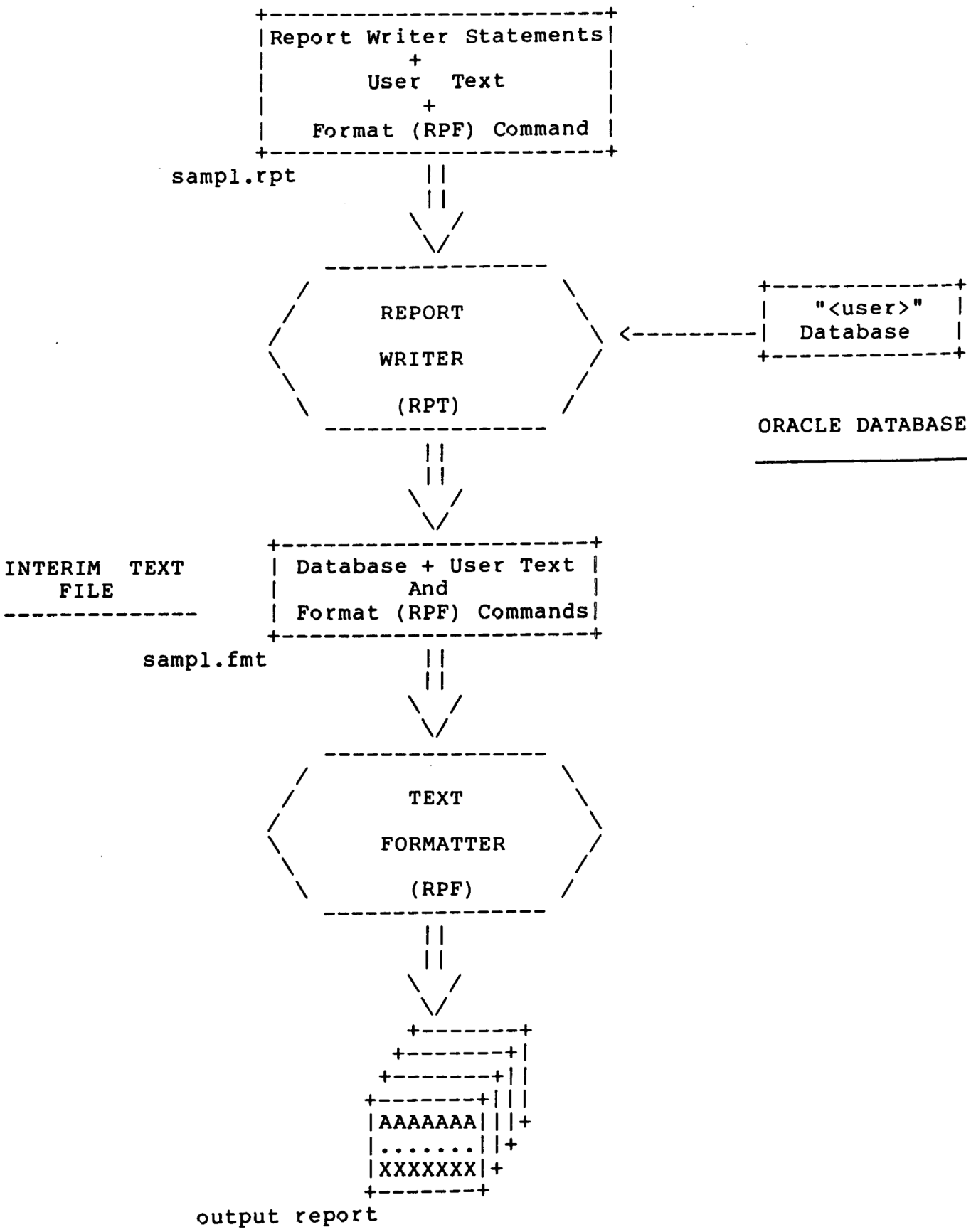
O V E R V I E W

The ORACLE **Report Writer** provides SQL's query capabilities, in addition to text formatting capabilities, to enable ORACLE to produce reports that combine information derived from the database with additional textual information such as headings, explanatory text and any other material desired. Potential applications for the Report Writer are quite varied. In the simplest case, it can be used to produce a report derived from a single ORACLE table, with column headings, columns of database information, and totals as desired. Examples of more complex applications are the production of reports with many levels of nesting, with multiple breaks in columns, and with a variety of subtotals and totals presented. Other applications that emphasize the text processing capabilities could include the production of computer-generated correspondence, with name and address information, as well as information included in the correspondence derived from the database, and the production of periodic budget and cost reports used in the management of a business. Of course, the Report Writer can also be used to print on preprinted forms.

The Report Writer is comprised of two programs that must both be used to derive information from the database and produce a report which presents the derived information in the desired fashion, as illustrated by Figure 1.1. The first of these, RPT, derives database information through SQL statements. The second, RPF, is a text formatter that formats the information based on format commands included in the text.

Generation of a report is controlled by a single file, the report control file, which contains report writer statements (for RPT), text formatter commands (for RPF) and any other text material that is to be included in the report.

This file can be created using any standard text editor. Once the report control file is complete, RPT is executed. RPT reads the report control file, scanning for report writer statements. Text formatter commands and user text are merged with the database-derived information that is produced by the RPT run.



" Report Generation Process "

Figure 1.1

Report Writer statements cause RPT to open the user's ORACLE database to derive information for the report. Incorporated within a report program are SQL queries to derive the desired data. Other statements cause RPT to include report heading and footing information, use specific data output formats, and conditionally branch to and execute other SQL or RPT statements. RPT can be directed to intersperse the database information within the RPF commands and user supplied text.

When RPT processing is completed, the interim file it produces is processed by the Report Formatter (RPF) to generate the desired report. Before processing by RPF, at the end of the RPT run, the interim file contains text supplied by the user, information extracted from the database and RPF commands specifying how that information should be placed on the report. RPF commands can specify:

- horizontal and vertical margins
- centering and underlining
- tabulation
- page numbering
- spacing and actual placement of text

RPF output can be directed to a line printer, typewriter terminal, or CRT. Figures 1.2, 1.3, and 1.4 are examples of Report Writer-generated reports. A detailed explanation of these report programs is provided in Section 5.

This manual serves two purposes. It is a detailed reference manual on the RPT Report Writer and the RPF Report Formatter. It also serves as a User's Guide, informing the reader by example how the various features of these two programs interact to construct a complete report. The reader is encouraged to first become familiar with the RPF text formatting language before tackling the RPT programming statements.

T E C H N O L O G Y S Y S T E M S , I N C

PERSONNEL REPORT FOR SEPTEMBER, 1980

EMPNO	NAME	JOB	SALARY	COMMISSION	DNO
7369	SMITH	CLERK	$800.00		20
7499	ALLEN	SALESMAN	$1,600.00	$300.00	30
7521	WARD	SALESMAN	$1,250.00	$500.00	30
7566	JONES	MANAGER	$2,975.00		20
7654	MARTIN	SALESMAN	$1,250.00	$1,400.00	30
7698	BLAKE	MANAGER	$2,850.00		30
7782	CLARK	MANAGER	$2,450.00		10
7788	SCOTT	ANALYST	$3,000.00		20
7839	OATES	PRESIDENT	$5,000.00		10
7844	TURNER	SALESMAN	$1,500.00		30
7876	ADAMS	CLERK	$1,100.00		20
7900	JAMES	CLERK	$950.00		30
7902	FORD	ANALYST	$3,000.00		20
7934	MILLER	CLERK	$1,300.00		10

END OF REPORT

"Example 1 - Tabular Report "

Figure 1.2

D i v i s i o n P a y r o l l R e p o r t

by Department

****

DEPTNO: 010 - - DEPTNAME: ADMINISTRATION

****

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
7782	CLARK	$3162.50		$37,950.00
7934	MILLER	$1300.00		$15,600.00
7839	OATES	$5750.00		$69,000.00
	Department Summary			
AVG = $3,404.17	MIN = $1,300.00	MAX = $5,750.00		$122,550.00

****

DEPTNO: 020 - - DEPTNAME: RESEARCH

****

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
7369	SMITH	$800.00		$9,600.00
7566	JONES	$3421.25		$41,055.00
7876	ADAMS	$1100.00		$13,200.00
7788	SCOTT	$3000.00		$36,000.00
7902	FORD	$3000.00		$36,000.00
7955	WILSON			
7956	JAKES	$1000.00		$12,000.00
	Department Summary			

" Example 2 - Nested Report "

Figure 1.3

Date : 03/25/81

To: BLAKE

Department : 30 - SALES

Location : PARIS

From : Bill James
Director of Personnel

Subj : 1982 Employee Compensation Plan

Its budget time again! To aid you in completing the salary portion of your budget I have computed your department's current salary expenses. For planning purposes we are presently estimating an across the board increase of 10% in the 1982 salary pool.

Your department's 1981 figure is: $140,430.00

Estimated 1982 figure is: $154,473.00

In completing your salary plan you should break this total down by individual employees. Please let me know if you have any questions on this or related issues.

Bill

Example 3 - Sample Letter

Figure 1.4

U S I N G

T H E R E P O R T F O R M A T T E R

(R P F)

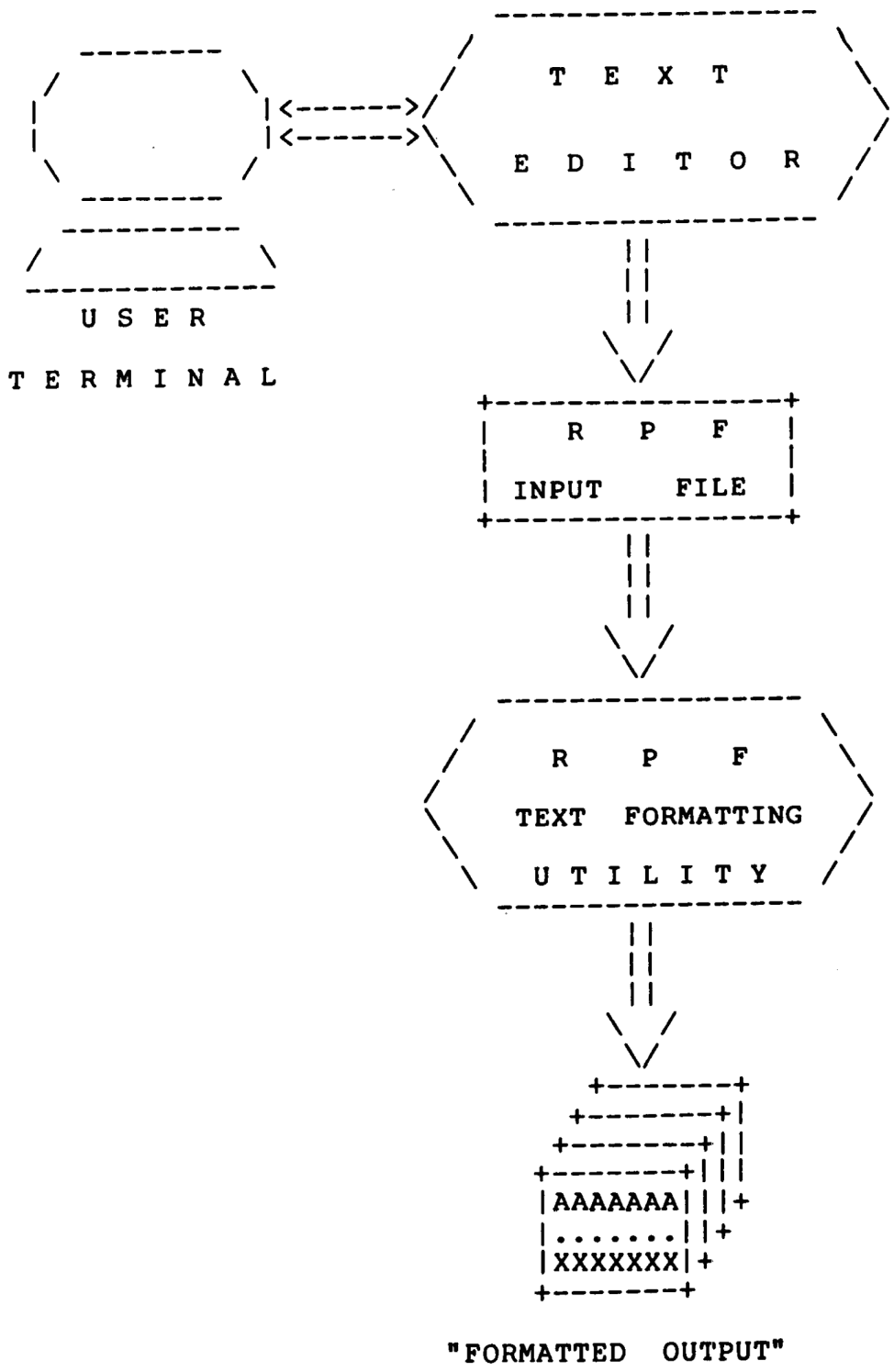
2.1 **Overview**

This section discusses methods for using RPF commands to create a complete report. First, a simple example is presented, and methods for producing it are described; then a sequence of increasingly complex examples is presented, each with its own discussion of methods for producing it.

RPF receives both the input it processes and the commands that direct the processing in the same file. Thus, commands imbedded within the file to be formatted are used to control the formatting operation. The RPF user presents commands to RPF by inserting them into the file to be formatted, usually using a text editor (of course, if the file is constructed by a program, RPF commands can also be inserted by the program). This process is illustrated by the diagram of Figure 2.1.

The imbedded commands control a variety of options, such as what margins to use, when to skip to the next page, how far to indent a paragraph, where to skip a line, how to number the pages, etc.

RPF reads an input file and formats it into output pages as directed by the imbedded commands. The text is viewed as a series of **words.** A word is a collection of one or more characters which is terminated by at least one space or an end of line character (An end of line character is inserted into the text file by the editor when the user enters a carriage return). RPF places each word, one after another into the output line. One space will separate each word on output, regardless of the number of spaces the user may have entered between the words on input.


T E X T
E D I T O R
U S E R
T E R M I N A L
R P F
INPUT FILE
R P F
TEXT FORMATTING
U T I L I T Y
"FORMATTED OUTPUT"

" Figure 2.1 "

When the line is full and the next word will not entirely fit within the current line, the last word is shifted to the right so that the last character is aligned with the right margin. A word is indivisable and is never split between lines. RPF puts spaces between words so that the right and left margins are justified. RPF will continue to place words in the output pages until the end of the input file is encountered.

RPF allows the user to define the starting and ending positions of the output line. The area on a line where text is placed is called a **column.** If the user does not specify the boundaries of a column, RPF will place the text in a default column which begins in position one and extends to position 132.

2.2 **Example 1 - A Simple Letter**

Now that the basics have been reviewed, let's look at the input file which is required to produce a simple letter. Figure 2.2A is a listing of the file which created the letter in figure 2.2B. The input file was created using a standard text editor. The line numbers and titles were added after RPF was executed.

RPF differentiates its commands from the user's text by the leading '.'(period) or '#'(pound sign). A user may begin words with these characters providing the word is not exactly like an RPF command. If it is, a second '.' or '#' will tell RPF that this is text. On output only one '.' or '#' will be printed. Commands may be specified with either upper or lower case letters.

The letter was printed with left and right margins of '13' and '73'. This was done by defining a single column beginning in position '13' and extending to position '73'. The command to do this is shown in Line 1.

The '.dt' command on Line 1 defines a table with an identifier of '1' and contains the column beginning in position '13' and extending through and including position '73'. Notice the period following the '73'. It is required to tell RPF that there are no more columns in this table.

```
 1: .dt 1 13 73 .
 2: #page 6 58
 3: .sp 1
 4: .t 1
 5: .s 3
 6: January 10, 1981
 7: .s 3
 8: Mr. William     O.       Smith .n
 9: 3752 Oakwood Drive .n Seaport, Me.  96142
10: .s 4
11: Dear Bill:
12: .s 4  .p
13: I hope you have received the medical and dental insurance forms that I se
14: .s 1
15: .p The form you submitted for your November 1, 1980 visit to Dr. George
16: Smedley did not indicate the nature of the treatment. Please have the
17: doctor write me a note describing the reason for the visit and the
18: treatment which was prescribed.
19: .b   .p
20: If you have any questions please give me a call at my office.
21: .s 2
22: Sincerely,
23: .s 4
24: Sharon Brown
25: .te
```

Example 1 - RPF Input Text

" Figure 2.2 A "

January 10, 1981

Mr. William O. Smith
3752 Oakwood Drive
Seaport, Me. 96142

Dear Bill:

I hope you have received the medical and dental insurance forms that I sent.

The form you submitted for your November 1, 1980 visit to Dr. George Smedley did not indicate the nature of the treatment. Please have the doctor write me a note describing the reason for the visit and the treatment which was prescribed.

If you have any questions please give me a call at my office.

Sincerely,

Sharon Brown

Example 1 - RPF Output

" Figure 2.2 B "

Defining a table does not automatically cause the text to be placed within the specified column boundaries. A table must be invoked before it will take effect. The 't' command in Line 4 invokes table '1'. If user text was included prior to Line 4 it would have been formatted according to the definition of the default table; a single column beginning in position '1' and extending through position '132'.

The 'page' command on Line 2 defines the top and bottom margins of a page. RPF will begin placing text on the sixth line of every page and automatically skip to the next page after completing line 58. All pages are assumed to have 66 lines. If the print spacing is six lines per inch, the 66 lines will cover a standard 11 inch page.

The 'sp' command defines the spacing between lines. Since the default is single spacing this command could have been omitted. The '.s 3' command on lines 5 & 7 will cause three lines to be skipped. Printing on stationery will require skipping enough lines to have the first line printed below the letterhead.

Line 6 contains the first line of user text. Three words are specified: 'January', '10,', and '1981'. Remember that the number of spaces between words on input has no effect on the spacing on output. These words could have been input on separate lines and still produced the same results.

Line 8 contains the name of the addressee followed by a '.n' (new line) command. This is the same as the 'return' key on a typewriter. The 'n' command on Line 9 causes the city/state to be printed on the line following the street address.

The paragraph command ('.p') on Line 12 causes the text which follows to begin on the next line and be indented five spaces. The text within the body of the letter is printed within the column boundaries. As noted above as many words as possible will be placed on a line. The word 'insurance' would not completely fit on the same line as 'dental'. The word 'dental' was aligned with the right hand margin, and the additional spaces were evenly distributed throughout that line.

The blank line ('.b') command inserts one blank line and has the same effect as '.s 1' (skip one line).

The table end ('.te') command on Line 25 instructs RPF that table '1' is no longer active. Column boundaries revert back to the default table. If the active table is not terminated an 'UNEXPECTED END OF COLUMN' message will be issued.

The sample letter was printed on a typewriter quality printer with the following system command:

```
RPF tt1:/pa=letter.abc
```

where tt1: is the system address for the desired printer. The /pa instructs RPF to pause and sound the alarm before printing each page. This will permit the user to individually load each sheet of paper. When ready, any character may be entered to start the printing. The entered character will not be printed.

'letter.abc' is the name of the input text file. If the file qualifier is omitted (in this case 'abc'), 'rpf' is assumed.

2.3 **Example 2 - A Tabular Report**

The second example is the tabular report shown in figure 2.3B. Figure 2.3A is a listing of the input text file which produced this report. The line numbers and titles in 2.3A were added after RPF was executed.

The '.dt' command on line 1 defines table '1' which contains a single column extending from position 13 to 73. A second table, table '2', is defined on line 2. This table contains 5 columns; the first extending from position 1 to 5, the second from 9 to 28, the third from 32 to 39, and so on. Notice that the fifth column is defined by the last pair of numbers; 52 and 0. The zero signifies that this column ends on the right hand margin of the column in which this table is invoked. We will see a little later how table '2' is used to create this tabular report.

```
 1: .dt 1 13 73 .
 2: .dt 2 1 5 9 28 32 39 42 49 52 0 .
 3: .page 6 58
 4: .t 1
 5: .s 4
 6: .cul " ALL SEASONS SPORTING GOODS " .
 7: .s 2
 8: .cul MONTHLY SALES REPORT .
 9: .s 4
10: .t 2
11: ITEM .n NO. .nc
12: .cen ITEM . .n .cen DESCRIPTION . .nc
13: .r PREVIOUS .n MONTH .nc
14: .r CURRENT .n MONTH .nc
15: .r Y-T-D .nc
16: .s 2
17: 2354 .nc NFL Football .nc $175.34 .nc $202.45 .nc $564.89 .nc
18: .s 1
19: 6734 .nc Chicago Cubs  Baseball Uniform .nc $56.10 .nc $162.38 .nc $287.01
20: .s 1
21: 8940 .nc Alpine Skis  .nc $941.84 .nc  $1005.93 .nc  $3582.57 .nc
22: .s 1
23: .nc .nc -------- .n $1173.28  .nc -------- .n $1307.76 .nc -------- .n
24: $4434.47
25: .te .s 3
26: .cul END \ OF \ REPORT .
27: .te
```

" Example 2 - A Tabular Report "

Figure 2.3 A

" ALL SEASONS SPORTING GOODS "

MONTHLY SALES REPORT

ITEM NO.	ITEM DESCRIPTION	PREVIOUS MONTH	CURRENT MONTH	Y-T-D
2354	NFL Football	$175.34	$202.45	$564.89
6734	Chicago Cubs Baseball Uniform	$56.10	$162.38	$287.01
8940	Alpine Skis	$941.84	$1005.93	$3582.57
		$1173.28	$1307.76	$4434.47

END OF REPORT

"Example 2 - A Tabular Report"

Figure 2.3 B

Line 3 defines the top and bottom margins of the output pages, and line 4 invokes table '1'. Since table '1' has only one column, the text that follows will be formatted within that column's boundaries. The '.cul' command on line 6 indicates that the text between the command and the terminating period should be centered within the current column and underlined. Since the current column extends from position 13 through 73, this text is centered on the page.

Figure 2.3A was printed on a special printer which allows bold face type to be substituted for underlined characters. If this option was not selected or a standard line printer was used, the specified text would have been underlined. If the output device is a video terminal (CRT) the user can specify at RPF execution time that all underlined text be displayed in reverse video. See Section 2.6 - 'Executing RPF' for additional information.

The '.cul' command on line 8 produces a second line of centered and underlined text.

At this point we are ready to create our tabular report. The report will contain five columns across the page. If this report was being typed the user would probably set tab stops for the first position of each column. With RPF a new table is invoked to establish these column boundaries. The command on line 10 invokes table '2', which was previously defined on line 2. Unlike table '1' which had only one column, table '2' provides five columns in which to output text.

When a table is invoked it subdivides the 'current' column. In this case the 'current' column is the single column of table '1'. Therefore, invoking table '2' subdivides the 61 print positions from position 13 through 73 into five separate areas or columns. The column boundaries for a table are interpreted relative to the current column. For example the first column in table '2' begins in relative position 1 and extends to relative position 5. Since the current column begins in position 13, the absolute boundaries of this column are 13 through 18. The last column begins in relative position 52 (absolute 65) and extends to the end of the current column (absolute position 73).

A table must completely fit within the column in which it is invoked. The column in which table '2' was invoked contained 61 print positions. If any columns in table '2' extended beyond relative position 61 an error would be indicated. For example, if the last column was defined as '52 65', when the command on line 10 was executed RPF would have returned the error "COLUMN TOO SMALL".

Now that table '2' has been defined and invoked, output text may be placed in each of the five columns. The text on line 11 is placed within the first column. The '.n' command causes the text 'NO.' to be placed on a new line within this column. The '.nc' command causes the placement of text in the current column to stop. The text which follows will be place on the first line of the next column in table '2'.

Line 12 requests that the words 'ITEM' and 'DESCRIPTION' be centered on the first two lines of the second column. The '.r' (right justify) command on lines 13, 14, and 15 tells RPF that all subsequent text lines for columns 3, 4, and 5 should be right justified. Notice that the column headings are aligned with the right hand boundary of these columns.

Advancing past the last column of a table ('.nc' on line 15) causes the first column to become current again. The '.s 2' command on line 16 would have the same effect in addition to skipping two lines. Line 17 places the first line of data into each column of the report. Notice that the dollar values in the last three columns are right justified. Lines 19 and 21 define the output for the second and third lines of the report.

The text 'Chicago Cubs Baseball Uniform' would not fit on one line of the 'ITEM DESCRIPTION' column. As explained above, RPF will place as many words as possible on a line of a column. When a line is full, the last word is right aligned and the remaining text is placed on the next line. Since 'ITEM DESCRIPTION' required two lines, RPF adjusted the other columns so that all the entries for the next item ('ITEM NO.' 8940) are on the same line. RPF will cause the appropriate number of lines to be skipped within each column so that text placement will begin on the line after the longest column.

The summary information is specified on lines 23 and 24. Since no text was displayed in the first two columns, two '.nc' commands were included to position to the third column. The output text for the last three columns contains a line of dashes followed by the column total on the next line.

The '.te' on line 25 terminates the current table; Table '2'. Terminating a table causes the previously invoked table to become active again; Table '1'. The text on line 26 will be centered and underlined within the column boundaries of 13 and 73. The back slash '\' requests RPF to insert a second blank character between each of the words. Normally only one blank separates words on output.

The '.te' on line 27 terminates Table '1' causing the default table to become active.

R E P O R T F O R M A T T E R

(R P F)

3.1 **Overview**

The Report Formatter (RPF) reads a file which contains text for the report, database-derived information produced by RPT, and RPF commands. RPF then writes a file that contains the text of the original file, arranged as directed by the RPF commands. In addition to the imbedded RPF commands, the final format is also controlled by the choice of RPF options, which are selected at the start of RPF execution. The file read by RPF is usually constructed by the use of a text editor; then RPT is executed first, to process commands to derive database information for the report, then RPF is executed to format all of the report information for printing.

In addition to the use described above to generate an ORACLE report, RPF can also be used alone as a general-purpose formatting program for a variety of word processing applications, such as correspondence, memoranda and reports such as this manual, which is itself produced using RPF.

3.2 **RPF Input**

Input to RPF consists of text and RPF commands, intermixed in the input file. The basic unit of text data is a **word** which is a string of one or more characters terminated by a blank, tab, carrier return, or form feed (newline) character. The input string:

The boy went<tab>to the<cr> store.

contains the following six words: "The", "boy", "went", "to", "the", and "store". Blanks, tab, carrier return, form feed, and characters serve only as delimiters and have no effect on the placement of words in the RPF output.

3.2.1 **RPF Command Format**

RPF commands control the placement of words in an output line, horizontal and vertical margins, page numbering and control, horizontal and vertical spacing, line skipping, etc.

All RPF commands start with either a pound sign (#) or period (.). The "#" or "." may be used interchangeably without affecting the meaning of the command. Words which begin with a "#" or "." but are not valid commands will be treated as text. To produce output that is identical to a valid command, the valid command is preceded with an additional "." or "#". For example, #b is a command to insert a blank line, but ##b will be treated as text and output as #b.

Commands may be specified in upper or lower case characters. Each command may be followed by one or more parameters. Commands and their associated parameters are separated by one or more blank, tab or form feed character. A single "." or "#" is used to terminate commands which have a variable number of parameters, or operate on a group of words (e.g. centering, underlining, etc.).

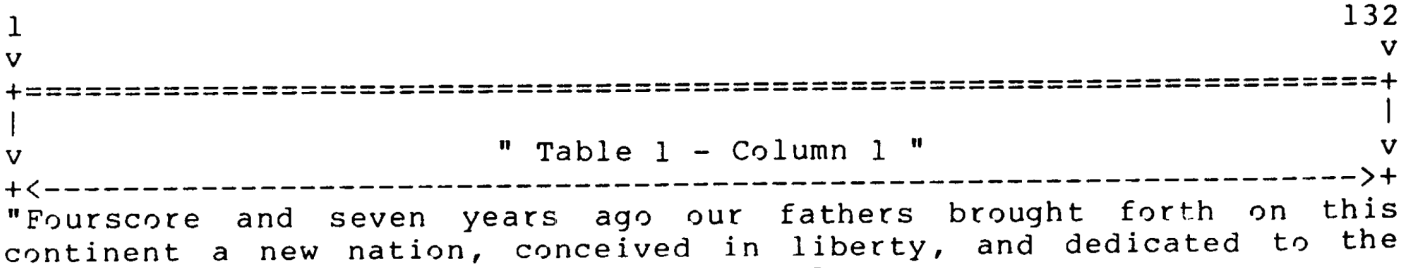
3.3 **Tables and Columns**

RPF processes text one word at a time, placing each word into the output line within the boundaries of the current column. A **column** is defined by its starting and ending character position. Words are placed into columns separated by at least one blank character, beginning with the first character position, and extending to the end of the column. Words are indivisable, and are not split across the lines of a column. Therefore, a word's length must be less than or equal to the width of the current column, or an error indication will be given by RPF.

The width of a column is equal to the last character position minus the first character position plus 1. The last word is aligned with its last character placed in the last column position. Any additional spaces introduced by alignment are evenly distributed throughout the line.

Figure 3.1, Example 1 shows the text of Lincoln's Gettysburg Address placed in a column beginning in position 1 and extending to position 132. Note that all lines except the last are right justified.

Example 1: **Default Table**



Example 2: **Define Table Within Current Column of Previous Table**

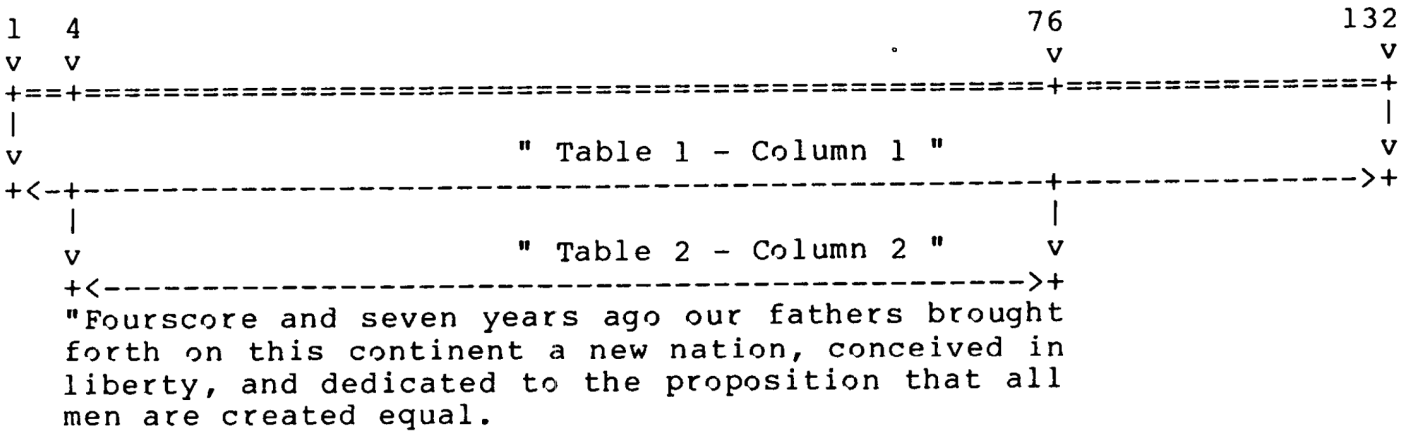
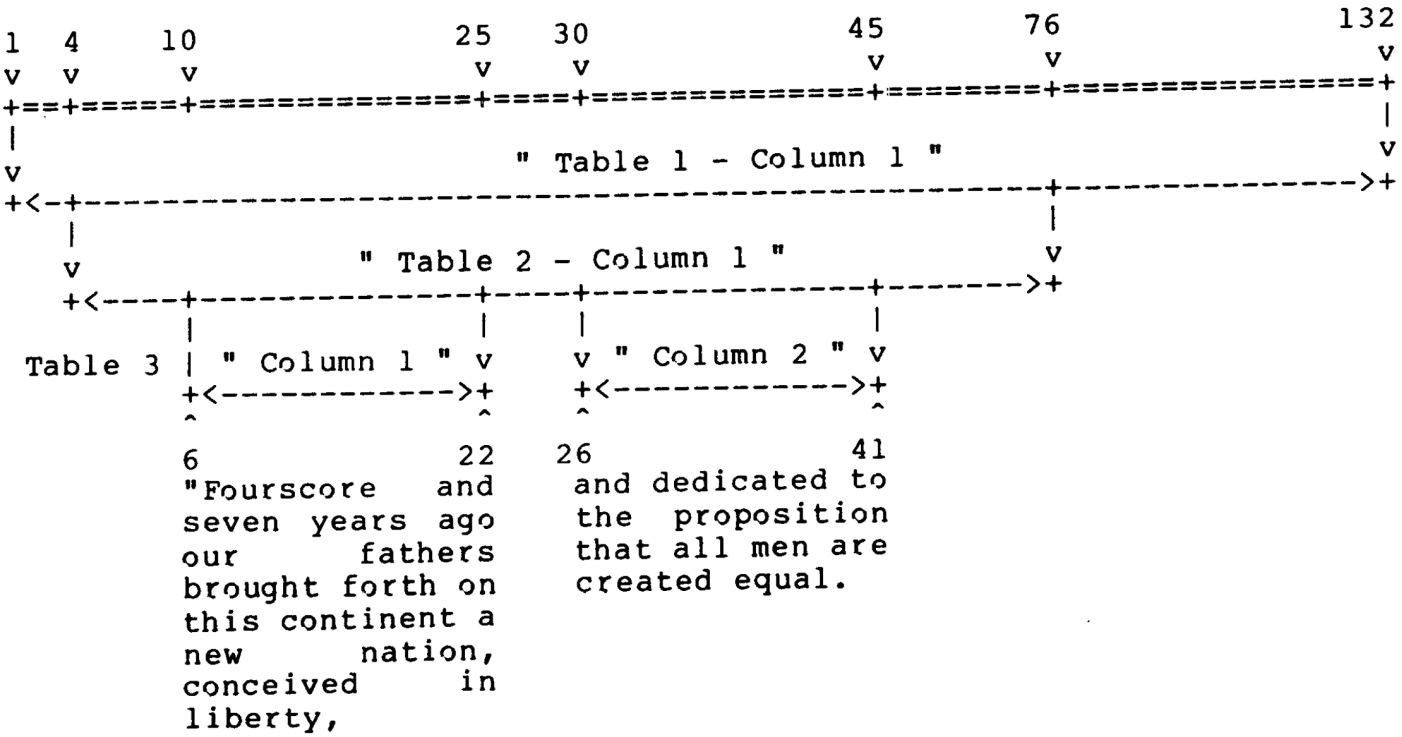


Figure 3.1 - Part 1 of 2

Example 3: **Define Two Column Table within Current Column**



TABLES AND COLUMNS

Figure 3.1 - Part 2 of 2

A **table** consists of from 1 to 20 columns, and defines the boundaries for word placement in the current line. The default table shown in Example 1 contains one column, and serves as the initial table definition. Other tables may be defined, and invoked within the boundaries of the current column.

Example 2 of figure 3.1 shows a second table which has been invoked within the single column of the default table. This table has one column which extends from the 4th position to the 76th position. The text of Example 1 is reformatted within this new table and column definition.

A table is always invoked within a column, and its column's character positions are counted relative to the beginning of that column. Example 3 shows a third table which contains two columns; the first begins in relative position 6 and ends in position 22 and the second begins in relative position 26 and ends in position 41. This table has been invoked within the single column of table 2. Since the column in Table 2 begins relative position 4 of the single column in the default table, the columns of Table 3 begin in absolute locations 10 and 30.

A table must fit within the column in which it is being invoked; that is, the length of the table must be less than or equal to the length of the column. The length of a table is equal to the last character position of the last column defined in that table. For example, Table 3 has a length of 41 (the second and last column extends from position 26 through 41). Therefore, this table can fit within the current column, whose length is 73 (76-4+1).

In Example 3, the sample text has been partially displayed in both columns of the table. RPF will place words into the current column until a command to advance to the next column is encountered. In this example, a "next column" command after the word "liberty," causes RPF to place the remaining text into column 2

Commands to define and invoke tables and advance the column position are presented in detail in section 00.

3.4 **Imbedded Blanks**

Sometimes it is necessary to preserve a specified number of blank spaces between two words. Normally, the number of blanks between words in the input file is ignored, and RPF will separate words by single blanks. A back slash character followed by a blank is used to include additional blanks in the output, as illustrated by this example:

	Input Text	Ouput Text
1-	Abraham Lincoln	Abraham Lincoln
2-	Abraham Lincoln	Abraham Lincoln
3-	Abraham \ Lincoln	Abraham Lincoln
4-	Abraham \ \ Lincoln	Abraham Lincoln
5-	Abraham\\Lincoln	Abraham\Lincoln

Lines 1 and 2 show normal RPF operation. The combined "\ " causes a second blank to be included in line 3 and a third in line 4. The double back slash on line 5 informs RPF that it is desired to have a "\" printed in the output text, as opposed to reserving blanks. A single "\" in the text will be ignored.

3.5 **RPF Commands**

This section will discuss each RPF command in detail. The commands will be presented in alphabetical order, and no attempt will be made to demonstrate their interrelationship. Figure 3.2 provides a summary of these commands.

Command	Description
.APN	Alternate Page Number: placement for even numbered pages.
.B	Blank: Insert one blank line in output text.
.CEN	Center: Center following text in current column.
.CL	Column Literal: Suspend formatting for the following lines in the current column
.CS	Column Skip: Skip 'n' lines in current column
.CUL	Center With Underline: Center and underline the following text within the current column
.DT	Define Table: Define the column boundaries for the specified table.
.F	Figures: Reserve specified page numbers in output document for figures, charts, etc.
.HS	Horizontal Spacing: for characters on Diablo type printer
.I	Indent: Ident the following text in current column.
.L	Literal: Suspend formating for the following text lines; column definitions are ignored.
.N	New Line: in the current column.
.NC	New Column: Advance to next column
.NP	New Page: End current and start new page.
.PAGE	Page: Define top and bottom page boundaries

RPF Commands

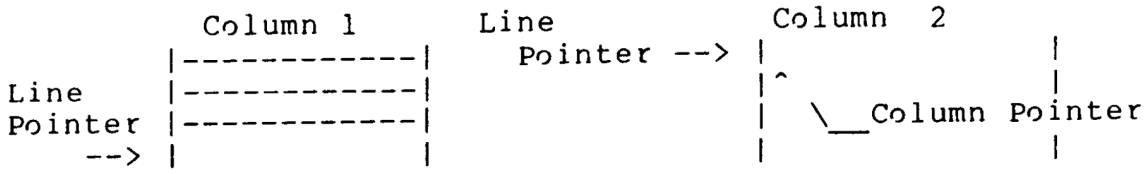
Figure 3.2

Command	Description
.P	Paragraph: New line to be started within the current column, indent 5 spaces at the beginning of the line.
.PAUSE	Pause: Pause outputting to terminal until signal from operator
.R	Right Justify: Set/reset switch to right justify all text placed in current column
.S	Skip: Skip specified number of lines
.SP	Spacing: Define spacing for current column
.SPN	Start Page Numbering: specify page numbering
.T	Table: Invoke specified table within the current column
.TE	Table End: Terminate table and revert to previous column definition
.UL	Underline: Underline the following text.
.VS	Vertical Spacing: Define vertical spacing for Diablo type terminal.

RPF Commands

Figure 3.2 (Continued)

For the purpose of the following command discussions two pointers will be introduced. The **column pointer** identifies the current column within the active table. The **line pointer** identifies where the next line of text for a column will be outputted. Each column has its own line pointer. For some commands, diagrams will be provided to exemplify the movement of these pointers. The following diagram shows a two column table with the **column pointer** positioned to column 2. Column 2's **line pointer** points to line 1, while Column 1's line pointer points to line 4.



3.5.1 **Alternate Page Number**

This command is used to define the character position where the page number will be printed for even number pages. If both sides of the paper are used during reproduction, this command will allow even numbered page numbers to appear on the outside edge of the page. See #SPN command for details on page numbering.

.APN <position>

<position>	is the absolute position that the first character of the page number will be placed on even numbered pages.

3.5.2 **Blank**

This command causes one blank line to be inserted, and is equivalent to #S 1. This command affects all the columns in a multi-column table. Refer to #S command for details on the advancing of column and line pointers.

.b

3.5.3 **Center**

This command causes the specified text to be centered within the current column. The centering takes place on the next line and includes all the words between the command and the first unattached "." or "#". The centered text must fit within the current column.

CEN <text> .

<text> .	is the collection of words to be centered. Note that standard formatting is performed; multiple blanks, and tab and form feed characters are ignored.

3.5.4 **Column Literal**

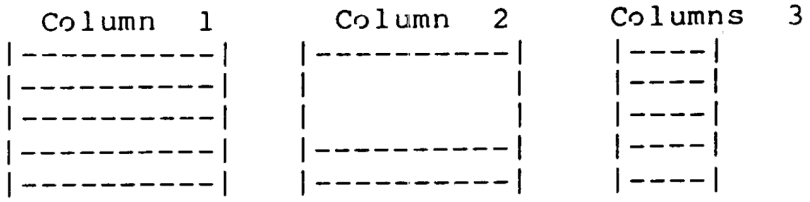
This command defines one or more lines of text which will be outputted in the current column without formatting. Each line will be sent to the output terminal exactly as entered, including multiple blanks, tab characters and form feed characters. The length of each line must not exceed the width of the current column.

.cl
...
...
...
<text lines>
...
...
...
.

<text lines>	lines of text sent unformatted to output terminal

3.5.5 **Column Skip**

This command causes the specified number of blank lines to be inserted into current column. This command has no effect on any other column. The following diagram shows two blank lines inserted into column 2 whereas these lines need not be blank for columns 1 and 3.



.CS <no. lines>

<no. lines> is the number of blank lines inserted.

3.5.6 **Center With Underline**

This command causes the specified text to be centered within the current column and underlined. Centering takes place on the next line and includes all words between the command and the first unattached "." or "#". The centered text must fit within the current column.

.CUL <text> .

<text> is the collection of words to be centered and underlined. Note that standard formatting is performed; multiple blanks, and tab and form feed characters are ignored.

3.5.7 **Define Table**

This command defines a table and its associated columns. The command does not cause the table to be invoked. For information on invoking a table, refer to section 3.5.21 (Table Command).

.DT <table id> <sp1> <ep1> <sp2> <ep2> ... <spn> <epn> .

<table id> is a number from one to ten which identifies the table.

Specifying a previously used **<table.id>** will cause the old definition to be replaced with the new definition.

<spn> <epn> Each pair of numbers defines the boundaries of a column.

<spn> is the starting position of the nth column relative to the beginning of the table.

<epn> is the ending position of the nth column relative to the beginning of the table.

The boundary of a column includes the starting and ending column positions.

The starting position of a column must be at least one greater than the ending position of the previous column.

At least one column must be defined in a table.

If the ending position of the last column is less than or equal to zero, the ending position will default to the end of the column in which the table is being invoked. Extending the last defined column to the end of the invoking column causes the right margin to be right justified.

The **.DT** command is terminated by a single unattached "." or "#".

Example:

.DT 1 12 24 32 48 .	This command defines a table with an id of 1 which contains two columns. The first extends from position 12 to position 24, and the second from position 32 to position 48.
.DT 1 13 73 . **.DT 2 5 0 .**	Two one column tables are defined. The single column of table 2 extends from position 5 to the end of the column in which it is invoked. If table 2 was invoked within table 1, the output text would be right aligned.

3.5.7 **Figures**

This command reserves the specified page numbers for figures, charts, diagrams, etc. If page numbering is used, the specified page numbers will be omitted from the output document.

.F <pgno1> <pgno2> ... .

<pgno1>	Is the page number to be skipped in the output document.

This command is terminated by a single unattached "." or "#".

3.5.8 **Horizontal Spacing**

This command is used to set the horizontal spacing on the Diablo printer terminal.

.HS <spacing>

<spacing>	is the number, when multiplied by 1/60 of an inch, will determine the spacing from one character to the next. For example, a value of 6 will provide spacing of 1/10 of an inch, yielding 10 characters per inch. Note: The spacing between words is variable and beyond the user's control, but is as close to the defined spacing as possible.

The default value is 6, or 10 lines per inch.

3.5.9 **Indent**

This command is used to indent the following text within the current column. It is a shorthand way to define and invoke a table with one column, which begins the specified number of spaces within the current column and extends to the end. For example, **.I 5** is equivalent to invoking the table defined by **.DT 1 5 0 . .** The indenting is terminated when a **.TE** command is encountered. This command follows the same rules used in invoking tables.

.I <indent-number>

<indent-number> is the number of spaces to indent the following text

3.5.10 **Literal**

This command defines one or more lines of text which will be written exactly as entered without formatting. Commands embedded within these lines are ignored and treated as text. Table and column definitions are suspended, with each output line beginning in position 1. Literals are the only case where multiple blanks, tab characters and form feed characters can be explicitly included in the output document.

The command is terminated when a "." or "#" is encountered in the first character position of a line. Any other text on this terminating line is ignored.

.L
...
...
...
<text lines>
...
...
...
.

<text lines> are the lines which are sent directly as entered to the output terminal.

3.5.11 **New Line**

This command causes a new line to be started in the current column. Note that an automatic **.N** command is issued whenever any other command is encountered (e.g. #T, #B, etc.).

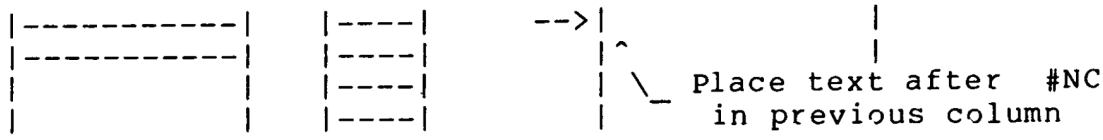
.n

3.5.12 **New Column**

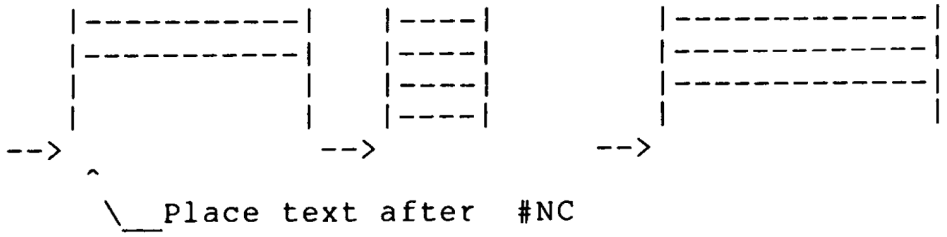
This command causes the current column to end and the next column to be started. If the current column is the last column in the table, the first column of the table is started.

.nc

If the current column contains multiple lines, **.NC** will cause the text to be placed in the top line of the new column. This is shown in the following diagram:



Advancing to the first column will cause each column's line pointer to be set to the line following the longest column.



3.5.13 **New Page**

This command causes a new page to be started.

.NP

3.5.14 **Paragraph**

This command causes a new line to be started within the current column, and five spaces to be inserted at the beginning of the line.

.P

3.5.15 **Page**

This command defines the top and bottom margins of a page.

.PAGE <top-line> <bottom-line>

<top-line>	is the number of lines skipped from the top of the page before outputting text.
<bottom-line>	specifies the last line on the page where text will be outputted.

Line number one is the first line of the page. When a page is full RPF will automatically skip to the next page. See Section (3.6.3) for a discussion of page advancing and printer form feed.

3.5.16 **Pause**

This command causes RPF to pause and wait for a single character to be input from the output terminal. Any character is acceptable, and will cause the printing to resume. The entered character will not be printed. The purpose of the pause is to allow the operator to adjust the form or change the paper before printing will continue.

The **.PAUSE** command is ignored if the PA switch is not selected when RPF is invoked. See section 3.6.4 for details on the PA switch.

.Pause

3.5.17 **Right Justify**

This command causes the text within the current column to be right justified. Right justification is most useful for aligning a column of numbers. The default for any column is left justification.

.R

This command sets a switch which indicates that all text placed in this column will be right justified. Right justification will remain in effect for this column until another **.R** command is issued or the table is terminated.

3.5.18 **Skip**

This command causes the specified number of lines to be skipped. This command should be specified while positioned in the first column of a line; skipping will effect all columns across the line.

.S <no. of lines>

<no. of lines> is the number of lines to be skipped

Following a **.S** , text output will resume at the beginning of the current column (automatic new line - **.N**).

3.5.19 **Space**

This command sets the spacing in the current column. The default is single spacing.

.SP <spacing>

<spacing>	is a number indicating the spacing for the current column. For example, **.SP 3** will cause the spacing to be set to triple spacing.

The spacing value will remain in effect until another **.SP** command is issued When a table is invoked, all the columns in that table will take on the spacing value of the column in which it was invoked. When a table is terminated, a column's spacing value will revert back to the value when the table was invoked.

Each column within a table may have a different spacing value.

3.5.20 **Start Page Numbering**

This command defines page numbering. Parameters specify the starting page number, type of numbering, and placement of the number on the page.

.SPN <type> <pos> <skip-lines> <start-number> [<sect-number>]

<type>	is the number 1, 2, or 3 indicating the type of numbering:

1- Section Page Numbering:
The page number is printed as m-n, where "m" is the section number and "n" is the page number.

2 - Letter Page Numbering:
The page number is printed as -n-, where "n" is the page number. If the page number is one, the page number is not printed.

3 - Period Page Numbering:
The page number is printed as n. , where "n" is the page number.

<pos> is the starting character position where the page number will be printed.

Note: Page numbering is always printed on the first printable line on a page. See section 3.2.15, #PAGE command for defining page margins.

<skip-lines> is the number of lines to skip after printing the page number.

<start-number> is the number of the first output page.

<sect-number> is the section number "m", if type 1 page numbering has been selected. Invalid for the other numbering types.

For example:

.cen .SPN 1 70 4 1 2 .

will cause Section Page Numbering to be used, beginning at page 2-1 of section 2 . The page number will be printed on the first line, beginning in position 70 , and 4 lines will be skipped before text will be printed.

3.5.21 **Table**

This command invokes the specified table within the current column. The column boundaries defined in the invoked table are interpreted relative to the start of the current column.

Text following the **.T** command is placed in the first of the invoked table. Placement of text in other columns is accomplished by using the **.nc** command. A table is terminated with a **.TE** command.

.T <table id>

<table id>	is the table number specified when the table was defined (See **.DT** command, Section 3.5.7).

Refer to Section 3.3 for a detailed discussion of Tables and Columns.

3.5.22 **Table End**

This command indicates the end of a table. When a table is ended, the column in which the table was invoked is in force, and text output will continue at the beginning of that column.

.TE

3.5.23 **Underline**

This command causes the specified text to be underlined.

.ul <text>

<text>	is the text to be underlined.

The underlined text is terminated by the first unattached "." or "#".

3.5.24 **Vertical Spacing**

This command is used to set the vertical spacing on the Diablo printer terminal.

.VS <spacing>

<spacing>	is the number, when multiplied by 1/48 of an inch, will determine the spacing from one line to the next. For example, a value of 8 will provide spacing of 1/6 of an inch, yielding 6 lines per inch.

The default value is 8, or 6 lines per inch.

3.6 **Executing RPF**

The Report Formatter (RPF) is invoked from the user's terminal. output can be directed back to the invoking terminal, or to any other terminal or line printer device including the system spool printer

To execute RPF enter the following command:

RSX-11M,VMS

RPF <output-dev>[/SW.../SW]=<input-file>

UNIX

RPF <input-file> <output-dev> [-SW .. -SW]

<input -file> is the name of the input text file. For RSX-11M and VMS, if the file name extension is not specified, the format <input-file>.RPF is assumed.

<output -dev> is the name of the output device. For RSX-11M and VMS, valid names are TT4:, TI:, LP:, or SPOOL for the spool printer.

For UNIX, valid device names are /dev/ttye, /dev/lp, /di etc.

One or more switches (SW) can be specified to control the execution of RPF. The order of specification is not significant, and switches can be expressed in upper or lower case characters. The following sections describe each switch in detail.

3.6.1 **All Bold -- AB**

This switch will cause the entire output document to be printed in bold face. Bold face printing is supported only on the DIABLO terminal. For other terminal types the switch is ignored. The BF (Bold Face) switch must also be specified.

3.6.2 **Bold Face -- BF**

This switch will cause all underlined text to be printed in bold face. Bold face printing is supported only on the DIABLO terminal. For other terminal types the switch is ignored.

3.6.3 **Form Feed -- FF**

This switch will cause a form feed character to be sent to the terminal prior to printing a page. If not used, the appropriate number of blank lines are printed to position the top of the next page. RPF assumes a 66 line page when inserting blank lines. Forms of other lengths requires the use of FF for positioning.

The switch is ignored for terminals which do not support a form feed character.

3.6.4 **Pause -- PA**

This switch will cause RPF to suspend printing at the end of each page and when a **.PAUSE** command is encountered. Printing is resumed after any input character is entered. The entered character is not printed.

Pause allows documents to be printed on cut forms, and for special setup during the printing process. The command will be ignored if PA was not specified.

3.6.5 **Page -- PG:n[:m]**

This switch will cause a range of pages to be printed, from n to m. For example:

```
PG:5    --- Print the fifth through the last page.
PG:5:7  --- Print pages 5, 6, and 7.
PG:5:5  --- Print only page 5.
```

3.6.6 **Upper Case -- UC**

This switch will cause all alphabetic characters to be printed in upper case.

3.6.7 **VT100 -- VT**

This switch will cause all underlined text to be output in reverse video. If not specified for a video display terminal, underline text will be displayed then overwritten with underline characters.

U S I N G T H E

R E P O R T W R I T E R U T I L I T Y

4.1 **Overview**

This section will demonstrate by example how Report Writer statements are combined with RPF commands and user text to generate a complete report program. The approach will be to present the output report along with the generating program. The discussion will center around the report structure and the Report Writer statements. A working knowledge of RPF is assumed, and individual commands will not be discussed. It is recommended that the reader refer to sections 2 and 3 to obtain the necessary RPF background.

4.2 **Example 1 - Tabular Report**

Figure 4.1A is a listing of a simple tablular report. The data for this report was obtained from the 'EMP' table within the ORACLE demonstration database ('PERSONNEL'). The report program which generated this report is listed in Figure 4.1B. The line numbers and titles were added after execution.

To aid the reader in distinguishing between RPF commands and RPT statements, all RPF commands will use the '#' instead of the '.'. Either symbol is allowed, but this convention simplifies the reading and debugging of report programs. Other conventions have also been adopted which simplify the report writing process.

T E C H N O L O G Y S Y S T E M S , I N C

PERSONNEL REPORT FOR SEPTEMBER, 1980

EMPNO	NAME	JOB	SALARY	COMMISSION	DNO
7369	SMITH	CLERK	$800.00		20
7499	ALLEN	SALESMAN	$1,600.00	$300.00	30
7521	WARD	SALESMAN	$1,250.00	$500.00	30
7566	JONES	MANAGER	$2,975.00		20
7654	MARTIN	SALESMAN	$1,250.00	$1,400.00	30
7698	BLAKE	MANAGER	$2,850.00		30
7782	CLARK	MANAGER	$2,450.00		10
7788	SCOTT	ANALYST	$3,000.00		20
7839	OATES	PRESIDENT	$5,000.00		10
7844	TURNER	SALESMAN	$1,500.00		30
7876	ADAMS	CLERK	$1,100.00		20
7900	JAMES	CLERK	$950.00		30
7902	FORD	ANALYST	$3,000.00		20
7934	MILLER	CLERK	$1,300.00		10

END OF REPORT

"Example 1 - Tabular Report "

Figure 4.1-A

```
 1: .REM ************************************************************
 2: .REM *******  SAMPLE REPORT 1 -----  SIMPLE TABULAR REPORT  ********
 3: .REM ************************************************************
 4: .REM
 5: .REM *******   Define RPF Tables   ---  Print   Title        ********
 6: .REM
 7: .REM                    " T a b l e    1 "
 8: .REM +-------------------------------------------------------------+
 9: .REM 4                                                            76
10: .REM                    " T a b l e    2 "
11: .REM +---+  +---------+  +---------+  +----------+  +--------+ +---+
12: .REM 1   5 10         20  25        35  40         50  55       65 67  0
13: .REM
14: #dt 1 4 76 #
15: #dt 2 1 5 10 20 25 35 40 50 55 65 67 0  #
16: #t 1
17: #page 6 58
18: #s 3
19: #cul T E C H N O L O G Y \ \ S Y S T E M S , \ \  I N C #
20: #s 3
21: #cul PERSONNEL REPORT FOR SEPTEMBER, 1980 #
22: #s 3
23: .REM
24: .REM *******     Declare    Program    Variables              ********
25: .REM
26: .DATABASE personnel
27: .REM *******  Print all positions of empno - ie. 0682        ********
28: .DECLARE empno 0999
29: .DECLARE ename a10
30: .DECLARE job a10
31: .DECLARE sal $99,999.99
32: .REM *******  Print 'comm' as blank if value is zero  **************
33: .DECLARE comm $B99,999.99
34: .DECLARE deptno 99
35: .REM
36: .REM *********          Define    SELECT      Macro           ********
37: .REM
38: .DEFINE selemp
39:         SELECT empno,ename,job,sal,comm,deptno
40:         INTO empno,ename,job,sal,comm,deptno
41:         FROM emp
42: ..
```

"Example 1 - Tabular Report"

Figure 4.1B - Part 1 of 2

```
43: .REM
44: .REM ********          Define    BODY       Macro            ********
45: .REM
46: .DEFINE body
47: .REM ****** Print each column variable  -- Advance to next column *****
48:         .PRINT empno
49:         #nc
50:         .PRINT ename
51:         #nc
52:         .PRINT job
53:         #nc
54:         .PRINT sal
55:         #nc
56:         .PRINT comm
57:         #nc
58:        .PRINT deptno
59:         #nc
60: ..
61: .REM
62: .REM ********          Define      HEAD      Macro              ******
63: .REM
64: .DEFINE head
65:         .REM  **** Invoke "Table 2" - Report Column Layout *********
66:         #t 2
67:         .REM  ***********   Print Column Headings **********
68:         #r EMPNO #n ----- #nc
69:         NAME #n ---- #nc
70:         JOB #n --- #nc
71:         #r SALARY #n ------ #nc
72:         #r COMMISSION #n ---------- #nc
73:         #r DNO #n --- #nc
74:         #b
75:         .REM  ********* Execute body macro to print first row *******
76:         .body
77: ..
78: .REM
79: .REM  **********   Define      Foot          Macro       *******
80: .REM
81: .DEFINE foot
82:         #te
83:         #s 4
84:         #cul   END OF REPORT    #
85: ..
86: .REM
87: .REM  ******** Procedure Section  --- Generate Report    ********
88: .REM
89: .REPORT selemp  body head foot
90: #te
91: .STOP
```

"Example 1 - Tabular Report"

Figure 4.1B - Part 2 of 2

The reader should be reminded of the basic rules which apply to RPT:

- RPT's main function is to copy RPF commands and user text, as encountered, to the interim file which will subsequently be processed by RPF. Database information can be included within this output file using the PRINT command.

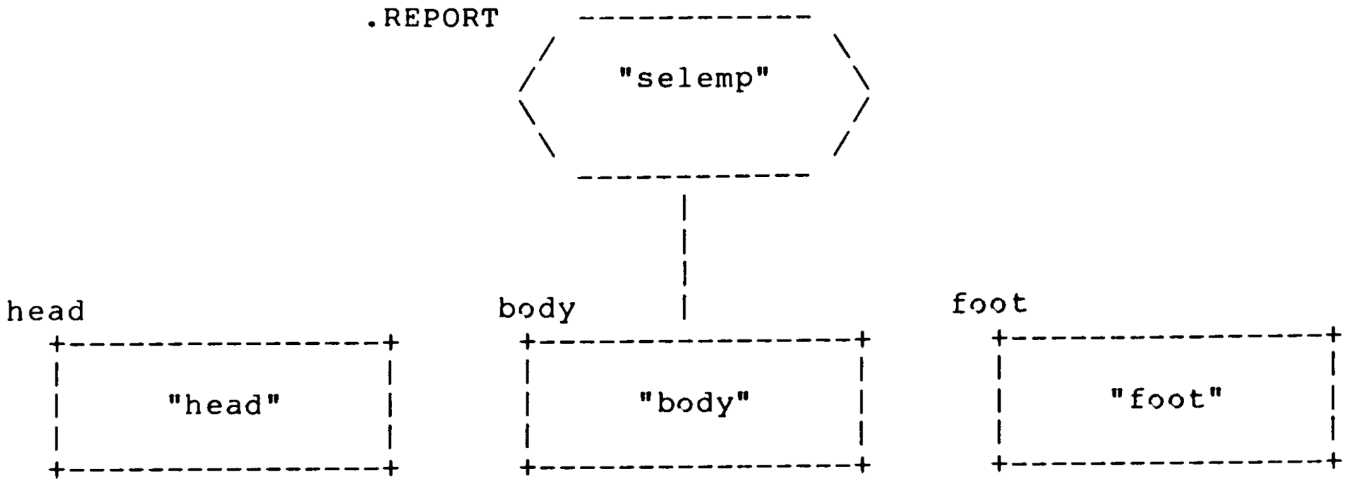
- Program variables and macros (Procedural and SELECT) must be defined prior to being referenced. Macro definitions are stored as encountered, and not executed until explicitly or implicitly requested.

- RPF commands and user text may be defined anywhere within a program.

The basic structure for this report is depicted in figure 4.1C. The **.REPORT** statement drives the retrieval of data and causes the appropriate head, body, and foot macros to be executed. The SQL query defined within the 'selemp' macro (lines 38 through 42 of figure 4.1B) selects all the columns from all the rows in the 'EMP' table. The procedural macros used to output and format this data will be discussed later.

The RPF table definitions used to format the output are pictorially displayed on lines 8 through 12. The data fields returned for each row will be formatted in the six columns defined by Table '2'. Lines 14 through 22 contain the RPF commands to define these tables and the report titles. These commands could also have been included after the program DEFINE and DECLARE statements.

Line 26 identifies the single database which will be referenced within this report. Since the 'personnel' database is secure, a valid 'userid/password' must be specified when the report is executed. Following the 'DATABASE' statement the program variables are declared. The declarative statements may be specified in any order, providing the definition precedes its use.



.REPORT selemp body head foot

"Example 1 - Tabular Report"

Figure 4.1C

By specifying a '0' in the format definition of 'empno', all leading zero digits will be printed. Remember that the declared format defines the maximum value of a variable in addition to the printing format. For example, 'sal' (line 31) can hold a maximum value of 99,999.99; 100,000.00 will overflow this variable. Overflowed variables will be print with a '#' in all digit positions. The 'B' on line 33 will cause zero commission value to be printed as blanks. Any variable which has the 'NULL' value will be printed as blanks.

The 'selemp' SELECT macro defines the SQL query which drives this report. The INTO statement identifies the program variables which will receive the column values defined in the select list. There is a one-to-one correspondence between each column returned and its associated program variable. Column and variable names do not have to be the same.

Lines 43 through 88 define the head, body, and foot procedural macros which will be implicitly executed as a result of executing the '.REPORT' statement on line 89. The head macro called 'head' will be executed once for the first row returned from the query. Table '2' is invoked to establish the six column print layout. RPF commands and user text is included to print the column headings (lines 67 - 74). Notice the '#r' command to establish right justification for the empno, salary, commission, and department number columns. After inserting a blank line the body macro is explicitly executed (Line 76). This will cause the first row to be printed. If omitted, the first row of data would be lost.

The body macro ('body') is implicitly executed for the second through last row returned. A '.PRINT' statement is used to cause the current value of the program variable to be included in the output interim file. Figure 4.1D is a partial listing of the generated 'interim file'. The '#nc' commands causes the row information to be printed in the correct report column. Notice the database information which has been interspersed between the RPF commands.

After the last row of the query has been processed, the foot macro (Lines 81-85) is executed. In this example the foot simply terminates Table '2' and prints 'END OF REPORT'.

```
#dt 1 4 76 #
#dt 2 1 5 10 20 25 35 40 50 55 65 67 0  #
#t 1
#page 6 58
#s 3
#cul T E C H N O L O G Y \ \ S Y T E M S , \ \  I N C #
#s 3
#cul PERSONNEL REPORT FOR SEPTEMBER, 1980 #
#s 3
#t 2
#r EMPNO #n ----- #nc
NAME #n ---- #nc
JOB #n --- #nc
#r SALARY #n ------ #nc
#r COMMISSION #n ---------- #nc
#r DNO #n --- #nc
#b
7369
#nc
SMITH
#nc
CLERK
#nc
$800.00
#nc

#nc
20
#nc
7499
#nc
ALLEN
#nc
SALESMAN
#nc
$1,600.00
#nc
$300.00
#nc
30
#nc
7521
#nc
WARD
#nc
SALESMAN
#nc
$1,250.00
#nc
```

" Example 1 - Tabular Report - 'Interim File' "

Figure 4.1D

Following the '.REPORT' statement in the procedure section is a '.te' to terminate Table '1' and a '.STOP' statement to terminate the report program. The 'STOP' is optional, and if omitted the program would end after the last statement. When included, a stop message is displayed on the executing terminal.

Report generation is a two step process. This report was generated using the following system commands:

Step 1 - RPT Execution:

```
>RPT sampl.rpt sampl.rpf SCOTT/TIGER
```

Where 'sampl.rpt' is the input report text file, and 'sampl.rpf' is the interim file created by RPT. Since the 'PERSONNEL' database is secure, a valid userid and password must be provided. Note: The '>' is an RSX11M system prompt, and will vary by operating environment, however the command text is the same across the supported systems.

Step 2 - RPF Execution:

```
>RPF LP:=sampl
```

Where 'LP:' is the system name for the output device. The output may be directed to either a system device or another file. The input file is the file created by RPT in the previous step. Since no file extension was specified, 'rpf' was assumed, and the fully qualified name 'sampl.rpf' is used.

4.3 **Example 2 - Nested Report**

Figure 4.2A is a listing of a **nested** report. A nested report is one in which a second report is executed within the head, body, or foot macro of the primary report. In this example, the primary or 'outer' report, provides a listing of the department names and numbers for each department within the division. Then for each department selected, a second report is executed which lists the employees in that department. After listing the individual employees, the department's salary summary is reported.

D i v i s i o n P a y r o l l R e p o r t

by Department

****

DEPTNO: 010 - - DEPTNAME: ADMINISTRATION

****

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
7782	CLARK	$3162.50		$37,950.00
7934	MILLER	$1300.00		$15,600.00
7839	OATES	$5750.00		$69,000.00

Department Summary ----------

AVG = $3,404.17 MIN = $1,300.00 MAX = $5,750.00 $122,550.00

****

DEPTNO: 020 - - DEPTNAME: RESEARCH

****

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
7369	SMITH	$800.00		$9,600.00
7566	JONES	$3421.25		$41,055.00
7876	ADAMS	$1100.00		$13,200.00
7788	SCOTT	$3000.00		$36,000.00
7902	FORD	$3000.00		$36,000.00
7955	WILSON			
7956	JAKES	$1000.00		$12,000.00

Department Summary ----------

" Example 2 - Nested Report "

Figure 4.2A - Part 1 of 3

AVG = $2,053.54 MIN = $800.00 MAX = $3,421.25 $147,855.00

****

DEPTNO: 030 - - DEPTNAME: SALES

****

EMPNO	NAME	MONTHLY SALARY	COMM	ANNUAL COMPENSATION
7499	ALLEN	$1600.00	$300.00	$19,500.00
7521	WARD	$1437.50	$500.00	$17,750.00
7698	BLAKE	$3277.50		$39,330.00
7654	MARTIN	$1437.50	$1400.00	$18,650.00
7844	TURNER	$1500.00	$0.00	$18,000.00
7900	JAMES	$950.00		$11,400.00
7989	CARTER	$1500.00	$0.00	$18,000.00

Department Summary

AVG = $1,671.79 MIN = $950.00 MAX = $3,277.50 $142,630.00

****

DEPTNO: 040 - - DEPTNAME: OPERATIONS

****

"Example 2 - Nested Report"

Figure 4.2A - Part 2 of 3

D i v i s i o n T o t a l s

MINIMUM SALARY	MAXIMUM SALARY	AVERAGE COMPENSATION	TOTAL COMPENSATION
$800.00	$5,750.00	$25,814.69	$413,035.00

E n d O f R e p o r t

"Example 2 - Nested Report"

Figure 4.2A - Part 3 of 3

A block diagram of this example is provided in Figure 4.2C. The 'inner' report which lists the employee information is explicitly executed within the body macro ("deptbody") of the 'outer' department report. Associated with this 'inner' report are head, body, and foot macros labeled "emphead", "empbody", and "empfoot". Notice that within the "empfoot" macro, two other macros are explicitly executed. The "deptsum" select macro uses a SQL query to compute the summary salary information. The procedural macro "compsum" adds the department summary to the division totals. The division summary is generated by the "deptfoot" macro after all departments have been processed.

Figure 4.2B is the program listing for this example. The line numbers and titles at the bottom of the page were added after the report was executed. The data for this report was obtained from the 'DEPT' and 'EMP' tables within the 'PERSONNEL' database (Line 16).

This report contains three SELECT macros. "seldept" (Lines 42-46) drives the outer report berequesting each department's name and number from the 'DEPT' table. The absence of the "WHERE" clause will cause all the rows in the table to be returned. The "selemp" macro will drive the inner report returning all the employees in a particular department. The substitution variable '&deptno' will cause the current value of the program variable 'deptno' to be substituted into the "WHERE" clause.

The fifth value returned in the SELECT list(Line 51) is the computed annual compensation. The null function was required for the 'comm' column because all employees other than salesman have a null value for commission. If 'nvl' was omitted the annual compensation for non-salesman would be assigned the null value and printed as blanks.

The "deptsum" SELECT macro is explicitly executed in the foot macro of the inner report. After all the employees in a department have been listed this query uses the SQL built-in functions to compute the count, minimum, maximum, average, and sum of the salaries for the reported department. This technique for computing summary information requires a second pass of the data.

```
001: .REM ************************************************************  **
002: .REM ********  SAMPLE   REPORT  2   ---  N E S T E D   R E P O R T  **  **
003: .REM ************************************************************
004: .REM
005: .REM    This sample demonstrates the capability to nest a report within the
006: .REM    head, body, or foot of another report. In this example the "payroll
007: .REM    status for each department within a division will be reported. The
008: .REM    compensation for each employee is listed ,                 followed
009: .REM    by the department's summary information. At the end of the report
010: .REM    the entire division's summary information is reported.
011: .REM
012: .REM ************************************************************
013: .REM
014: .REM            D e c l a r e              V a r i a b l e s
015: .REM
016: .DATABASE personnel
017: .DECLARE  deptno 000
018: .DECLARE  dname a15
019: .DECLARE  empno 0999
020: .DECLARE  ename a15
021: .DECLARE  monsal $9999.99
022: .DECLARE  comm   $9999.99
023: .DECLARE  annsal $99,999.99
024: .DECLARE  deptsum $999,999.99
025: .DECLARE  deptmin $9,999.99
026: .DECLARE  deptmax $9,999.99
027: .DECLARE  deptavg $9,999.99
028: .DECLARE  deptcnt 9999
029: .DECLARE  divsum  $999,999.99
030: .DECLARE  divavg  $99,999.99
031: .DECLARE  divmin  $9,999.99
032: .DECLARE  divmax  $9,999.99
033: .DECLARE  empcnt  9999
034: .REM  ************************************************************
035: .REM
036: .REM          D e f i n e   S E L E C T   M a c r o s
037: .REM
038: .REM  ************************************************************
039: .REM
040: .REM          Select   Department   Information
041: .REM
042: .DEFINE seldept
043:         SELECT deptno,dname
044:         INTO   deptno,dname
045:         FROM   dept
046: ..
047: .REM
048: .REM          Select Employee Data Within a Department
049: .REM
050: .DEFINE selemp
```

" Example 2 - Nexted Report Program Listing "

Figure 4.2B - Part 1 of 5

```
051:          SELECT empno,ename,sal,comm,(sal*12 + nvl(comm,0))
052:          INTO   empno,ename,monsal,comm,annsal
053:          FROM   emp
054:          WHERE  deptno = &deptno
055: ..
056: .REM
057: .REM           Select Department Summary Information
058: .REM
059: .DEFINE deptsum
060:         SELECT count(sal),min(sal),max(sal),avg(sal),sum(sal*12+nvl(comm,0)
061:         INTO   deptcnt,deptmin,deptmax,deptavg,deptsum
062:         FROM   emp
063:         WHERE  deptno= &deptno
064: ..
065: .REM    ***************************************************************
066: .REM
067: .REM         D e f i n e    P r o c e d u r a l    M a c r o s
068: .REM
069: .REM    ***************************************************************
070: .REM
071: .REM         D i v i s i o n    R e p o r t    " H e a d "
072: .REM
073: .DEFINE depthead
074:           .REM   This head  macro just processes the first row of
075:           .REM   Department report. It is required only because a foot
076:           .REM   is specified on the REPORT statement.
077:           .deptbody
078: ..
079: .REM
080: .REM      D E P A R T M E N T   R e p o r t    " B o d y "
081: .REM
082: .DEFINE deptbody
083:           .REM
084:           .REM    E x e c u t e    E m p l o y e e       R e p o r t
085:           .REM
086:           .REM      P r i n t   H e a d i n g   F o r   D e p a r t m e n t
087:           .REM
088:          #s 2
089:          #cul  ************************************************** #
090:          #b
091:          #cen
092:          DEPTNO:
093:          .PRINT deptno
094:          - - DEPTNAME:
095:          .PRINT dname
096:          #
097:          #b
098:          #cul  ************************************************** #
099:          #s 2
100:          .REM      Report Each Employee within the current Department
```

" Example 2 - Nexted Report Program Listing "

Figure 4.2B - Part 2 of 5

```
101:          .REM
102:          .REPORT selemp empbody  emphead  empfoot
103: ..
104: .REM
105: .REM       D E P A R T M E N T   R e p o r t   " F o o t "
106: .REM
107: .DEFINE deptfoot
108:         .REM   Compute Division summary information
109:         .REM
110:         .REM      Compute "Average Monthly Salary"
111:         .REM
112:         .DIV divavg divsum empcnt
113:         .REM
114:         .REM      Output: Maximum and Minimum Salary
115:         .REM              Average and Total Yearly Compensation
116:         .REM
117:         #np
118:         #cul D i v i s i o n  \  T o t a l s #
119:         #s 1
120:         #t 4
121:         #r #nc #r #nc #r #nc #r #nc
122:         MINIMUM #nc MAXIMUM #nc AVERAGE #nc  TOTAL   #nc
123:         SALARY #nc SALARY #nc COMPENSATION #nc COMPENSATION #nc
124:         ------- #nc --------     #nc ------------ #nc ------------ #nc
125:         .PRINT divmin
126:         #nc
127:         .PRINT divmax
128:         #nc
129:         .PRINT divavg
130:         #nc
131:         .PRINT divsum
132:         #te
133:         #s 2
134:         #cul E n d  \  O f \ R e p o r t  #
135:         #te
136: ..
137: .REM    ***************************************************************
138: .REM
139: .REM             E M P L O Y E E        "H e a d"    M a c r o
140: .REM
141: .DEFINE  emphead
142:           #t 2
143:           #s 1
144:           EMPNO #nc #cen NAME # #nc MONTHLY #nc COMM #nc ANNUAL #nc
145:           #nc #nc SALARY #nc #nc COMPENSATION #nc
146:           ----- #nc --------------- #nc -------- #nc -------- #nc
147:           ------------ #nc
148:           .REM   Set-up Right/Left Justification  Switches
149:           .REM        Insert One Blank Line
150:           #nc #nc #r #nc #r #nc #r #nc
```

▪ Example 2 - Nexted Report Program Listing ▪

Figure 4.2B - Part 3 of 5

```
151:          .REM
  2:          .REM    Execute 'Body' Macro to Process First Row
  3:          .empbody
154: ..
155: .REM
156: .REM          E M P L O Y E E        R e p o r t    " B o d y "
157: .REM
158: .DEFINE  empbody
159:          .PRINT empno
160:          #nc
161:          .PRINT ename
162:          #nc
163:          .PRINT monsal
164:          #nc
165:          .PRINT comm
166:          #nc
167:          .PRINT annsal
168:          #nc
169: ..
170: .REM
171: .REM          E M P L O Y E E        R e p o r t    " F o o t "
172: .REM
173: .DEFINE  empfoot
174:          .REM
175:          .REM   Compute Department Summary Information
176:          .REM
177:          .EXECUTE deptsum
  8:          #te
179:          #t 3
180:          #s 2
181:          #cul Department Summary # #nc ------------ #nc
182:          #s 1
183:          \ AVG =
184:          .PRINT deptavg
185:          \ MIN =
186:          .PRINT deptmin
187:          \ MAX =
188:          .PRINT deptmax
189:          #nc #r
190:          .PRINT deptsum
191:          #te
192:         .REM   Execute macro to compute division totals
193:          .compsum
194: ..
195: .REM  ************************************************************
196: .REM
197: .REM          C o m p u t e     D i v i s i o n     T o t a l s
198: .REM
199: .DEFINE compsum
200:         .ADD empcnt empcnt deptcnt
```

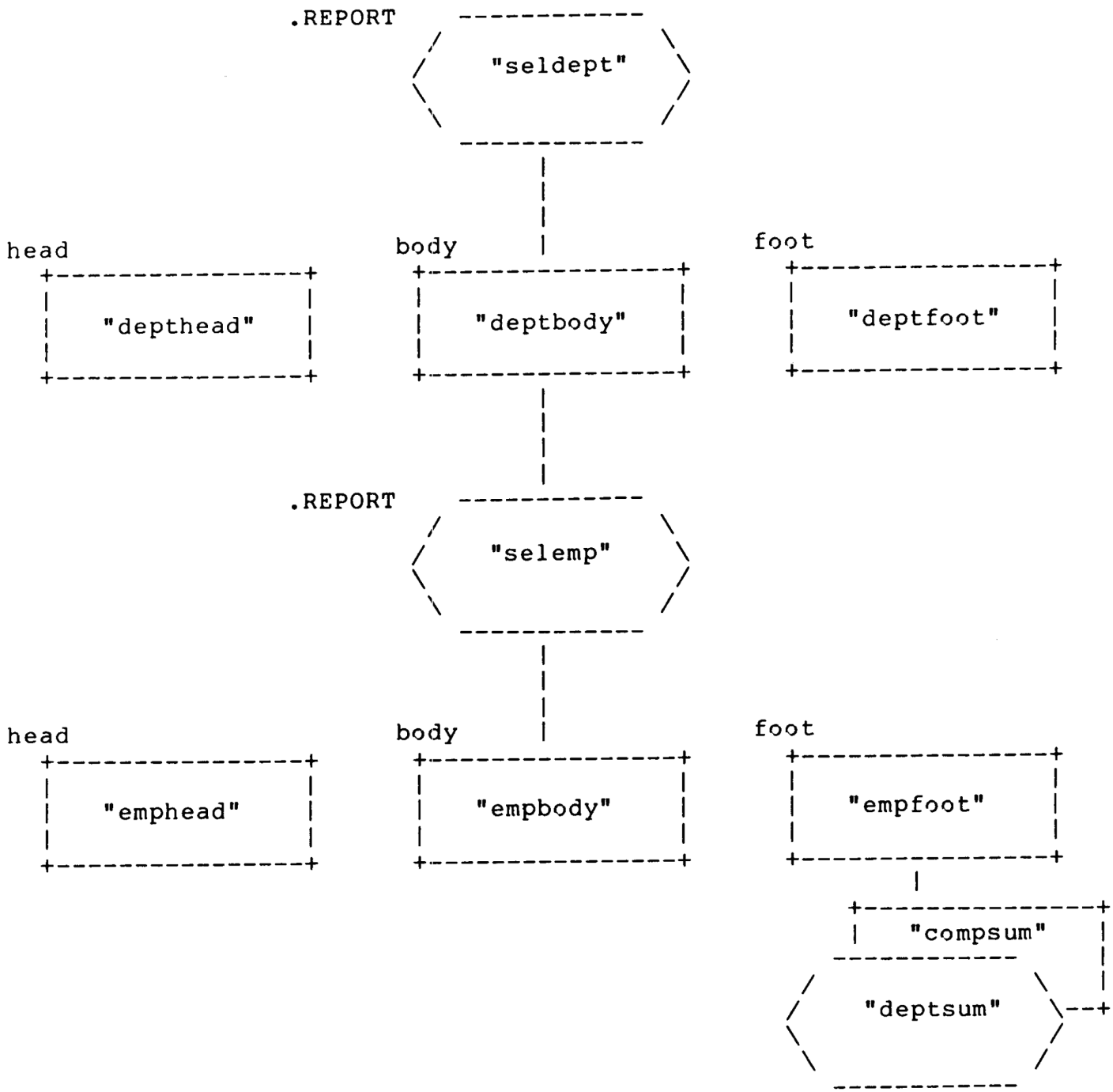
" Example 2 - Nexted Report Program Listing "

Figure 4.2B - Part 4 of 5

```
201:          .ADD divsum divsum deptsum
202:          .IF "&deptmax <= &divmax" THEN compl0
203:          .REM     S e t     N e w    M a x i m u m
204:          .EQUAL divmax deptmax
205:          .&compl0
206:          .IF "&deptmin >= &divmin" THEN comp20
207:          .REM     S e t     N e w    M i n i m u m
208:          .EQUAL divmin deptmin
209:          .&comp20
210:   ..
211: .REM
212: .REM  ***************************************************************
213: .REM
214: .REM
215: .REM            R e p o r t :    M a i n     S e c t i o n
216: .REM
217: .REM
218: .REM            D e f i n e     " R P F "   C o m m a n d s
219: .REM
220: #dt 1 10 75 #
221: #dt 2 5 9 13 27 31 38 42 49 55 0 #
222: #dt 3 1 51 55 0 #
223: #dt 4 1 9 12 20 23 37 47 0 #
224: #page 6 58
225: #spn 2 42 2 1
226: #sp 1
227: .REM
228: .REM            E x e c u t e     D i v i s i o n   R e p o r t
229: .REM
230: .REM           Print    Report    Title
231: #t 1
232: #s 4
233: #cul D i v i s i o n  \  P a y r o l l  \  R e p o r t #
234: #s 2
235: #cul by Department #
236: .REM       Initialize Division Minimum and Maximum Values
237: .REM          Highest value for Minimum; Lowest Value for Maximum
238: .SET divmin 9999.99
239: .SET divmax 0
240: .REM
241: .REPORT  seldept  deptbody depthead deptfoot
242: .STOP
```

" Example 2 - Nexted Report Program Listing "

Figure 4.2B - Part 5 of 5



.REPORT seldept deptbody depthead deptfoot

"Example 2 - Nested Report"

Figure 4.2C

An alternative approach would be to accumulate the values in program variables as each row is processed, then compute the summary data using the RPT mathematical functions. This technique was used to generate the "Division Totals" shown on the third page of the report (Figure 4.2A). However, instead of computing for each employee row, division totals were computed using each department's summary data. This choice was made to improve performance by reducing the frequency of execution of the "compsum" macro. In most cases the difference would not be significant, however to analyze the maximum and minimum values of 'salary' two "IF" statements are required. As utlined in section 4.4.4.3.1 "IF Statement Guidelines", "IF" statements processing is fairly time consuming and therefore the frequency of execution should be minimized.

The procedural macro "depthead" was included only because a "foot" macro was required for the outer report. Because the procedural macros on the .REPORT statement (Line 241) are positionally defined a 'head' macro name must precede the 'foot' name. The explicit execution of the body macro "deptbody" causes the first row to be processed.

The body of the "Department Report" lists the department's name and number, then executes the inner employee report (Line 102). In the 'foot' of the department report the average compensation is computed (Line 112). The headings for the division totals are printed in Lines 121 to 124. Line 121 does not output any text, but has the effect of skipping a single line. The '#r' commands executed in each column forces right justification.

The head macro of the "Employee Report" ("emphead") prints the headings for this inner report and sets the justification for the output columns (Line 150). Again, the body must be executed to cause the first row to be processed. The 'body' outputs each column of data retreived, and the 'foot' computes and lists the department summary information.

"compsum" uses the 'ADD' statement to accumulate the sum of the compensation and the total number of employees. The "IF" statements compare current department minimum and maximum salaries with those values for the division. Since blanks were included within the expressions they must be enclosed within double quotation marks ("). If a new max or min values is **not** established then the expressions are true and control is passed to the macro label specified after the "THEN" clause. Since no "ELSE" statement was specified, a false condition causes the next instruction to be executed. The "EQUAL" statements cause the division values to be set equal to the corresponding department values.

Line 211 is the end of the macro definitions and the beginning of the Procedure section of this report program. As defined on lines 220 through 223, four tables were required for this report. Tables 2, 3, and 4 are invoked within table 1 at various places in the report. The '#spn' command (Line 225) causes the pages of the report to be numbered. Type 2 format (Letter Page Numbering) was selected to be printed beginning in print position '42'. Two lines will be skipped after printing the page number. Numbering will begin with '1', however with type 2 the number is not printed on page one.

The "SET" statements (Lines 238,239) initialize the values of the division minimum and maximum salaries. The "REPORT" statement executes the outer report which drives the entire program.

4.4 **Example 3 - Sample Letter**

This example demonstrates the flexibility of the report writer to construct a computer generated letter. Fixed text is combined with database information to form the completed memo. The power of SQL allows the desired data to be obtained from the database, and the RPF commands and RPT statements permits the data to be interleaved within the letter format.

Figure 4.3A shows the sample which is composed of data from the 'EMP' and 'DEPT' table. The report was executed to generate multiple copies of this memo, each addressed to a different manager. The manager's name was obtained from the 'EMP' table, and the manager's department number, name and location from the 'DEPT' table.

The current 1981 and proposed 1982 salary figures are included within the body of the memo. This information is obtained by computing the sum of the salaries of each employee in the 'EMP' table who is in the addressee's department. This information will be different for each manager's copy of the memo. Although this memo is very simple, it demonstrates the report writer's ability to combine and include data from various tables in the database into a single letter.

Figure 4.3C provides a block diagram of the report program. A single '.REPORT' statement with only a body macro was used. The SELECT statement which drives the report generation joins the 'EMP' table to the 'DEPT' table to retrieve the adressee data. Within the body a second SELECT is explicitly executed to compute the current and proposed salary data for the addressee's department.

A program listing of the report is provided in Figure 4.3B. The line numbers and titles were added after execution. In the 'DECLARE' variables section, the variable 'curdate' has a format of 'date'. This means that the value of curdate is in the internal Julian day number format. If this variable was assigned its value from a column in a SELECT list, that column value must also be in Julian day format. Presently, only IAF supports the creation of a column value in this format. In this program 'curdate' is used to output the current date at the time of execution. Assigning the variable to the current date is accomplished by the .SET statement on line 22. The literal '$$DATE$$' instructs RPT to obtain the date from the system, convert it into Julian format, and then store it into the variable referenced on the SET statement.

The first of the two SELECT macros, 'seladdr', joins the 'EMP' to the 'DEPT' table to acquire the addressee information. The second predicate in the WHERE clause restricts the list of addressees to department managers. The 'selsum' macro uses the SQL arithmetic capabilities to compute both the current sum and the sum incremented by 10% for employees in the manager's department (Line 44).

Date : 03/25/81

To: BLAKE

Department : 30 - SALES

Location : PARIS

From : Bill James
Director of Personnel

Subj : 1982 Employee Compensation Plan

Its budget time again! To aid you in completing the salary portion of your budget I have computed your department's current salary expenses. For planning purposes we are presently estimating an across the board increase of 10% in the 1982 salary pool.

Your department's 1981 figure is: $140,430.00

Estimated 1982 figure is: $154,473.00

In completing your salary plan you should break this total down by individual employees. Please let me know if you have any questions on this or related issues.

Bill

Example 3 - Sample Letter

Figure 4.3A

```
.REM ************************************************************************
.REM **********  SAMPLE REPORT 3  -----  GENERATED LETTER    **********
.REM ************************************************************************
.REM
.REM   This sample demonstrates the capability to create a computer
.REM   generated letter or memorandum. Information from the database is
.REM   combined with the predefined text to form the completed letter.
.REM   A REPORT statement is used to drive the generation of multiple
.REM   letters, each with a different addressee and variable data.
.REM
.REM ************************************************************************
.REM
.REM              D E C L A R E     V A R I A B L E S
.REM
.DATABASE personnel
.DECLARE addressee al0
.DECLARE deptno 999
.DECLARE dname al5
.DECLARE location al5
.DECLARE curdate date
.REM   Set the variable 'curdate' equal to today's date
.SET curdate $$DATE$$
.DECLARE sumsal  $999,999.99
.DECLARE sumsal+ $999,999.99
.REM   ***************************************************************
.REM
.REM          D e f i n e      S E L E C T     Macros
.REM
.REM   ***************************************************************
.REM
.REM          Select   Managers   and their Departments
.REM
.DEFINE  seladdr
         SELECT emp.ename,dept.deptno,dept.dname,dept.loc
         INTO   addressee,deptno,dname,location
         FROM   emp,dept
         WHERE  emp.deptno=dept.deptno
         AND    job = 'MANAGER'
..
.REM
.REM          Select  Department's  Current Payroll
.REM
.DEFINE  selsum
         SELECT sum(sal*12),sum((sal*1.1)*12)
         INTO   sumsal,sumsal+
         FROM   emp
         WHERE  deptno = &deptno
..
.REM    ***************************************************************
.REM
```

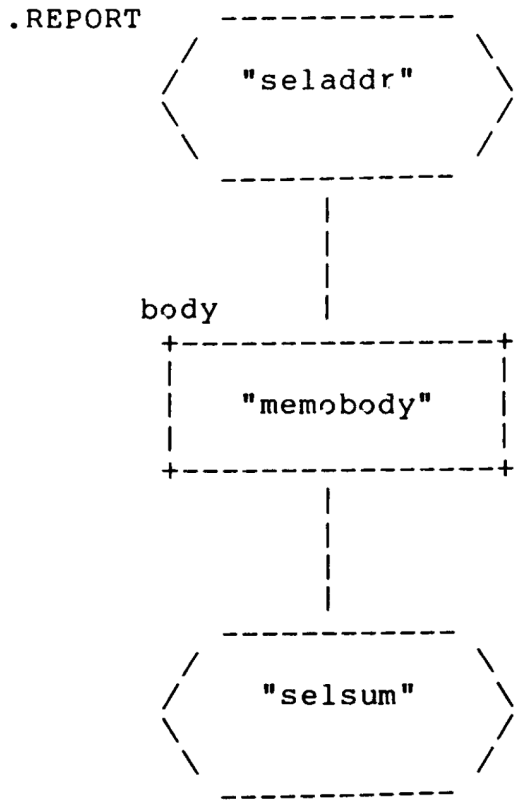
" Example 3 - Sample Letter Program Listing "

Figure 4.3B - Part 1 of 2

```
051: .REM            D e f i n e   P r o c e d u r a l   M a c r o s
05   .REM
05   .REM    ****************************************************************
054: .DEFINE  memobody
055:          #np
056:          Date :
057:          .PRINT curdate
058:          #s 2
059:          To:
060:          .PRINT addressee
061:          #b Department :
062:          .PRINT deptno
063:          -
064:          .PRINT dname
065:          #b Location :
066:          .PRINT location
067:          #s 2  From : Bill James #p \ \ Director of Personnel
068:          #s 2  Subj : 1982 Employee Compensation Plan
069:          #s 2
070:          Its budget time again! To aid you in completing the salary portion
071:          of your budget I have computed your department's current salary
072:          expenses. For planning purposes we are presently estimating an
073:          across the board increase of 10% in the 1982 salary pool.
074:          #s 2
075:          .REM
076:          .REM         S E L E C T  Department's Salary Total
077:          .REM
0             .EXECUTE selsum
079:          #i 10 Your department's 1981 figure is:
080:          .PRINT sumsal
081:          #b Estimated 1982 figure is:
082:          .PRINT sumsal+
083:          #s 3
084:          #te
085:          In completing your salary plan you should break this total down by
086:          individual employees. Please let me know if you have any questions
087:          on this or related issues.
088:          #s 3
089:          Bill
090: ..
091: .REM ***************************************************************************
092: .REM
093: .REM          P R O C E D U R E          S E C T I O N
094: .REM
095: .REM ***************************************************************************
096: #dt 1 13 73 #
097: #t 1
098: #page 8 56
099: .REPORT seladdr memobody
100: .STOP
```

"Example 3 - Sample Letter Program Listing"

Figure 4.3B - Part 2 of 2



"Example 3 - Sample Letter "

Figure 4.3C

The .REPORT statement on line 99 drives the generation of these multiple memos. Only a body macro "memobody" is specified. Since no head macro was included, the body will be executed for the first row returned. The structure of the body is similar to the RPF examples presented in Section 3. Interspersed within the text are PRINT statements which cause the appropriate database information to be included within the memo output. The EXECUTE statement on line 78 executes the 'selsum' to compute the salary sums. The '#p' command and double back slash '\' on line 67 are used to force the author's title to be alligned under his name. Each '\' causes one additional blank to be included. The '#np' on line 55 causes each copy of the memo to be printed on a separate page.

R E P O R T - W R I T E R - U T I L I T Y

(R P T)

5.1 **Overview**

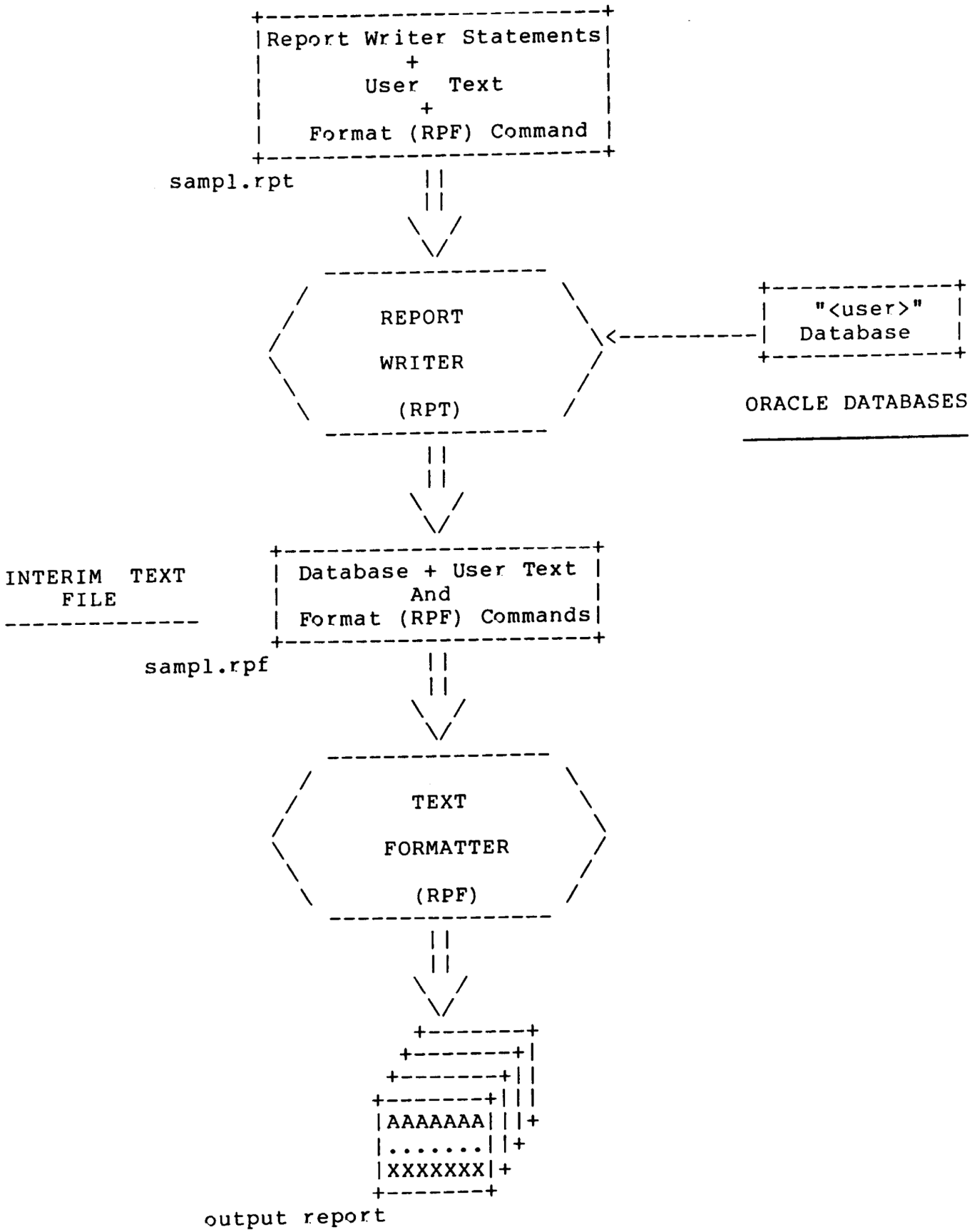
The ORACLE Report Writer Utility (RPT) interprets and executes a report generation program. A **Report Program** may consist of report writer statements, RPF commands, and user text. The use of RPF to format text is described in an earlier section, and a working knowledge will be assumed.

This section will concentrate on the statements necessary to construct a report program. The structure of a program will be described along with a detailed discussion of each programming statement.

5.2 **Report Generation Process**

Figure 5.1 is an overview of the report generation process. The report program is created using a standard text editor. This file is passed as input to the RPT utility.

The program file may contain report text and RPF commands in addition to the RPT program statements. The text and RPF commands are ignored by RPT and are copied as encountered directly to the "Interim File". On the other hand, Report Program statements are interpreted and executed. The main purpose of the report program is to direct the retrieval of database information, and properly place that data into the "Interim File". Each data item which RPT places into the output file is treated by RPF as a "word". This "Interim File" must subsequently be processed by the RPF utility.



" Report Generation Process "

Figure 5.1

In this manner database information is merged with text to form the finished report. Text can be included for report titles, column headings, descriptive information, or the body of a letter for computer generated correspondence. RPF commands can be used to control text and data placement into a tabular format, spacing, underlining, margin control, and page numbering. Any valid RPF command is permitted in a report program.

5.2.1 **Executing RPT**

The RPT utility may be executed from the user's terminal or scheduled within a batch procedure. The utility is executed with the following command:

```
RPT <input file> <output file> [userid/password] [-c]
```

<input file>	Is the name of the file which contains the RPT report program.
<output file>	Name to be assigned to the "Interim File" created by RPT and subsequently processed by RPF.
userid / password	This parameter is required only if the ORACLE database to be processed is secure.The 'userid' must have been defined using the SQL Define User Statement, and have been granted 'read' privileges to the desired data.
-cN	Where N specifies the size of the SQL work area to be requested for each SQL query defined in the report. If omitted, a default value of 3K bytes will be used.

5.3 **Report Structure**

The RPT programming language is similar in many ways to conventional languages like COBOL or PL/I. Although no explicit declaration of program sections is required (ie. COBOL Data Definition Section) a logical grouping of statements is helpful in providing clarity. The three sections of a RPT program are:

- Data Declaration Section
- Macro Definition Section
- Procedure Section

The Data Declaration Section contains the definition of the user's ORACLE database, and "local" variables to temporarily store retrieved database information. Variables may also be defined for counters, to store totals, and as temporary storage.

Two types of "macros" may be defined within a Macro Definition Section. The "SELECT" macro is used to define a SQL select statement. A "Procedural" macro is similar to a COBOL program subroutine, and is a collection of executable RPT statements.

The Procedure Section contains executable RPT statements which comprise the main body of a report program. Within this section "SELECT" and "Procedural" macros can be either explicitly or implicitly executed.

Again, these are purely logical sections; no language statements exist to define the beginning of one or the end of another. There are virtually no restrictions on where RPT statements may be used within a program. Variables however, must be defined before they are used. RPF commands and report text may be included anywhere within a program, and will be copied into the "Interim File" as encountered within the program's execution.

5.4 RPT Language Statements

This section will describe each RPT statement. Figure 5.2 is a summary of these statements.

The general form of a report statement is:

.<command> <arg1> <arg2> ... <argn>

Report statements begin with a period (".") immediately followed by the RPT command. <command> must be one of the commands listed in Figure 5.2. Although commands and reserved words will be specified in upper case, either upper or lower case is permitted.

A RPT command may begin anywhere on the input line. Each command occupies the entire line; report text or other commands may not be included. All the arguments associated with a command must be specified on the same line.

A command may have one or more arguments. Arguments are specified positionally in the defined order. Each argument must be separated by at least one blank space. Arguments are specified by replacing <arg1> <arg2> ... <argn> with either a numeric or alphabetic character string.

Some arguments may contain blank spaces. If one or more imbedded blanks are included the entire argument must be enclosed in double quotes. For example

```
.ASK "Please enter request date : " date
```

contains two arguments; the message "Please enter request date : " and the variable 'date'. The double quotes serve to delimit the beginning and end of an argument, and are not considered a part of the string. Arguments which may contain blanks are denoted with optional quotes (["]<arg>["]).

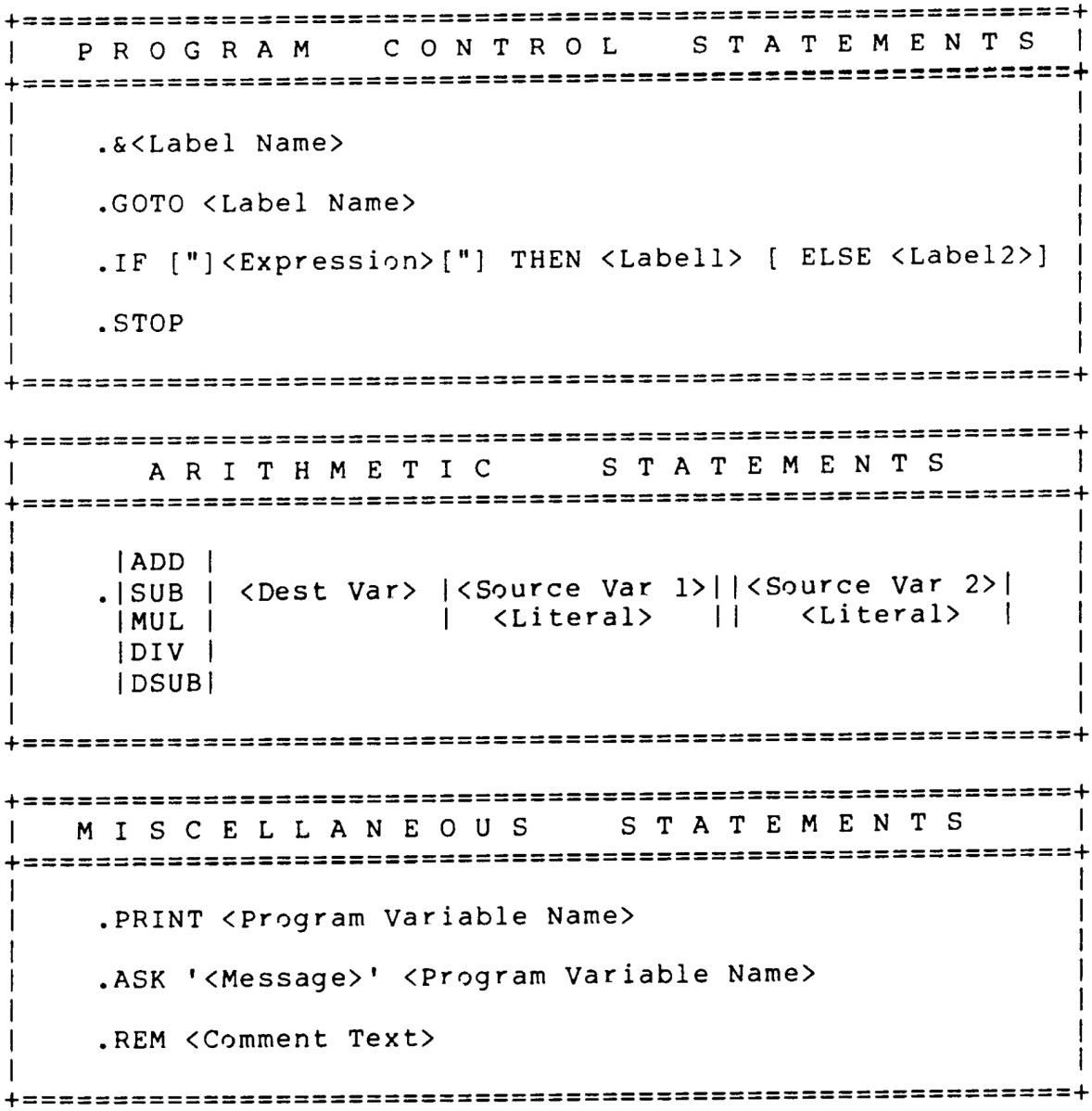
Arguments enclosed within square brackets "[ ]" are optional; those enclosed within vertical bars "| |" indicates a choice of one.

```
+=============================================================+
|          D E C L A R A T I V E   S T A T E M E N T S        |
+=============================================================+
|                                                             |
|     .DATABASE <database name>                               |
|                                                             |
|     .DECLARE <program variable name> <format>               |
|                                                             |
|     .SET <program variable name> <literal value>            |
|                                                             |
|     .EQUAL <destination variable> <source variable>         |
|                                                             |
+=============================================================+

+=============================================================+
|     M A C R O   D E F I N I T I O N   S T A T E M E N T     |
+=============================================================+
|                                                             |
|     .DEFINE |<SELECT Macro Name>     |                     |
|             |<Procedural Macro Name>|                     |
|                        .                                    |
|                  Macro Text Lines                           |
|                        .                                    |
|                        .                                    |
|     ..                                                      |
|                                                             |
+=============================================================+

+=============================================================+
|    M A C R O   E X E C U T I O N   S T A T E M E N T S      |
+=============================================================+
|                                                             |
|     .<Procedural Macro Name>                                |
|                                                             |
|     .EXECUTE <SELECT Macro Name>                            |
|                                                             |
|     .REPORT    <SELECT Macros >  <Body Macro>               |
|                        [<Head Macro> [<Foot Macro>]]        |
|                                                             |
+=============================================================+
```

RPT Statements - Figure 5.2 (Part 1 of 2)



RPT Statements - Figure 5.2 (Part 2 of 2)

5.4.1 Declarative Statements

5.4.1.1 DATABASE

This statement specifies the ORACLE database to be processed within this report. Only one database may be processed per report program.

.DATABASE <database name>

<database name>	is the name of any valid database. If the database is secure a userid/password must be supplied at RPT execution time as part of the RPT command line.

5.4.1.2 DECLARE

This statement is used to declare a program variable and the edit format for printing. All program variables are initialized to null and must be referenced in other statements.

.DECLARE <var name> <format>

<var name>	Is the name assigned to this program variable. The name may be from 1 to x characters, with the first character alphabetic.
<format>	is the edit format used when outputting this variable to the "Interim File". The format will also control rounding and overflow when used in arithmetic statements. A variable's data type is indicated by its format specification. Three data type are currently supported; alpha-numeric, numeric, and date.

Date

A 'Date' variable is defined by specifying a format of 'DATE'.

The output format of a 'date' type variable is MM/DD/YY. Internally, a 'date' variable is a numeric variable which contains the associated date in an absolute julian day number. This approach allows one date to be subtracted from another using a standard arithmetic subtraction function (see section 5.4.5.5 - DSUB). For example:

01/20/81 - 12/29/80 = 22

provides the numeric result of 22. Numeric values may also be added to or subtracted from a date. This is useful in printing invoices where the due date may be computed to be the current date plus 30 days.

A 'date' variable may have its value initialized in a number of ways. It may be assigned a literal value of the form MM/DD/YY, or may be set equal to the value of another 'date' variable. If its value is assigned as a result of a column returned in a SQL SELECT, the database value must be in internal 'Julian Day' format. Presently, the only method for storing internal date data items within a database is by entering the data using an IAF application.

To create a 'date' compatible column, the column must be defined as a 'number' in the SQL CREATE TABLE statement. The column's value must be initially entered and always updated using an IAF application, where the field type is defined as 'date'.

Alpha-numeric

An 'Alpha-numeric' variable may contain any printable character, and is defined by specifying a format of 'An' (where n = number of characters). When a report is executed all alpha-numeric variables are assigned the 'NULL' value.

Numeric

'Numeric' variables are specified using the following symbols:

9 - defines each digit of a numeric variable. Leading zeros are not displayed.

. - Defines the position of the decimal point within a numeric variable. The position is used for arithmetic alignment and is displayed on output.

, - Causes a comma to be inserted on output. Omitted on output if there are no digits to the left of this position.

$ - causes a dollar sign to precede the number on output.

MI - Causes the minus sign to be displayed to the right of a negative number. The default is to the left.

PR - Causes the variable to be displayed within "< >" brackets when negative.

0 - May be used instead of a 9 to designate a digit. Normally leading zeroes are suppressed, however a zero in the format will cause every digit position to be filled.

V - Defines the position of the decimal point within a numeric variable. The position is used for alignment in arithmetic statement, but the decimal point is not displayed on output.

B - Causes the variable to be output as blanks if its value is zero.

The following are examples of various formats:

Format	Value	Displayed
999.99	56.478	56.48
999V99	56.478	5648
9,999	8410	8,410
9,999	639	639
99999	607	607
09999	607	00607
9999	-5609	-5609
9999MI	-5609	5609-
9999PR	-5609	<5609>
B999	564	564
B999	0	blanks
99.99	124.98 (1)	##.##
		24.98
$99.99	45.23	$45.23
DATE	2441453 (2)	12/23/80
A5	Customer	Custo
A20	Customer	Customer

(1) - If the value retrieved into this variable from the database is greater than can be displayed by the format, #'s will be displayed. If the variable is overflowed due to an arithmetic operation, a truncated value will be displayed.

(2) - Julian day number for 12/23/80

5.4.1.3 **SET**

This statement sets the value of the variable equal to the specified literal value.

.SET <variable name> <literal value>

<variable name>	any previously defined program variable.
<literal value>	numeric or character literal. The literal type must match the variable type. The special literal "$$DATE$$" may be used to assign the system date to a date variable.

.SET name JONES - sets the current value of the variable 'name' equal to JONES.

.SET empno 5647 - sets the current value of the numeric variable 'empno' equal to 5647.

.SET today $$DATE$$ - sets the value of the date variable 'today' equal to the current date.

5.4.1.4 **EQUAL**

This statement will set the value of one variable equal to the value of another variable. Both variables must be of the same data type.

.EQUAL <dest var> <source var>

The <dest var> will be set equal to the value of the <source var>.

For character variables, the value of the <source var> will be truncated if longer than the <dest var>, and blank filled if shorter.

For numeric variables, if the format of the <dest var> contains fewer decimal places, the value of the <source var> will be rounded. No provisions are made for overflow.

If the value of the <source var> cannot be stored within the format of the <dest var> variable, the value will be truncated in the destination.

5.4.2 **Macro Definition Statements**

RPT recognizes two types of macro statements; SELECT and Procedural. Both types are defined in the same manner, and RPT will distinguish them by the way they are invoked and the type of statements they contain.

5.4.2.1 **DEFINE**

This statement is used to define a SELECT or Procedural Macro. Execution of this statement stores the macro away for future use. Nothing is output to the interim file. Notice that ".." (two periods) on a line by themselves are used to complete the macro definition.

```
.DEFINE |<SELECT Macro Name>      |
        |<Procedural Macro Name>|
              .
      Macro Text Lines
              .
              .
              .
..
```

<SELECT Macro Name> <Procedural Macro Name>	Name of the macro being defined.
..	Ends the Macro definition.

5.4.2.2 **SELECT Macro**

A SELECT Macro contains the text of a SQL query. Only one query may be specified within each SELECT macro. These queries are used to retrieve the data which will be included within the report. The macro may include any SQL clause or parameter which is valid within a SELECT statement. In this manner the full power of the SQL query language may be used to extract the database information.

In addition to the standard SQL clauses, an INTO clause must also be included. This clause specifies the program variables which will receive the column values returned in the SELECT clause. For example a report program has three variables defined; alpha, beta, and gamma. If the following SELECT macro named 'sample' were executed:

```
.DEFINE sample
        SELECT empno,ename,loc
        INTO   alpha,beta,gamma
        FROM   emp,dept
        WHERE  emp.deptno=dept.deptno
               and sal > 5000;
..
```

the values returned for 'empno', 'ename', 'loc' are stored in the program variables 'alpha','beta','gamma' respectively. The program variable must be of the same data type as the column or expression in the SELECT clause. It should be noted that an INTO clause, as well as any SQL clause, may be defined in a 'free format', and the structure used here is for readability purposes only.

The value of a program variable may be substituted for any literal defined in the WHERE or SELECT clause of a SQL query. When used in this manner, the variable name must be preceded with an '&'. The variable name may not include an underscore character '_'. For example, if the following SELECT macro:

```
.DEFINE seldept
        SELECT deptname,location
        INTO   dname,loc
        FROM   dept
        WHERE  deptno = &dno
..
```

were executed with 'dno' equal to 20, the value of 20 would be substituted into the WHERE clause, and the 'name' and 'location' of department 20 would be stored into the program variables 'dname' and 'loc'. Both the program variable and the database column must be the same date type.

From the above examples, it can be seen that the results of one query may be stored into a program variable, and used as a substitution variable within another query. This technique is used to construct queries which produce nested reports. Nested reports are discussed in Section 5.4.3.3.1. The following two SELECT statements would be used to generate a report of each course in the 'PHYSICS' department, and for each course a list of students and their grades:

```
.DEFINE courses
        SELECT cname,cnumber,cdate
        INTO   cname,cnumber,cdate
        FROM   course
        WHERE  cdept = 'PHYSICS'
..

.DEFINE students
        SELECT sname,grade
        INTO   sname,grade
        FROM   students
        WHERE  course  =   &cnumber
..
```

5.4.2.3 **Procedural Macro**

A 'Procedural Macro' is similar to a programming language subroutine. It may contain both RPT language statements, and user text and RPF commands. When invoked, the RPT statements within the macro are executed, and user text and RPF commands copied to the 'Interim File'.

A macro may not be **defined** within a macro. Macros may be invoked from within other macros. The following procedural macro outputs the heading of a report. The .PRINT statement outputs the value of the variable 'year', and is explained in a later section.

```
.DEFINE heading
        #t 2
        #cul Corporate Finance Report #
        #s 2
        #cul Fiscal Year
        .PRINT  year
        #
..
```

5.4.3 **Macro Execution Statements**

5.4.3.1 **Procedural Macro Execution**

A procedural macro may be explicitly executed anywhere within an RPT program including from within another procedural macro. The named macro is executed by specifying:

```
.<procedural macro name>
```

The macro must have been defined in the program prior to its execution. The statement:

```
.summary
```

will cause the procedural macro named 'summary' to be executed. Following the macro execution the next sequential statement is executed.

5.4.3.2 **SELECT Macro Execution**

A SELECT macro may be explicitly executed with the following RPT statement:

```
.EXECUTE <SELECT macro name>
```

The EXECUTE statement will cause the specified SELECT macro to be executed. Following the execution of the query, the first row will be returned. The values of the selected columns will be placed into the corresponding program variables as defined on the INTO clause.

Executing a query in this manner will always return only the first row. If the macro is re-executed, the entire query will be reprocessed, and again the first row is returned.

Explicit execution of SELECT macros is useful where only one row is returned. For example, if the average sal for a department was to be reported in a summary section, the following query could be executed to return this result:

```
.DEFINE deptavg
        SELECT avg(sal)
        FROM   dept
        INTO   avgsal
        WHERE  deptno = &dno;
..

        .
        .
        .
.SET   dno 30
.EXECUTE deptavg
.PRINT avgsal
```

Another use of the EXECUTE would be to look up a customer's name and address, based on the customer number, for a computer generated form letter. The following example would accomplish that task:

```
.DEFINE custaddr
        SELECT custname,custaddr
        INTO   name,addr
        FROM   custlist
        WHERE  custno = &customer
..
         .
         .
         .
.EXECUTE custaddr
.PRINT name
#n
.PRINT addr
#n
```

5.4.3.3. **REPORT Statement**

The **REPORT** statement causes the automatic execution of SELECT and procedural macros which drives the generation of most reports. Unlike the EXECUTE statement, the REPORT statement will cause every row returned from the SELECT to be processed. For each row the specified procedural macros are executed. Figure 5.3 depicts the structure of a REPORT statement.

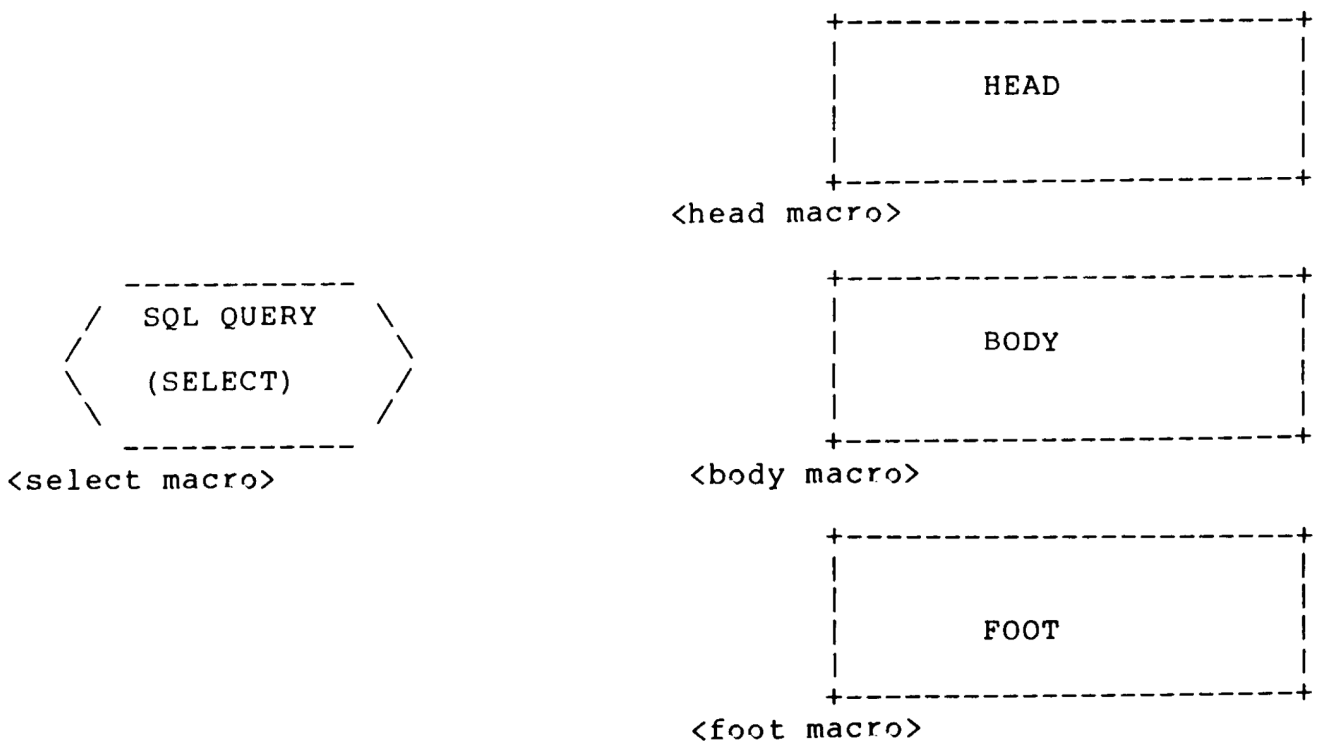
The three procedural macros correspond with the head, body, and foot of a report. The **head macro** is executed once within a REPORT statement, when the first row is returned. Included within this macro would be the column headings, descriptive text, and report titles. Since the body macro is not executed for the first row returned (if the heading argument is included) you should either execute the body macro as part of the heading or make other arrangements to print the first row. This allows flexibility in setting up reports since it may be desirable to handle the first row differently than succeeding rows.

The **body macro** is executed for the second through the last rows. If a foot macro was not specified, the body would also be executed for the first row. Its function is to output each row of data within the desired format. Other functions could be to accumulate totals, maintain counters, and control page breaks.

The **foot macro** is executed after the last row of the query has been processed. Within this macro summary calculations and footnotes could be included.

The head, body, and foot are standard procedural macros and may contain any valid RPT statement. Within these macros, other macros can be executed. For example, within the foot of a report the following SELECT could be executed to compute salary statistics for the department 10:

```
.DEFINE summary
        SELECT max(sal),min(sal),avg(sal),sum(sal)
        INTO   maxsal,minsal,avgsal,sumsal
        FROM   emp
        WHERE  deptno=10
..
```



.REPORT <select macro> <body macro> [<head macro> <foot macro>]

REPORT Statement Structure

Figure 5.3

5.4.3.3.1 **Nested Reports**

A REPORT statement can be executed within the head, body, or foot macros of another REPORT. Figure 5.4 shows the structure of such a report.

In this example, the departments within a company are reported. For each department, a list of the projects within that department is generated. Within the body of the 'department' report, is a REPORT statement to generate the projects report.

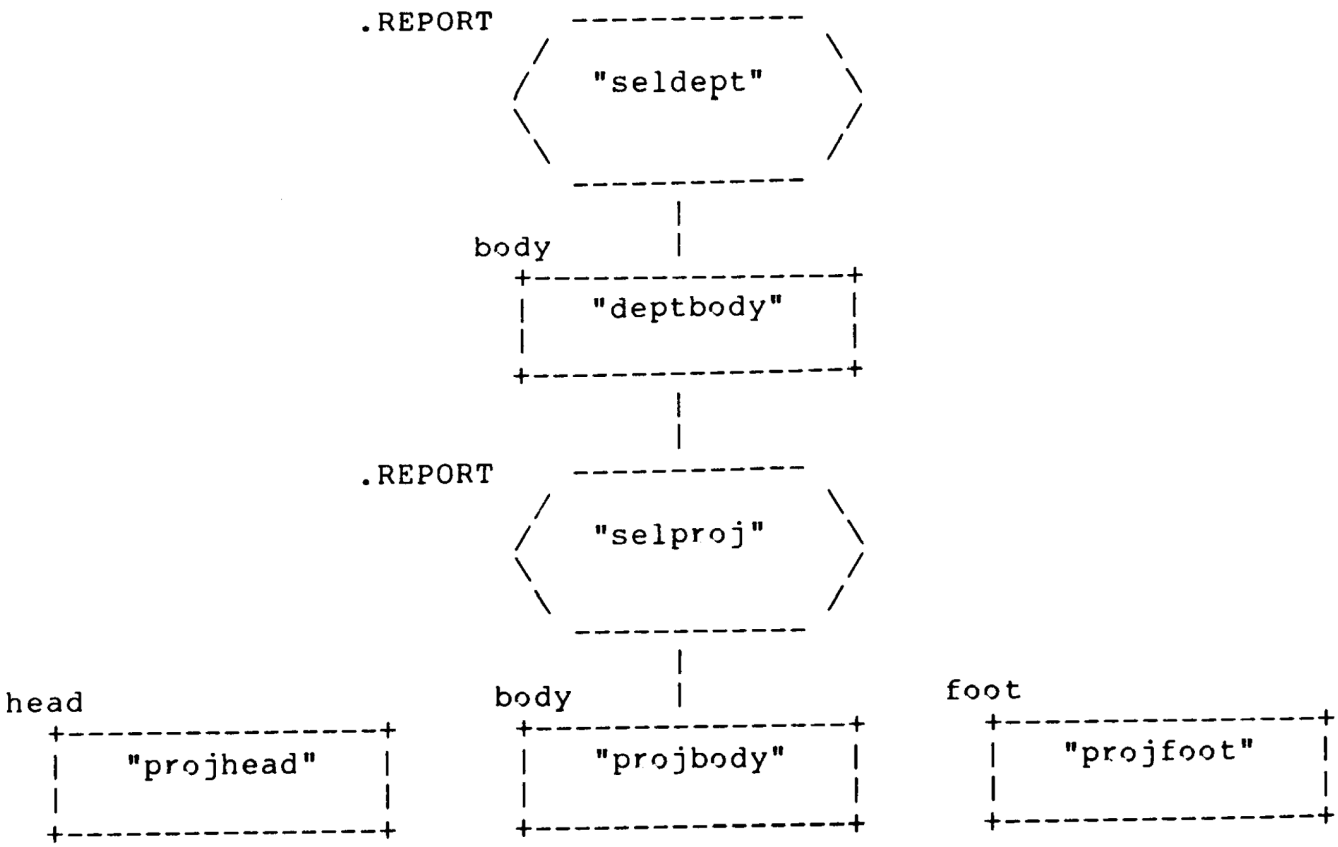
Multiple levels of **nested** REPORT statements may be constructed. Additionally, multiple REPORT statements may be included within a head, body, or foot of another report. Figure 5.5 shows the structure and format of an 'employee personnel' report, listing for each employee, the employee's job history, salary history, and project assignments.

5.4.3.3.2 **Disjunctive Reports**

Two or more SELECT macros may be specified in a REPORT statement. In this case multiple SELECT's will be executed, and a row from each returned. The head, body, and foot macros would be executed exactly as with a single SELECT, however data from multiple selects may now be printed. The rows from each SELECT are returned in step with each other.

The data retrieved from multiple selects may be entirely independent, referencing different tables. For example, an employee's job history may be retrieved by one SELECT, while a list of the employee's current projects by another. This data could be printed side by side in separate columns across the page as shown in figure 5.6.

The multiple SELECT macros may be specified with either an 'AND' or an 'OR' between each select macro. 'AND' indicates that the report should be executed only if both selects (on either side of the 'AND') return at least one row. An 'OR' indicates that the report should be executed if at least one of the selects (on either side of the 'OR') returns at least one row. Note that if only one SELECT is specified in a report and no rows are returned, the report will not be executed. Also note that if two or more selects are specified the select argument will need to be enclosed in quotes as in "sel1 AND sel2".



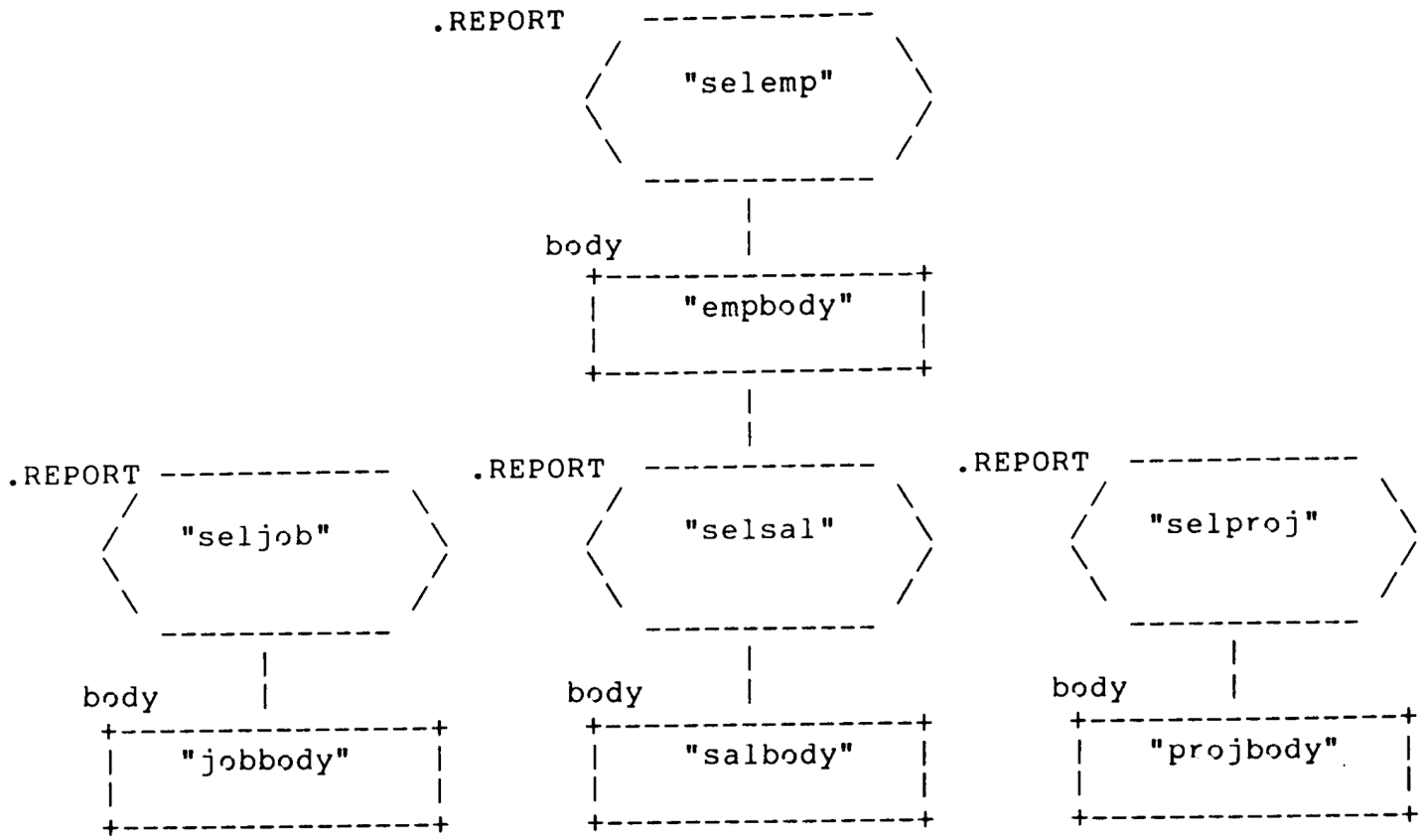
DEPARTMENT / PROJECT REPORT

Department Number	Department Name	Department Location
067	Circuit Dev	Boise

Project No.	Project Name	Comp Date
563	Board Design	04/30/81
894	Chip Manufact	06/01/81

NESTED REPORT

Figure 5.4



EMPLOYEE PERSONNEL REPORT

Employee Number	Employee Name
8945	Martin, R.J.

Job Title	Start Date
Prog Trainee	06/14/72
Programmer	12/06/75
Analyst	05/23/77

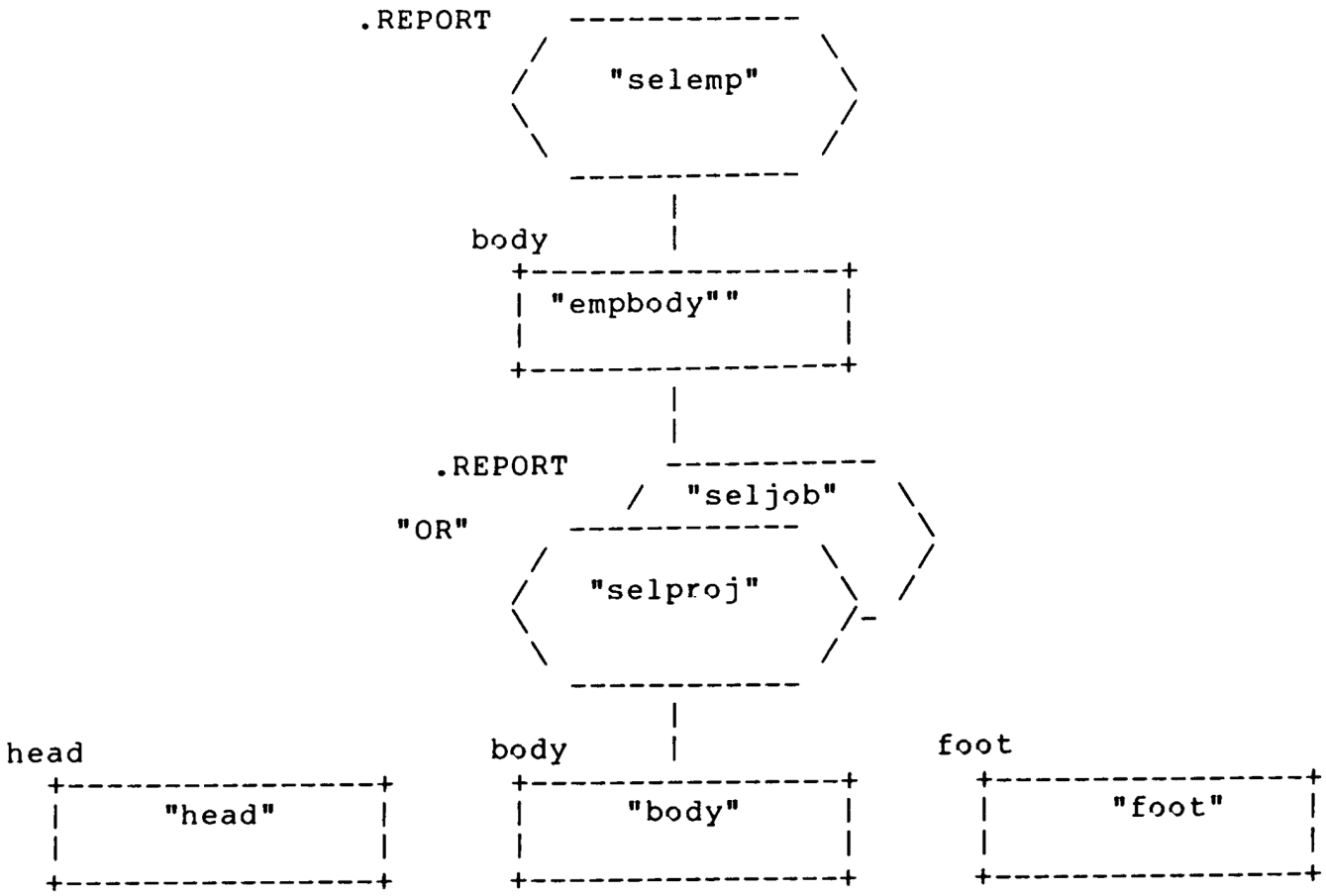
Salary	Start Date
$9,600.00	06/14/72
$13,200.00	02/01/74

Project Assignments

456 Sales Report System

MULTIPLE REPORTS WITHIN A REPORT

Figure 5.5



E M P L O Y E E R E P O R T

Employee Number	Employee Name	Job History		Current
		Title	Date	Projects
4028	Murphy, P.E.	Pgmr Lvl1	06/72	Design 33
		Pgmr Lvl2	12/74	Doc 45
		Analyst	07/75	
		Sr. Anal	08/78	

" DISJUNCTIVE REPORT "

Figure 5.6

5.4.3.3.4 **REPORT Statement Format**

The format of the REPORT statement is:

.REPORT <SELECT macros> <body macro> [<head macro> <foot macro>]

<SELECT macros>	name of SELECT macro executed for this report. If two are specified, they must be joined by an 'AND' or 'OR' and enclosed within double quotes.
<body macro>	name of the body macro which will be executed for the 2nd thru last returned rows.
<head macro>	name of the head macro which is executed for the first row returned. This parameter is optional.
<foot macro>	name of the foot macro which is executed after the last row is returned. This is an optional parameter and can only be specified if a head macro is also specified.

5.4.4 **Program Control Statements**

RPT provides statements to control the program execution within a **procedural** macro. Only the .STOP statement may be used outside of a procedural macro.

5.4.4.1 **Label Definition Statement**

A label may be defined within a procedural macro using the following statement:

.&<label name>

<label name>	From 1 to 8 character name; first character must be alphabetic.

A label may be referenced only within the macro in which it was defined. Label definitions do not span macros. Since label definitions are local, the same label name may be used in multiple macros.

5.4.4.2 **GOTO Statement**

The **GOTO** statement causes an unconditional branch to the specified label. The format of this statement is:

.GOTO <label name>

<label name>	The name of a label defined within the current procedural macro.

Figure 5.7 lists a sample program which demonstrates the use of the GOTO statement.

5.4.4.3 **IF Statement**

This statement causes a branch to the specified macro label depending on the result of an expression. The IF statement is only valid within a procedural macro. The format is:

IF <expression> THEN <label1> [ELSE <label2>]

<expression>

An <expression> may compare the value of a program variable with other program variables or literal constants. Program variable names must be preceded with an ampersand '&'. If the '&' is omitted an "Invalid Column Name" message will be displayed.

The IF statement supports the complete set of relational and logical operators, and arithmetic expressions permitted within a WHERE clause of a SQL statement.

Character constants must be enclosed within single quotation marks. All literal values and program variables within an expression must be of the same data type.

```
.REM    ************************************************************
.REM    **********      EXAMPLE OF "GOTO" STATEMENT       *********
.REM    ************************************************************
.REM
.REM                        Define Macro
.REM
.DEFINE macro1
        .GOTO label1
        .&label2
           two
        .GOTO end
        .&label1
           one
        .GOTO label2
        .&end
..
.REM
.REM                        Execute Macro
.REM
.macro1
```

==========

Generated Output

```
one two
```

"Sample GOTO Program"

Figure 5.7

If the literals, program variables, and relational operators are separated by one or more blank characters, the entire expression must be enclosed within double quotation marks(").

A program variable used in an expression may not have the NULL value. If the value is NULL, the report program will be terminated with an error. To prevent this situation all variables should be initialized to either zeros or blanks.

If the variable is assigned a value from a database column which allows NULLs, the null value function (NVL) should be used to assign a non-NULL value. For example the variable salary used in following IF statement:

```
.IF " &salary > 10000 " THEN ...
```

then the SELECT macro should define the NVL function in the SELECT list:

```
SELECT NVL(sal,0)
INTO salary
FROM ...
```

If any variable participating in the expression has the null value the ORACLE error: "unexpected end of sequel statement" will be displayed.

The following are valid expressions:

```
&name='SMITH'

&salary*2<4500

"&dept=10 or &div='MOTOR'"

" &sal > 5000 and &job = 'plumber' "
```

<label1>

If the <expression> evaluates to 'True', processing control will be transferred to <label1>

<label2>

If the <expression> evaluates to 'False', processing control will be transferred to <label2>. If "ELSE <label2>" is omitted, and the expression is 'False', the next sequential statement is executed.

```
.REM   ************************************************************************
.REM   **********     EXAMPLE OF  "IF  THEN"  STATEMENT                 ********
.REM   ************************************************************************
.DATABASE IAFDEMO
.REM
.REM                        Define   Loop    Variable
.REM
.DECLARE x   999
.REM
.REM                              Define     Macro
.REM
.DEFINE macro1
         .&label1
         Loop Counter =
         .PRINT x
         .ADD x x  1
         .IF "&x <= 5 "  THEN label1
         End of Loop
.
.REM
.REM                             Execute     Macro
.REM
.macro1
```

==========

Generated Output

```
Loop Counter =  1
Loop Counter =  2
Loop Counter =  3
Loop Counter =  4
Loop Counter =  5
End of Loop
```

▪ Sample IF THEN Program ▪

Figure 5.8

5.4.4.3.1 IF Statement Guidelines

IF statement processing is fairly time consuming, and should therefore be used cautiously within a report program. It can be used effectively for controlling the overall flow of a report, where the statement will be executed infrequently. For example, depending on the current date or a terminal input variable, certain sections of a report may be included or excluded. The IF statement(s) controlling these conditionally executed sections would be executed once for the entire report.

In contrast, executing an IF statement for each row processed, could severely lengthen the report execution time if a large number of rows were processed. For example, although RPF supports page control, there is no mechanism to reprint page headings for each page break. One method of reprinting the headings would be to count each line which is generated, then at the top of page execute a special heading routine. However, for each line printed an IF statement would have to be executed to test for an end of page condition. This use of the IF statement is not recommended for reports of substantial length.

Some uses of the IF statement can be replaced with SQL functions. For example computing the maximum and minimum value of a variable over a range of rows could be accomplished by comparing each new value with the previous maximum and minimum values. This requires two IF statements for each row. Another approach would be to execute a SELECT in the report 'foot' which used the MIN and MAX SQL functions. The second approach would be more efficient.

5.4.4.4 STOP Statement

The STOP statement will terminate the execution of the report program. STOP may be included in the procedure section or a procedural macro. The format of the statement is:

```
.STOP
```

If a STOP is not included, the program will terminate following the execution of the last program statement.

5.4.5 **Arithmetic Statements**

RPT provides statements to perform addition, subtraction, multiplication, and division between two program variables. The results of the operation are stored in a third variable. The formats are:

```
 |ADD |
.|SUB |  <dest var>  <source var1>  <source var2>
 |MUL |
 |DIV |
 |DSUB|
```

<dest var>
<source var1>
<source var2>

For all arithmetic statements except DSUB, a numeric literal may be substituted for the input arguments <source var1> and <source var2>.

These numeric variables must have been previously defined. The maximum number which can be stored in a variable is determined by the format specified on the DECLARE statement. If the result of the arithmetic operation overflows the <dest var>, the high order digits will be lost, and no error will be indicated. These digits are lost both on output and in subsequent arithmetic operations.

For example, if the result is '456' and the format of <dest var> is '99', '56' is stored in <dest var> and the high order digit '4' is lost.

If the decimal portion of the arithmetic result contains more digits them defined in the <dest var> format, the low order digits will be rounded off. For example, if the the result was '46.576' and the format of the <dest var> was '99.99', the number '46.58' will be stored.

5.4.5.1 **ADD**

The value of <source var1> is added to <source var2> and the result is stored in <dest var>. For example, if X=5 and Y=6 then:

```
.ADD Z X Y
```

will set Z=11. The result may be stored into one of the source variables with:

```
.ADD X X Y
```

results in X=11.

5.4.5.2 **SUB**

The value of <source var2> is subtracted from <source var1> and the result is stored in <dest var>. For example, if X=5 and Y=3 then:

```
.SUB Z X Y
```

will set Z=2. The statement:

```
.SUB X X Y
```

set the value of X=2.

5.4.5.3 **MUL**

The value of <source var1> is multiplied by <source var2> and the result is stored into <dest var>. For example, if X=5 and Y=6 then:

```
.MUL Z X Y
```

will set Z=30. The statement:

```
.MUL X X Y
```

sets the value of X=30.

5.4.5.4 **DIV**

The value of <source var1> is divided by <source var2> and the result is stored into <dest var>. For example, if X=10 and Y=2 then

```
.DIV Z X Y
```

sets the value of Z=5. The statement:

```
.DIV X X Y
```

sets the value of X=5. If a number is divided by zero (ie. <source var2>=0), a "** DIVIDE BY ZERO (X/Y) **" error message will be issued and the report program will be terminated.

5.4.5.5 **DSUB**

This statement subtracts one 'date' variable from another, storing the result into a 'numeric' variable. This allows the number of days between two dates to be computed. For example, if DATE1=01/24/81 and DATE2=12/25/80 then

```
.DSUB RESULT DATE1 DATE2
```

set the value of RESULT=30. Note that neither DATE1 nor DATE2 may be literals. 5.4.6 **Miscellaneous Statements**

5.4.6.1 **PRINT Statement**

The PRINT statement inserts the contents of the specified program variable into the output "Interim File". This is the only mechanism for inserting database information into the output report. The content of the variable will be formatted according to the format defined on the DECLARE statement. The data will be treated as a separate word when processed by RPF.

The format of the statement is:

```
.PRINT <variable name>
```

PRINT statements may be interspersed with RPF commands to print the data in the various columns of a tabular report. Refer to the Sample Reports in Section 5 for examples of the use of PRINT.

5.6.2 **ASK Statement**

The ASK statement displays a message on the user's terminal, and allows the user to enter a value to be assigned to the specified program variable. ASK provides a means for the user to dynamically control the flow and output of a report.

The format of this statement is:

```
.ASK "<message>" <variable>
```

<message> is a 1 to x character message which will be displayed on the user's terminal. If the message contains blank characters, the message text must be enclosed within double quotes(").

<variable> is the name of the program variable whose value will be set equal to the data value entered by the user. The entered data value must be of the same data type as the program variable.

If a numeric variable is specified, and the entered data is alphanumeric, the variable is set equal to zero and no error is indicated.

If a 'date' variable is specified, the format of the input is MM/DD/YY. The date routine will verify that the entered date is valid. If invalid, the operator will be requested to re-enter the data.

5.4.6.3 **Remark Statement**

The REM Statement allows the report programmer to include comment lines within the program source file. The entire line of text following the REM statement is treated as a comment and ignored by RPT. The remark statement will not be output to the interim file. The format of this statement is:

.REM <comment text>