INTRODUCTION

Analysis of survey data has, now a days, become a very important management function since many surveys are new being conducted by various organisations. These days, surveys are conducted regularly by many organisations to acquire up-to-date knowledge in different fields. Normally, the volume of data collected through the surveys is stupendous and consequently one has to make extensive use of computers to process the data collected.

The job of computatèrised survey data processing involves two basic steps. The first step ensures that the data entered are valid. This data validation involves locating both data entry errors and errors due to mistakes on the part of the investigator.

The second step is that of summarising information from the raw data through construction of various tables. These two basic phases of survey data processing are called scrutiny and tabulation respectively.

Traditionally, the job of survey data processing ~~using~~ used to comprise of writing tailor-made programs for both scrutiny and tabulation. However, this was a time consuming affair since this approach called for writing similar programs several times. This time could be drastically cut down if a very high level language, specially suited for

scrutiny and tabulation, is designed. The objective of this dissertation was to design one such language and to implement it on a micro computer.

As already mentioned, the subject of analysis of survey data is assuming more imporaance day by day. Consequently, efforts were made to develope tailor made packages for scrutiny and tabulation of survey data. One very important package for cross-tabulations, developed by the 'International Statistical Program Center (ISPC)' of the U.S. Bureau of the Census, is CENTS. This package was the result of a substantial amount of research. ISPC developed another package called COCENTS, also used for tabulation systems.

Later, these packages were unified and a new, more flexible and more portable package called CENTS-4 was developed. All the aforementioned packages, however, were implemented on main frames possibly because they were developed before the PC boom.

Thus, though packages were ~~available~~ available for scrutiny and tabulation of survey data, no well-known micro computer based packages were available for these jobs.

This report contains a description of the language developed and its implementation details. The report contains a total of four chapters. The contents of these chapters are as follows -

(i) The first chapter contains general description of the problem, the way the input data are described, definitions of variables, expressions and constants and also contains a short description of the parser generator used to generate parsing-tables.

(ii) The second chapter contains a broad description of the scrutiny language. In this chapter the major constructs of the scrutiny language has been described. Also, the different semantic checks implemented has been discussed.

(iii) The third chapter contains a broad description of the tabulation language. Here also the structure of the language has been discussed at length. Some implementation details also has been provided in this chapter. This was not given in the scrunity section since the steps of implementation are essentially the same.

(iv) The fourth chapter contains a hypothetical data description. A few scrutiny and tabulation programs have been written with respect to this data description, to illustrate the usage of the different language constructs.

Apart from these four chapters, a language manual was also prepared. The language manual contains detailed

description of the constructs alongwith illustrations. However, this is supplied as a suppliment to the report.

It would be worthwhile to mention here that the different modules of the entire software developed are -

(i) The data description module (common to both scrutiny and tabulation program).

(ii) Parser generator module (common to both scrutity and tabulation program).

(iii) Code generator module (Separate programs were developed for scrutiny and tabulation program. This module includes the lexical analyzer and the parser).

(iv) Conversion module to convert the generated intermediate code to COBOL code (Seperate programs were developed for tabulation and scrutiny languages).

All source programs and sample outputs, formats of intermediate code and grammars have been attached as annexures.

CHAPTER - I

GENERAL DESCRIPTION

A. BANERJEE

S. SUBRAMANIAM

BACKGROUND AND OBJECTIVE -

The National Sample Survey Organisation (NSSO) conducts surveys regularly on a veriety of socio-economic aspects. A division of this organisation is completely attached to the processing of the collected data which natuarally involves extensive ~~in use~~ of computers. The aforementioned problem of being forced to write similar programs time and over again was also being faced by NSSO officials and the package was being developed basically in this context.

The primary objective of the package was to design a very high level language to fecilitate scrutiny and tabulation of survey data and to implement it on a micro-computer.

Though the developed package has a special reference to NSSO data, it is not limited to that data only. In fact this package can be used for scrutiny and tabulation in general.

The package has been implemented on the SN - PC's available at ISI and can be directly used by other PC - compatible micro computers. However, since the final output of the software is a COBOL-program, a COBOL compiler is needed to run it successfully.

APPROACH -

The job of designing and implementing the proposed languages was taken up in phases. To start with, the basic constructs of the languages were designed. Also the organisation of the input data was considered and conventions of communicating the data structures were developed. While the basic constructs of the proposed languages have been described in detail in the respective chapters, the organisation of the input data is described in the next section.

The next step was to develope a parser genrator for constructing the parsing tables. A SLR(1) parser generator was designed. The details of the parser generator is given in a separate chapter. Finally parsers were written to generate intermediate code from the input program.

Although generating code in machine language would be the best way from the point of view of execution time of the resulting module, it requires knowledge about the operating system convention for loading and other details. Then the second best is generating in assembly, but no assembler is available in out PCs. This forced us to go for COBOL language since it is easily portable also.

DATA DESCRIPTION -

The input data description forms the most vital part of the entire software developed. Through the data description module, the user specifies the input data file precisely and this description is used elsewhere in the programs for compiling and executing the input programs

The input data are usually gathered in terms of large logical records called schedules. The schedules contain a large number of physical records of equal length. These physical records are again grouped into several groups represented by different level numbers. Physical records having the same level number have the same format and usually these physical records represent items of similar nature.

Each physical record again contain several fields. These fields are referred to as 'items' of the record. As already mentioned, several physical records with the same level number are likely to be present. Thus an item in the schedule can be uniquely represented as a triplet (i, j, k) where

$i \longrightarrow$ level number

$j \longrightarrow$ item (field number)

$k \longrightarrow$ occurrence

Each physical record of the input data file contain an identification field. This field indicates when a particular logical record has been completely described. The location of the identification field is specified by the user at the data description stage.

The physical records are sorted by the identification fields. The level numbers occur at a prefixed location of the physical records. The location of the level numbers within a physical record is, therefore, known to the user and it is communicated to the execution module. Since the formats for every level are known, at run time the level number is first read and the format is found and the data are then read accordingly.

It has been already seen that a variable is uniquely described as a triplet (i, j, k). The third entry of the triplet indicates the physical record number within the level where the variable of interest is to be found. However, this third entry is rarely known to the user. Thus the user normally specifies the level number i and the field number j. The physical record number (also often referred to as the row number) is specified indirectly through certain conditions. Depending on the value of the specified level number, physical records are chosen. Then these records are scanned and when a physical record is found to satisfy the specified condition, the j-th field from that physical record is selected.

The different information provided by the user in the data description module are -

(i) The number of levels in the schedule

(ii) The location of identification fields -- It has been assumed that the location of identification fields also would be the same for different levels. Though it is possible to process the data even without this constraint, it has been added since it is an usual practice and therefore, does not impose any serious restriction.

(iii) The location of level number

(iv) The field descriptions of the items in different levels -

This will be described in the following manner -

At first, the actual level number and the number of fields associated with this level could be described. Once the number of fields associated with a level is described, each of the fields are described in a greater detail. The starting column and the number of bytes to be associated with each field is described. Also the data type of the fields are described.

(v) The number of records placed in a physical record. The necessity of this arises due to the fact that the length of physical records may be large (eg. 127 characters in case of NSSO - data), but there may exist levels requiring only 60 characters. In such cases, data for two records may be combined and stored in one physical record.

(vi) The item code(s) associated with a level. -- Occassionally a level may have more than one item code associated with itself. These should be described alongwith the level description.

(vii) The maximum number of occurrences of a level should be described. This will be used at runtime to store the physical records.

OTHER DETAILS

It has been already described that the variables of the language are written in the (i, j, k) form and usually the third co-ordinate would be absent. The proposed language can also accept numeric and non-numeric constants.

The expressions of the language are formed by combining variables and constants. As usual, an expression may be either a single variable or a single constant or any combination of variables and constants connected through usual arithmetic operators.

Expressions may be combined through relational operators to form logical expressions. Logical expressions can again be connected through connectives like AND or OR and may be negated through NOT.

PARSER GENERATOR -

The parsar generator, though not directly used in the compilation or conversion to COBOL process, forms a core module of the entire study. The input to the parsar generator is the grammar itself and the list of terminals and non-terminals used in the grammar. The program and its input-output formats are given in annexures.

The logic of the program can be described briefly as follows. The program generates a below-matrix with respect to the grammatical occurrences. The columns of this matrix represent different grammatical occurrences. The columns corresponding to thesame grammar-symbol are pooled together to form a NFA. This NFA is then converted to a DFA. The rows of the DFA are sets and these are the different stack-symbol sets. Considering this DFA as a matrix, a submatrix having column headings as non terminals is taken out to obtain the 'GO TO' Table. From the remaining portion the parsing table is obtained through usual method.

CHAPTER - 2

DESCRIPTION OF SCRUTINY LANGUAGE

S. SUBRAMANIAM

OBJECTIVE

The primary objective of the scrutiny program is to provide an 'easy to understand' language through which the user can write a variety of scrutiny tests. Scrutiny tests are of utmost importance since they are used to validate the input data. Thus the quality of the scrunity determines the quality of the input data.

Usually, the designer of the questionnire incorporates a variety of cross-checks in the questionnire so that, at a later stage, consistencies can be checked and even lapses on the part of the investigator could be determined. Also, errors may creep in at the data entry stage, which the scrutiny programs should find and issue appropriate error messages.

GENERAL DESCRIPTION

In this section a broad description of the scrutiny language is presented. The most important constructs are -

i) CHECK STATEMENT - The format for this statement is -

CHECK <Logical expression>

- where Logical expression can be one ofthe following

a) Direct comparison of two expressions.

b) Checking type of a variable such as numeric, alphabete.

c) Checking whether variable of a particular level follow a specific sequence or not. (Serial number checking).

d) Tallying total of a variable level-wise with the supplied total.

e) Expressions formed by the above using connectives OR and AND.

ii) <u>SET MATCHING</u> - The format for this statement is -

CHECK INDEX FOR Var-name 1 [WHERE Logical-exp]
, Var-name 2

This check is needed when a set matching is required. A specific situation could be a case where one level contains the details of all family membersand another level contains only adult males of the family and the objective of the test is to find whether all adult members in the second level has occurred in the first level or not.

It is worthwhile to mention that the WHERE condition is optional and when it is skipped one gots complete set matching.

iii) INTER LEVEL CHECKING -

The format of this statement is -

FOR VAR [WHERE ⟨Logical expression⟩]
[, VAR [WHERE ⟨Logical expression⟩]] .....
CHECK ⟨Logical expression⟩

An example of a situation where this check is performed is the case where the expenditure on an item and the amount consumed are given in two different levels and the objective of the test is to divide the expenditure by amount to get a price per unit which is compared with some reasonable limits.

(iv) IF -THEN -ELSE -

The format of this statement is -

IF ⟨Logical expression⟩ THEN ⟨Statement⟩
[, ⟨statement⟩] .....
ELSE ⟨Statement⟩ [, ⟨statement⟩] ....
END IF

The objective of this test is to conduct certain checks subject to certain conditions. In case the condition followed by the keyword 'IF' is true, the checks following the keyword 'THEN' are executed.

In case the 'else' option is used, the condition following 'IF' would be tested and if it fails, the 'else' part would be executed.

A 'Report' option is also available in the IF - THEN - ELSE construct. The idea behind using this option is to report the success/failure of the main condition depending on whether the option has been used in the THEN part or ELSE part. The explicit form when this option is used is -

IF ⟨Logical expression⟩ THEN REPORT
[⟨Statement⟩]
ELSE ⟨Statement⟩ [⟨Statement⟩]
ENDIF

OR

IF ⟨Logical expression⟩ THEN ⟨Statement⟩
[⟨Statement⟩] ...
ELSE REPORT
ENDIF

v) <u>ASSIGNMENT STATEMENTS</u> -

The format of this statement is -

identifier = expression (arithmetic)

Where identifier is a string of alpha-numeric characters starting with a letter. Embedded dols are also permitted.

The assignement statement is used to store frequently used values and constants. This statement, when properly used is likely to saveexcution time by a large amount. The idenfifier can be used later in other expressions.

TECHNICAL DETAILS

The scrutiny program executes a number of semantic checks to ensure that the input file going for tabulation is totally error-free. The different semantic checks included in the scrutiny language are -

i) Check to ensure that the number of levels and items are within sepecified range.

ii) When a level has variable occurrence, all its variables should be used only in the following cases -

a) FOR statement

b) CHECK INDEX statement

c) Statement forchecking continuity of serial number

d) CHECK TOTALS Statement

e) As argument of built-in functions (SUM, COUNT, AVG, MAX, MIN)

iii) The WHERE condition should involve at least one occurrence of a variable from the corresponding level which preceeds the WHERE condition.

iv) The CHECK condition in the FOR statement should involve at least one occurrence of a variable from any of the levels corresponding to loop variables.

v) The levels corresponding to loop variables must have variable occurrence.

vi) The item code fields should have been defined in the data description module for the levels involved in the set matching.

vii) Any identifier must be assigned before it is used in an arithmetic statement.

CHAPTER - 3

DESCRIPTION OF TABULATION LANGUAGE

A. BANERJEE

OBJECTIVE

The primary objective of the tabulation program is to provide an 'easy to understand' language through which a user can generate a variety of tables. Table generation is the basic process required to extruct information from the raw data and therefore, both NSSO officials and outsiders, who may like to use the NSS-data, are possible users of this tabulation program. Since a substantial portion of these users are not professional programmers, efforts were made to make the language as simple as possible.

GENERAL DESCRIPTION

In this section, a broad description of the tabulation language is presented.

The tabulation language has two basic commands, namely, TABULATE and ENUMERATE. The purpose of these two commands are as follows -

a) TABULATE - This command forces a 'variable' to be added to a specific cell of the table being constructed. The cell to which the 'variable' is added is determined through other considerations specified by the user as a part of his tabulation program. The 'variable' represents some value to be computed from the input data. For example, the

user may like to tabulate the total rice consumption, statewise. In this case, the variable represents the total rice consumption and the different states represent the different cells of the table.

The user may choose to carry out a 'weighted tabulation' too. In this case, the computed variable is multiplied by a suitable weight and this new variable is added to the appropriate cell of the table.

(Syntax of TABULATE is - TABULATE [WEIGHTED] <var> BY .....)

b) ENUMERATE - This command is basically same as the 'tabulate' command except that in this case 1 is added to the appropriate cell instead of adding a 'variable'. In case of weighted enumeration, the weight is added instead of the number 1.

(Syntax of ENUMERATE is similar, but does not have the var. The list followed by 'BY' is the list of classificatory variables.)

CELL - FIXATION

In this section, the method of finding the particular cell of the table, where the addition is to be made, is described.

The cells are implicitly chosen by the user by specifying the list using which the selection is to be made. The list

is written after the key-word 'BY' which conveys to the compiler that the list of variables following 'BY' are the ones, to be used for selection of a specific cell. The variables here has usual (i,j) representation with or without a condition.

To find the specific location of the cell, each one of these variables are converted to their current value and consequently the address is computed.

As already indicated, the total area required to generate the tables is essentially fixed by the range of the variables following the keyword 'BY'. An example would illustrate the point amply -

Let us consider the tabulation statement -

TABULATE AVG [2,4] BY [3,5], [6,7]

The meaning conveyed by this tabulation program is that, for every schedule the average of the 4th item of the 2nd level (ie. level 2) would be computed. For the same schedule, the value of [3,5,1] and [6,7,1] would also be found. The values of [3,5,1] and [6,7,1] would determine the cell to which the value of the average of [2,4] is to be added. Depending on the ranges of these two variables, the table size would vary. It is, therefore, imperative that the user specifies the ranges of these variables.

The ranges of these classificatory variables are conveyed to the compiler through the DIMENSION statement. The DIMENSION statement has two basic formats

i) Var-name followed by range specifications where ranges are of the form - const - const.

or const

Example 1

DIMENSION [3,5] (0 - 10, 11 - 20, 21 - 30)

Example 2

DIMENSION [3,5] (1, 2, 3, 4)

In the first case, during tabulation, the value of the classificatory variable would be evaluated and then a test would be made to find the range to which it belongs. Correspondingly, a value would be assigned to compute the cell location. Thus, if in the example-1 the classificatory variable is found to have the value 15, the value 2 would be used to find the cell where the final addition is to be made.

In the 2nd case also the problem of finding cell address is resolved in a similar manner.

Since The DIMENSION statement specifies the range of variation for each variable the table size for each tabulation is computed from the details given in the DIMENSION statement. In case of ENUMERATION also the structure and usage of the DIMENSION statement is exactly same.

ASSOCIATED CONDITIONS

Often certain conditions are associated alongwith the TABULATION/ENUMERATION statements. The objective of these conditions is to ensure that the TABULATION/ENUMERATION is carried out for a schedule only if the schedule satisfies certain properties. As an example, one may like to tabulate the total income of households, statewise and familysize-wise only if the total income exceeds Rs.20000 per annum (ie the interest may be to tabulate only high or middle income group people). In this situation, a household having an income of less than Rs.20000 per annum should be skipped. The conditions associated with the TABULATION/ENUMERATION progam serves this purpose.

The associated conditions are specified as a list of conditions starting with the keyword 'FOR'.

Thus the schedules which do not satisfy the conditions listed after the keyword, 'FOR' are simply skipped.

The conditions have usual logical expression format and allows the use of simple arithmetic operatore like +, -, * and / . The usual relational operators like =, < , >, =, > =, < > are permitted. Also connectives like AND, OR and negation (NOT) can be used to form the conditions. Unary + and - are also allowed.

Conditional variables and conditional functions -

In executing the TABULATION/EXUMERATION program, one is frequently required to find a variable satisfying some condition or evaluate a function satisfying some constraints. It has been already mentioned that some of the levels appear more than once. Whenever a level appears more than once, it usually describes similar things on different items. For example, a particular level of a schedule may be devoted to details of different kinds of food items produced by a household. Thus the level may have entries like the amount produced, the amount sold, the profit made and so on. Naturally, this level would occur more than once, once for every item produced by a household. Thus the level would allow an 'items code' to be satisfied, through which the commodity being described by a particular physical record would be known. Now, it may be a requirement to find the amount of milk produced by a household. The only way of doing this is to check the item code of all physical records of the concerned level and pick the one in which the item code matches with the item code of milk. This type of a variable selection is termed as a 'conditional variable'. Conditional functions' also have a similar meaning.

The tabulation language permits use of conditional functions and conditional variable in all its clauses except the DIMENSION clause - where it does not have a meaning.

THE FINAL STRUCTURE

The tabulation language may, thus be described as -

i) Every TABULATION/ENUMERATION statement must start with a DIMENSION statement. The dimension of each classificatory variable must be declared. However, the keyword DIMENSION should be mentioned only once.

Thus, while declaring DIMENSION of two variables, one would specify it as -

DIMENSION [5,7] (10 - 20, 21 - 30)
[3,8] (0 - 10, 11 - 20, 21 - 40)

ii) The statement following DIMENSION must be a TABULATE or ENUMERATE statement. This statement must start with the corresponding key-word. This key-word is ~~optimally~~ optionally followed by the key-word WEIGHTED.

This is followed by either a variable or a function of a variable or a conditional variable or a conditional function, in case of TABULATE statement. In case of ENUMERATE statement, the key-word 'BY' is expected.

Thus, the different possible structures are -

a) TABULATE var-name BY .....

b) TABULATE WEIGHTED var-name BY .....

c) TABULATE fn. name var-name BY

d) TABULATE WEIGHTED fn. name var-name BY

e) TABULATE fn.name var-name WHERE (condition) BY

f) TABULATE WEIGHTED fn.name var-name WHERE (condition) BY

g) TABULATE var-name WHERE (condition) BY

h) TABULATE WEIGHTED var-name WHERE (condition) BY

i) ENUMERATE BY

j) ENUMERATE WEIGHTED BY

iii) The keyword BY must be followed by a list of variables. These variables may be either simple variables or may be conditional variables. A simple variable would be described through the usual (i,j) representation. This would be used when the classificatory variables belong to a level having a single occurrence. This may be the case when the classificatory variables are state name, stratum name etc. Normally, such variables are expected to be described in level no.1 having a single occurrence.

The conditional variables also can be used in more general cases, to pick up a specific variable from a level having more than one occurrence. The conditional variable would be specified through the following structure -

Var-name WHERE (Condition)

When more than one classificatory variable is used, these variables must be seperated by commas.

A point worth mentioning at this juncture is that, the classificatory variables should be specified in the same order in which their dimension had been specified. This restriction had been imposed to fecilitate implementation of the language.

iv) The list of classificatory variable is followed by an optional condition list, which must start with the keyword 'FOR'. In case the condition-list is absent, the user must terminate the list of classificatory variables by either a ; (semicolon) or by a . (dot).

The list should be terminated by a semicolon when the current tabulation statement is not the last statement. In case it is the last statement, the list should be terminated by dot.

The keyword 'FOR' would be followed by a condition-list. The conditions in the condition list, as already mentioned, has the structure of usual logical expression of high level languages like BASIC.

The list of conditions should be terminated by a semicolon or a dot, as the case may be.

Technical details -

Implementation of the language was a four-step process. Once the basic constructs of the language was though of, a grammar was written to formalize the language. As a second step, a parsing table was obtained by subjecting this grammar to a SLR(I) parser generator. At the third step, a code generator was developed which included two main modules namely the lexical analyzer to supply the tokens and a syntax-analyzer and code generator to generate an intermediate code. The grammar and the format of the intermediate code are given in annexures 1 and 2 respectively. The described grammar has many facilities but all of them were not implemented in the language designed. However, the parsing table was generated for the ~~either~~ entire grammar and hence these extra features which are not part of the language currently can be easily incorporated by writing the corresponding reduction routines. These features were not incorporated in the language implemented since the chances of these features being used are rather remote. Since the available time was short, it was decided to implement the most useful portion.

The fourth step of the process was to write the execution module. At first it was decided to write the execution module based on the generated intermediate code. The idea in that case was to have routines for the different op-codes, which will be called and there-by the execution would be performed.

The main drawback in this method is that, the execution time is very high. Since one of the primary users of this language would be NSSO where the volume of the data is extremely large, the efficiency is of utmost importance. Thus it was decided to generate a COBOL program from the intermediate code. The entire job of designeing and implementing the language for survey data tabulation was thus completed in the four steps mentioned above.

A SHORT DESCRIPTION OF PROGRAMS -

Details of the parser-generator program has already been provided in the general section. In this section a brief description of the code generation and generation of COBOL - program from the intermediate code is presented.

As already mentioned, the code generator has two basic modules - the lexical analyzer and the code generator. The code generator is the main module which calls the lexical analyzer as and when the next token is needed. The exact implementation was to read one line of the source code from the input program file as a string and send this string to the lexical analyzer. The lexical analyzer then extracts tokens from this line and keeps them in a buffer and transfers control

back to the parser. The parser now takes the token one by one from the buffer, consults the parsing table and takes appropriate action. Whenever a reduction is called for, code is generated if that reduction calls for a code generation. When the token buffer is empty the lexical analyzer is called again to read the next line of input.

The parser program is attached as an annexure for further reference.

The program which generates a COBOL program from the code generated by the parser, works as follows.

This program uses as its input a few files created by the parser. These files are the code file holding the generated code, the variable file holding the description of all the variables encountered in the program. The organisation of the variable file is as follows -

The entries would be sets of size three or four. The first entry of the set would equal the size of the set minus one. The next entries would be the level entry, the item number entry and so on. Thus the variable [2, 7] would be represented in the variable file as - 2, 2, 7.

Another file used by the generator and created by the parser is the range-file. This file stores the details of DIMENSIONS specified by the user. This file stores the number of locations required for each variable and the upper

and lower limit specified by the user in each class for each variable. Alongwith the range file, an associated file is maintained which points to the starting entry for each tabulation/enumeration command.

The last file created by the parser and used by the generator is the file of constants where the constants encountered in the user program are stored.

The generator program reads these input files and then converts the pseudo code into COBOL-code on a instruction-by-instruction basis. A few example programs, the code generated and the generated COBOL-program is provided in the annexures.

CHAPTER - 4

EXAMPLE OF SCRUTINY AND TABULATION

LANGUAGE PROGRAMS

A. BANERJEE

S. SUBRAMANIAM

USAGE OF THE LANGUAGES -

In this section a hypothetical input data has been described and a few scrutiny and tabulation programs were written in the proposed languages. The problems are first prosented verbally and then programs were written for the problems.

Suppose the survey data consists of three basic parts say details of age, income and occupation as part 1, details of monthly consumption and expenditure on general food items like rice, wheat, milk and so on as part 2 and details of expenditure incurred on the education of children of the family as the third part. Let us assume that part 3 describes the educational expenditure incurred by each adult male of the family.

Let us further suppose that the data are available state-wise, district-wise, village-wise and there are 5 states, at most 8 districts for each state and at most 12 villages for each district.

Let the actual layout of the input file be as follows -

1. Cols. 1 - State code
2. Cols. 2-3 - District code
3. Cols. 4-5 - Village code
4. Cols. 6-7 - Household number
5. Cols. 8 - Level.

It could be easily noted that these eight columns represent the same items in all physical record. In this particular case, columns 1 to 7 represent the identification fields. It could be also noted that, in this case, a 'schedule' is nothing but complete data for a household. As soon as the house-hold number changes, one understands that one logical record (household) has been covered and the next logical record is being accessed. Thus, change from one schedule to another brings about a change in the identification fields. Suppose, the field description of the other items are -

Level 1 -

6. Cols.9-10 - Serial number
7. Cols.11-12 - Age in completed years
8. Col. 13 - Sex
9. Cols.14-18 - Monthly income (Rs.)
10. Col. 19 - Occupation code

(1 - Self employed
2 - Working in private sector
3 - Working in govt. sector
4 - unemployed)

Level 2 -

6. Cols. 9-10 - item code
7. Cols. 11-15 - monthly consumption of item (Kgs)
8. Cols. 16-19 - amount spent on this item (Rs.)

Level 3 -

6. Cols. 9-10 - serial number (as in level 1)

7. Cols. 11 - education code

(1 - illeterate

2 - upto primary level

3 - upto aecondary level

4 - upto graduate level

5 - above graduate level)

8. Col 12 - marital status

9. Cols. 13-14 - no. of children

10. Cols. 15-16 - no. of school-going/college-going children

11. Cols. 17-19 - monthly expenditure on education

The above data description, though a hypothetical case, is a typical example of structure of survey data. A few scrutiny and tabulation problems are now verbally described with respect to this data description and the solution in terms of the proposed language is also presented.

Examples of scrutiny problems -

1) Check whether the serial numbers appearing as item 6 of level 1 are continuous or not (ie check whether the family members are given proper serial numbers or not)

Soln. CHECK COLUMN [1,6] IS CONTINUOUS ;

2) Check whether all adult males appearing in level 1 also appear in level 3 or not. Assume sex code for male is 1

Soln. - MALE = 1

CHECK INDEX FOR [1] WHERE

[1, 8, *] = MALE

AND [1, 7, *] > = 16,

[3] ;

3) Check whether the number of children in level 3 (item 9) is greater than or equal to the no. of school-going children (item 10 of level 3) for all entries in level 3.

Soln. - FOR [3] CHECK [3, 9, *] > = [3, 10, *]

4) Check whether the total monthly income is greater than the total expenditure on education or not.

Soln. - CHECK SUM ([1, 9]) > SUM ([3, 11])

Examples of tabulation problems -

1) Tabulate the total income (monthly) for those families, whose size is less than or equal to 8, state and district-wise.

Soln. - DIMENSION [1, 1] (1, 2, 3, 4, 5)
[1, 2] (1, 2, 3, 4, 5, 6, 7, 8)
TABULATE SUM ([1, 9]) BY [1, 1] , [1, 2]
FOR COUNT([1]) >= 8.

2) Tabulate milk consumption of all those families, where at least one person is self-employed, total family income-wise choosing the income ranges as 0-1000, 1001-5000, 5001-10000 and 10001 to 99999. Assume that 'item code' of milk = 10.

Soln. - DIMENSION [1, 9] (0-1000, 1001-5000, 5001-10000, 10001-99999)
TABULATE SUM ([2, 7]) WHERE ([2,6] = 10)
BY [1, 9]
FOR ((COUNT([1]) WHERE ([1, 10] = 1) >) = 1).

jg.