TPL Primer
An introduction to Table Producing Language

by
Barbara B. Noble

for
U. S. Department of Justice
Law Enforcement Assistance Administration
National Criminal Justice Information and
Statistics Service

Under Grant No. 78-SS-AX-0028

BUREAU OF SOCIAL SCIENCE RESEARCH, INC.
1990 M Street, N. W.
Washington, D. C. 20036

June 27, 1979

TPL Primer
An Introduction to Table Producing Language

INTRODUCTION

Table Producing Language (TPL) is a set of computer programs which produces printed tables from machine-readable data files. TPL can be used to select data for display, to compute new variables from old values, and to format many different types of statistical tables.

TPL was written by the Bureau of Labor Statistics and converted to run under the Michigan Terminal System by the LEAA Research Support Center.

The purpose of this document is to introduce you to TPL and to provide information sufficient to produce useful tables. A brief explanation, a statement model, and examples of each TPL statement, and a description of the MTS procedures for running TPL tables are included in this primer. It is assumed that a TPL codebook and data file have been prepared for your use.

This primer is not intended to be a complete description of TPL. Complete documentation of TPL, including instructions for preparing a TPL codebook and data file, can be found in Table Producing Language, Version 3.5, Users Guide, prepared by the Bureau of Labor Statistics.

Definitions

A TPL codebook describes a data file to the TPL table generating programs. It is stored in a file separate from the data. The codebook includes information about the name, location, type, and values of each variable in the data file. The sample codebook in the appendix is used in the examples in this primer.

TPL recognizes two types of variables: control and observation variables. Control variables are classification or grouping variables such as sex or type of offense. All values of the control variables in a data file are listed in the codebook.

Observation variables are quantitative variables on which arithmetic can be done. In most cases, it makes no sense to do arithmetic with control variables. Examples of observation variables are amount of bail or number of years of sentence.

When generating a TPL codebook, each type of record in the data file is given a name. In the sample codebook, one record name is defined; DEFENDANT. The record name is used as an observation variable. It has an assumed value of one for each record and can be used to count records.

In a TPL codebook, some variables may be defined as both control and observation variables. This allows some flexibility in using the data. Rules for choosing the type of variable are explained later.

TPL TABLE REQUEST STATEMENTS

Format Of TPL Statements

There are six TPL statements for producing tables: USE, TABLE, SELECT, DEFINE, COMPUTE, and POST COMPUTE. A TPL request must at least include a USE statement and a TABLE statement.

TPL statements are made up of keywords, parameters, punctuation, and some special symbols. A model of each statement shows its format and content. Keywords, such as the names of statements, are shown in uppercase and are to be used as shown. Parameters are shown in lowercase and are to be replaced by information specific to your request. Punctuation and special symbols are also to be used as shown.

A TPL statement cannot extend past column 72. If a statement is longer than 72 characters, it may be extended over more than one line by breaking the command at a convenient blank. End each statement with a semicolon.

USE Statement

The USE statement provides TPL with the name of the codebook to use. It is always the first statement in a table request. The model of the USE statement is:

```
USE codebookname CODEBOOK ;
```

Replace codebookname with the name of your TPL codebook. For example:

```
USE ARREST CODEBOOK;
```

The name of the sample codebook is ARREST. Notice that this is not the name of the file which contains the codebook but the name of the codebook in the file.

TABLE Statement

Format Of The TABLE Statement

The TABLE statement defines the name, title, and content and structure of the table. The following is a model of the TABLE statement:

```
TABLE name 'title' : stub-exp, heading-exp ;
```

Replace name with a letter, a number, or a short label to identify the table. For example, A, TABLE1, or First are all valid names for a table. The table name is used to label the table.

A title for the table is optional. If you want a title, enclose it in single quotes as illustrated.

The stub and heading expressions, stub-exp and heading-exp, specify the structure and content of a table. The stub expression defines the rows and the heading expression defines the columns. The stub and heading expressions are identical in format and are composed of the names of variables and some special keywords.

Let's first consider a simple example to illustrate the TABLE statement.

```
TABLE TABLE1: RACE, SEX;
```

TABLE1 is the name of the table, RACE is the stub expression and SEX is the heading expression. From the codebook in the appendix, we see that SEX and RACE are control variables, and that SEX has two values and RACE, four values. The resulting table looks like this:

TABLE1

	SEX OF DEFENDANT	
	Male	Female
RACE OF DEFENDANT		
White.....		
Black.....		
Oriental.....		
Other.....		

Notice that labels from the codebook are printed instead of the variable names and values.

Table Structure

The structure of a table is primarily determined by the control variables specified in the stub and heading expressions. In the last example, the row structure is determined by the stub expression, RACE, and the column structure by the heading expression, SEX. The table has eight cells with the coordinates White Male (RACE=1, SEX=1), White Female (RACE=1, SEX=2), Black Male (RACE=2, SEX=1), etc.

Table Content: Counts Of Records

Having defined the structure of a table, let's consider content. The content of a table is the values that appear in the cells. For example:

TABLE1

	SEX OF DEFENDANT	
	Male	Female
RACE OF DEFENDANT		
White.....	5	2
Black.....	3	1
Oriental.....	2	-
Other.....	2	-

- DATA NOT AVAILABLE.

The content of this table is how many records there are in the data file with each of the eight combinations of SEX and RACE. In the sample data file in the appendix, you can verify this table by counting the number of records for each combination of values for SEX and RACE.

Table Content: Sums Of Variables

A count of the number of data records which matches each combination of coordinates is one kind of content of a table. The second kind of content is sums or aggregations of an observation variable. The TABLE statement:

```
TABLE TABLE2: RACE, BAIL_AMT;
```

will produce the table:

TABLE2

	BAIL AMOUNT
RACE OF DEFENDANT	
White.....	20,000
Black.....	2,000
Oriental.....	6,000
Other.....	15,000

The heading expression, BAIL_AMT, consists of one variable which is an observaion variable. There is no control variable to affect the column structure of the table so the table contains only one column. The cells of the table contain the sum of the values of BAIL_AMT for each category of RACE. You can prove this to yourself by finding all records in the sample data file where RACE is 1 (White) and adding up the values of BAIL_AMT for those records.

THEN, The Concatenation Operator

A single table can contain sums of more than one observation variable, or counts of records classified by more than one set of control variables, or sums and counts. To do this, use the concatenation operator, THEN, between variable names in either the stub or heading expression or both. For example:

```
TABLE TABLE3: RACE, SEX THEN BAIL_AMT;
```

results in the table:

TABLE3

	SEX OF DEFENDANT		BAIL AMOUNT
	Male	Female	
RACE OF DEFENDANT			
White.....	5	2	20,000
Black.....	3	1	2,000
Oriental.....	2	-	6,000
Other.....	2	-	15,000

\- DATA NOT AVAILABLE.

Notice that this table contains the same information as the first two tables, counts and sums. The first and second tables are put side by side or concatenated to form this third table.

Totals

Adding totals to a table is easy: include the keyword TOTAL in place of a variable in the stub or heading expression. For example:

```
TABLE TABLE4: RACE THEN TOTAL, TOTAL THEN SEX
     THEN BAIL_AMT;
```

produces the table:

TABLE4

	TOTAL	SEX OF DEFENDANT		BAIL AMOUNT
		Male	Female	
RACE OF DEFENDANT				
White.....	7	5	2	20,000
Black.....	4	3	1	2,000
Oriental.....	2	2	-	6,000
Other.....	2	2	-	15,000
TOTAL.....	15	12	3	43,000

- DATA NOT AVAILABLE.

Notice that the keyword TOTAL can appear anywhere in an expression and in both the stub and heading expressions. The keyword TOTAL in the stub expression results in a count of the total number of records in the file, 15; a count of records for each category of SEX, 12 and 3; and the total BAIL_AMT, 43,000, from all records in the file. Similarly, TOTAL in the heading expression results in counts for each RACE group and of all records in the file.

BY, The Nesting Operator

The nesting operator, BY, used between two control variables in an expression, produces all possible combinations of the values of the variables. Consider:

```
TABLE TABLE5: DRUG_USER BY RACE, BAIL_AMT
```

gives the table:

TABLE5

	BAIL AMOUNT
DRUG USER	
Yes	
RACE OF DEFENDANT	
White.....	11,000
Black.....	0
No	
RACE OF DEFENDANT	
White.....	1,000
Black.....	2,000
Oriental.....	1,000
Other.....	5,000
Missing Data	
RACE OF DEFENDANT	
White.....	8,000
Oriental.....	5,000
Other.....	10,000

Three values for DRUG_USER and four values for RACE are nested to produce twelve combinations. Because some combinations are not present in the file, some rows which were empty were not printed.

Combining Nesting And Concatenation Operators

The nesting and concatenation operators, BY and THEN, can be combined in an expression. When these operators are combined in an expression, nesting takes precedence over concatenation. This means that nesting is done first; then the results of nesting are concatenated. For example, the TABLE statement:

```
TABLE TABLE6: TOTAL, SEX BY DRUG_USER THEN BAIL_AMT;
```

generates the table:

TABLE6

	SEX OF DEFENDANT						BAIL AMOUNT
	Male			Female			
	DRUG USER			DRUG USER			
	Yes	No	Missing Data	Yes	No	Missing Data	
TOTAL.....	4	5	3	1	2	-	43,000

- DATA NOT AVAILABLE.

Rules For Using Observation Variables

TPL determines the contents of the table cells. Cells will contain counts of records if no observation variable is mentioned, or will contain sums of an observation variable when an observation variable is specified. Each cell can contain one or the other, but only one. This leads to some restrictions on the use of observation variable names in table expressions.

1. Observation variables can be named in only one table expression; the stub expression or the heading expression.

2. More than one observation variable can be used in an expression if they are separated by the concatenation operator, THEN. The parts of an expression separated by THEN are called terms. The name of an observation variable can appear anywhere in a term. Its placement affects only the location of labels in the table.

Some examples will illustrate these rules. Two control variables, RACE and SEX, and two observation variables, BAIL_AMT and BOND_AMT will be used and only the stub and heading expressions of the TABLE statement will be illustrated.

```
BAIL_AMT, RACE BY BOND_AMT;
```

This is not valid because observation variables are named in more than one expression.

```
RACE, BOND_AMT THEN BAIL_AMT;
```

This example is correct: observation variables are named in only one table expression and they are named in separate terms of the expression.

```
SEX, RACE BY BAIL_AMT;
SEX, BAIL_AMT BY RACE;
```

These expressions, both valid, result in tables identical is structure and content and differing only in the placement of labels in the heading.

SELECT Statement

The SELECT statement specifies conditions to be met by each record in the data file to be included in a table. One SELECT statement is allowed per request and it must come immediately after the USE statement. The SELECT statement applies to all

tables requested in a TPL request. The SELECT statement looks like this:

```
SELECT IF name relation name ;
```

or

```
SELECT IF name relation 'value' ;
```

Replace name with the name of a control or observation variable. If two names appear, then both must be control variables or observation variables.

The value can be numeric or a literal value enclosed in single quotes. If the variable named is an observation variable then the value must by numeric and can be signed, e.g., 10, +42, or -8. Control variable values are handled in a different way. Only unsigned numbers can be used without surrounding single quotes. All other values including negative numbers are considered literals and must be enclosed in single quotes.

The relation can be any one of the following logical operators:

<	less than
>	greater than
=	equal to
¬<	not less than
¬>	not greater than
¬=	not equal to
<=	less than or equal to
>=	greater than or equal to

For example:

```
SELECT IF BAIL_AMT > 0;

SELECT IF RACE ¬= '-1';
```

In the first example, the SELECT statement would cause selection of all records whose BAIL_AMT is greater than zero. The second example shows use of a literal value. All records except those containing a value of -1 for RACE will be selected.

The selection criteria can be extended to more than one variable by using the logical operators AND and OR. Consider the following examples:

```
SELECT IF BAIL_AMT > 0 AND RACE  = '-1';

SELECT IF BAIL_AMT > 0 OR RACE  = '-1';
```

In the first case, both conditions, BAIL_AMT greater than zero and RACE not equal to '-1', must be true before a case will

be selected. In the second example, a case will be selected if either condition is true.

More than one of the logical operators can be used if needed.

COMPUTE Statement

The COMPUTE statement provides a way of creating a new observation variable from other observations variables and constants. A value is computed for each record in the data file. The format of the COMPUTE statement is:

```
COMPUTE newname : algebraic-expression ;
```

Replace newname with a name for the newly created observation variable. The name may consist of up to 30 letters, numbers, and the underscore symbol. The first character must be a letter and the last cannot be an underscore. Examples of names are AGE_AT_ARREST and X10.

An algebraic-expression is formed in the usual way: observation variables and constants separated by arithmetic operators. The operators are:

+	addition	*	multiplication
-	subtraction	/	division

The arithmetic operators follow the usual precedence rules; i.e., multiplication and division are done before addition and subtraction. The order of calculation can be altered by enclosing parts of the expression in parentheses. For example:

```
COMPUTE TOTAL_AMT: (BOND_AMT + BAIL_AMT) / 1000;
```

The name of the new observation variable is TOTAL_AMT. Its value, computed for each record in the data file is the sum of BAIL_AMT and BOND_AMT, divided by 1,000. Notice that if the parentheses were omitted, the division would be done before the addition and the resulting values would not be the same.

POST COMPUTE Statement

The syntax of the POST COMPUTE statement is the same as the COMPUTE statement.

```
POST COMPUTE newname : algebraic-expression ;
```

The difference between the COMPUTE and the POST COMPUTE statement is the order in which aggregation and calculation are

done. The POST COMPUTE statement sums observation variables specified in the POST COMPUTE statement over the entire data file for each table cell, then calculates the value of the new variable for each cell of the table. Consider:

```
COMPUTE X: BAIL_AMT / DEFENDANT;
POST COMPUTE AVG_BAIL: BAIL_AMT / DEFENDANT;
TABLE TABLE7: DRUG_USER, BAIL_AMT THEN X
     THEN DEFENDANT THEN AVG_BAIL;
```

produces the table:

TABLE7

	BAIL AMOUNT	X	DEFENDANT	AVG BAIL
DRUG USER				
Yes.....	11,000	11,000	5	2,200
No.....	9,000	9,000	7	1,286
Missing Data...	23,000	23,000	3	7,667

For the new variable, AVG_BAIL, TPL sums BAIL_AMT for each DRUG_USER value in the data file, and divides the sums by the number of defendants in each DRUG_USER group. The resulting values appear in the table cells. Notice that the values for AVG_BAIL can be derived from the values shown for BAIL_AMT and DEFENDANT. Also, notice that the values for X are not the same as the values for AVG_BAIL. X is computed for every data record by dividing BAIL_AMT by DEFENDANT which has an assumed value of one. The results are then summed and the final values are the same as the values shown for BAIL_AMT.

DEFINE Statement

The DEFINE statement allows a new control variable to be defined which groups, deletes, and/or reorders values of an old control or observation variable. The old variable may either be defined in the codebook or be a newly computed variable. It may not be a variable created by another DEFINE or POST COMPUTE statement. The DEFINE statement is made up of several lines:

```
DEFINE newname : oldname ;
'condition-name-1' : relation entry-1 ;
'condition-name-2' : relation entry-2 ;
                   .
                   .
                   .
```

The name of the new control variable replaces newname.

Newname is formed as described above under the COMPUTE statement. Oldname is the name of the observation or control variable whose values will determine the values of the new control variable.

Condition-name is the label for the category to be used in tables. It is enclosed in single quotes. The new categories are automatically assigned values of one through n.

Relation is one of the logical operators listed under the SELECT statement and entry is a value or range of values of the old variable. A range of values is specified by two numbers separated by a colon. No relation should be used with a range. If entry is a single value, then a relation should be used. For example:

```
DEFINE BAIL_GROUP: BAIL_AMT;
'0-$1,000' : 0:1000;
'$1,001-10,000' : 1001:10000;
'More than $10,000' : >10000;
```

In this example, a new control variable, BAIL_GROUP, is defined based on the values of the old observation variable, BAIL_AMT. The new variable has three categories with values 1, 2, and 3. BAIL_GROUP has the value 1 if BAIL_AMT is in the range of zero through 1,000; the value 2 if BAIL_AMT is in the range 1,001 through 10,000; and the value 3 if BAIL_AMT is greater than 10,000.

APPENDIX: SAMPLE CODEBOOK AND DATA FILE

RECORD NUMBER	LEVEL NUMBER	KEY VALUE	KEY LOCATION
0.	0	' '	29

DEFENDANT FIELDS: 1

FIELD NAME (SOURCE LINE) / LABEL (IF ANY)	FIELD TYPE	LEVEL NUMBER	RECORD NUMBER	FIELD LOCATION
ARREST_MO (100) 'MONTH OF ARREST'	CONTROL (13)	0	0	30
ARREST_MO_OBS (106)	OBSERVATION	0	0	30
ARREST_YR (88) 'YEAR OF ARREST'	CONTROL (4)	0	0	32
ARREST_YR_OBS (95)	OBSERVATION	0	0	32
BAIL_AMT (80) 'BAIL AMOUNT'	OBSERVATION	0	0	17
BIRTH_MO (22) 'BIRTH MONTH'	CONTROL (13)	0	0	8
BIRTH_MO_OBS (28)	OBSERVATION	0	0	8
BIRTH_YR (33) 'YEAR OF BIRTH'	CONTROL (10)	0	0	10
BIRTH_YR_OBS (41)	OBSERVATION	0	0	10
BLANK_KEY (110)	CONTROL (1)	0	0	34
BOND_AMT (84) 'BOND AMOUNT'	OBSERVATION	0	0	23
DEFENDANT (6)	RECORD NAME	0	0	0
DRUG_USED (60) 'G-DEP DRUG USED'	CONTROL (16)	0	0	15
DRUG_USER (53) 'DRUG USER'	CONTROL (3)	0	0	13
RACE (45) 'RACE OF DEFENDANT'	CONTROL (4)	0	0	6
SEX (16) 'SEX OF DEFENDANT'	CONTROL (2)	0	0	4

DEFENDANT VALUES: 1

FIELD NAME (SOURCE LINE) LABEL (IF ANY) VALUE	CONDITION LABEL OR NAME	NUMBER OF CONDITIONS
ACHAR (110)		1
' '	ACHAR'	
ARREST_MO (100) 'MONTH OF ARREST'		13
'-1'	'Missing Data'	
'01'	'January'	
'02'	'February'	
'03'	'March'	
'04'	'April'	
'05'	'May'	
'06'	'June'	
'07'	'July'	
'08'	'August'	
'09'	'September'	
'10'	'October'	
'11'	'November'	
'12'	'December'	
ARREST_YR (88) 'YEAR OF ARREST'		4
'-1'	'@/@Missing Data'	
'76'	'1976'	
'77'	'1977'	
'78'	'1978'	
BIRTH_MO (22) 'BIRTH MONTH'		13
'-1'	'Missing Data'	
'01'	'January'	
'02'	'February'	
'03'	'March'	
'04'	'April'	
'05'	'May'	
'06'	'June'	
'07'	'July'	
'08'	'August'	
'09'	'September'	
'10'	'October'	
'11'	'November'	
'12'	'December'	

DEFENDANT VALUES: 2

FIELD NAME (SOURCE LINE) — NUMBER OF CONDITIONS
LABEL (IF ANY)
VALUE — CONDITION LABEL OR NAME

BIRTH_YR (33) 10
'YEAR OF BIRTH'

'-1'	'Missing Data'
'54'	'1954'
'55'	'1955'
'56'	'1956'
'57'	'1957'
'58'	'1958'
'59'	'1959'
'60'	'1960'
'61'	'1961'
'62'	'1962'

DRUG_USED (60) 16
'G-DEP DRUG USED'

'01'	'Heroin'
'02'	'Morphine Base'
'03'	'Opium'
'06'	'Methadone'
'09'	'Cocaine'
'10'	'Methamphetamines'
'11'	'Stimulants'
'12'	'Depressants'
'13'	'LSD'
'14'	'Other Hallucinogens'
'15'	'Marijuana'
'16'	'Hashish'
'17'	'Barbiturates'
'18'	'PCP'
'90'	'Other Drugs'
'99'	'Missing Data'

DRUG_USER (53) 3
'DRUG USER'

'1'	'Yes'
'2'	'No'
'9'	'Missing Data'

RACE (45) 4
'RACE OF DEFENDANT'

'1'	'White'
'2'	'Black'
'3'	'Oriental'
'4'	'Other'

DEFENDANT VALUES: 3

FIELD NAME (SOURCE LINE) NUMBER OF CONDITIONS
LABEL (IF ANY)
VALUE CONDITION LABEL OR NAME

SEX (16) 2
'SEX OF DEFENDANT'

'1' 'Male'
'2' 'Female'

TEST DATA

SEX	RACE	BIRTH MO	BIRTH YR	DRUG USER	DRUG USED	BAIL AMT	BOND AMT	ARREST MO	ARREST YR
1	1	258	2	99	1000	3000	12	76
1	1	462	1	15	4000	1000	-1	-1
2	1	355	1	16	0	1000	4	77
1	2	660	2	99	0	0	6	77
1	4	661	9	99	10000	5000	10	76
1	3	554	9	99	5000	5000	2	78
1	4	1259	2	99	5000	6000	3	78
2	2	160	2	99	0	2000	6	76
2	1	357	2	99	0	0	11	78
1	1	1062	1	15	7000	5000	8	77
1	1	861	9	99	8000	0	8	77
1	3	655	2	99	1000	2000	4	78
1	1	556	1	09	0	1000	5	78
1	2	759	1	15	0	4000	7	76
1	2	460	2	99	2000	3000	6	77

END