Chapter 4

Natural Language Control System (NLCS)

4.0 Introduction

The notion of Natural Language Control System (NLCS) described in this chapter performs the following four functions i.e.:

i) to understand a natural language question/query with the help of keywords.

ii) to provide an interface to convert the user query into intermediate form as understood by the system.

iii) to process the intermediate query, produce an intermediate file with relevant information extracted from numerous databases.

iv) to generate a report, through a report writer that makes extensive use of this intermediate file.

The Natural Language Interface (NLI) which performs above tasks is named as Intelligent Advisor (IA). It has been implemented on NEC S1000/20D system and acts as an interface between energy database (ENEDB) and user's/energy experts. Its implementation will free users from learning syntactic rules and restriction of tedious query language of MDQS (software supplied by vendors). The aim of IA is to allow the user to interact with database, to ask questions about the information in a variety of ways using a natural language (say English), and relieve the user from the botheration of knowing the cumbersome semantic and syntax requirements in constructing a query.

1 described in 72 annexures of appendix V
Additionally, IA allows user to bypass even the grammatical restrictions of the English language. This is significant in the context that not all the users are good at English.

4.1 **Capabilities of Intelligent Advisor**

The design of IA makes the working of the system user friendly. It permits user to ask questions in an interactive or batch mode in a multi-user environment. Its capabilities are summarized below.

4.1.1 **Adequate accessibility**

IA permits users to pose a query in English and is particularly useful for selective information retrieval, analysis, manipulating individual records and updating the database. It supports the following operations:

(a) **Retrieval** : Capable of searching information from numerous databases with specified selection criteria. For example:

"Get all units of Badarpur Station having type Thermal"

(b) **Reports** : Capable of producing reports in tabular form. For example when following query is processed,

"PRINT ALL units of BADARPUR STATION with its capacity and installation date", the result will be in report form as shown in (Query 2 of section 4.5.2.6).

(c) **Computation** : Capable of performing mathematical calculations. For example: the execution of the following query

"What is the Plant Load Factor of Badarpur Station from 9001 to 9002", requires computational formulae relating to plant Load Factor.

124
(d) **UPDATION**: Permits a) deletion of a record, b) addition of a record and c) modification of fields of a database record in interactive or batch mode.

4.1.2 **Habitability**

Permits the user to submit a request in a natural language sentence that comes readily and comfortably to his mind. For example:

"Print all stations whose type is thermal"

"Print a list of all stations in Delhi state"

System has been made habitable in three ways. Firstly, training users through an interactive process wherein the user selects and goes along with the logic already built in. Secondly, permitting users, who are expert in Energy sector to submit their query more directly and obtain the required information. Thirdly, providing the facility to submit a query either in an interactive or on batch mode. Furthermore, IA permits storage of query output in a predefined file which can later be accessed by the user.

4.1.3 **Verifiability**

IA accepts questions asked in English on a wide variety of categories; however, there are often ambiguities. To ensure that whether the interpretation of a request is consistent with what the user intended, IA displays on terminal a modified form of query as understood by it and accordingly, it asks user's permission before proceeding further with the query.
4.1.4 **Resilience**

As already stated earlier, IA is designed in such a way that it takes away the worry from users mind about the grammatical errors (of in English language sentence) in the query submitted. For example, although the words have been jumbled in the following query, still it will be treated as a valid query.

"BADARPUR STATION ALL UNIT LIST"

4.1.5 **Performance**

The IA has been developed on a MAINFRAME system. The time taken to process a query is reasonably fast and it takes, on an average, about 20-30 seconds.

4.1.6 **Adaptability**

IA interface supports both the technical and non-technical users and hence, it is be acceptable by all types of users.

4.1.7 **Synchronization**

NLCS maintains information on all permitted legal (reserved words) words in a system lexicon built for the purpose of providing the requisite infrastructure for use by IA. The reserve word relates to:

(a) Category (e.g., Capacity, Generation)
(b) Query identifier/type,
(c) States (of Indian Republic and additional states classified by Ministry)
(d) Station (Power plants) and its synonyms
(e) Sector
(f) System (Power supply agencies, e.g., Damodar Valley corporation)
These words\textsuperscript{2} should be used by user in his query. In case they are not, then the IA informs the user to take help, using the HELP utility of the NLCS system, which provides the display of contents of system lexicon. The purpose of the above effort is essentially to help the user in matching/synchronizing his thought process with the usage of NLCS as devised.

4.1.8 Avoiding overloading & congestion due to wrongly formulated queries

IA permits user(s) to submit same or different queries from a number of terminals. In this process, naive users are likely to submit ambiguous and vague question that will cluster and overload the system as well as cause congestion on communication network. To obviate this problem, IA breaks all queries into sub-queries and returns these to user for confirmation. User would need to type \texttt{CONTINUE} for processing to begin. Determining the answerability\textsuperscript{3} of a query, prior to actual search of a database may improve the search aspects. Firstly, it will reduce communication network load and congestion by stopping an invalid and ill-formulated query at the remote source. Secondly, computer could be used extensively by normal

\textsuperscript{2} Please refer to appendices I(B) to IV for details.

\textsuperscript{3} The query as understood by the system, is displayed to user before postprocessing starts. Once user is satisfied and has given his concurrence to proceed ahead, it means that the chances of answerability is increased in comparison to normal mode of working ( i.e. when the user’s consent is not taken).
and casual users during this time (i.e. the time that might have been taken in processing illegal query). Thirdly, the reduction of the congestion on communication network will yield better system response time, reduce overhead and bring better performance. Finally, preprocessing of queries and determination of an answerability may also benefit the database user in formulating more meaningful queries.

4.2 Information processing through IA

IA is an intelligent program which provides a sophisticated interface between man and machine for processing queries relating to Energy Sector from the Energy database (ref. figures 4-1 & 4-2).

The user query is first parsed by parser (refer section 4.5.1.2 to 4.5.1.4) which reduces it to a simpler form, next it is passed to IRS which processed it, and finally, the results of the query is made available to user via a post-processor described under post-processing of query in section 4.5.2.

4.3 Assumptions on which IA works

(1) IA accepts any query written in Natural Language (English in this case).

(2) Query can be submitted in free form, wherein the user need not worry about the grammatical aspects of English language in framing his query.

(3) Space Character or a Carriage Return (CR) is treated as separator between two words.

(4) The word in a query sentence can be a Query Identifier/type or a Category word (CW) or a date or a value to be searched. If a query does not contain a valid Identifier or a valid CW, query will be rejected.
1. LOGIN THE SYSTEM
2. INVOCATE INTELLIGENT ADVISOR (IA)
3. QUERY (OBJ) IN ENGLISH
4. CALL GET_TOKEN()
   1. ACCEPT OBJ
   2. PARSE OBJ
   3. SCAN OBJ
5. SCAN FOR CATEGORY USING FIND_CAT word (CM)
6. IS IT VALID Category S
7. WRONG CATEGORY
8. CALL CODESEL() ROUTINES
   1. CODESTN OR CODEST OR CODESC OR CODESEC OR CODESYS OR CODEOUT
   2. to find query type
9. CALL KEYS() OR KEYS() OR KEYS() OR KEYS() OR KEYS() OR KEYS() TO FIND DETAILED QUERY TYPE
10. CALL SYSDTO TO VALIDATE DATE
11. SECOND DATE = CURRENT DATE
12. DATE1-DATE2 = YYYY MM DD
13. DURATION: ONE DAY
14. 1. CONVERT DATE INTO FORMAT YYYY MM DD
15. 2. SUBSTITUTE DATE
16. TAKE CURRENT DATE FROM SYSTEM
17. YES
18. IF SECOND DATE MISSING
19. YES
20. 'SAY' 'TO' 'FROM' 'MONTH' OF
21. COMPARE AS PER LOGIC OF FIG. 4-6
22. NO
23. NO
24. NO
25. YES
26. IF BOTH DATE MISSING
27. YES
28. DISPLAY SUB-QUERIES TO USER
29. INPUT NEW CRITERIA
30. WANT TO CHANGE CRITERIA
31. YES
32. DISPLAY ADVISE
33. YES
34. IS QUERY ON ADVISE
35. NO
36. NO
37. STOP
38. CONT. PROCESSING
39. RESULTS FROM PHASE II
40. A
41. B
42. GO FOR PROCESSING AS SHOWN IN PHASE II

(Fig. 4-1)
PHASE II - POST PROCESSING OF INTERMEDIATE QUERY

1. PRE PROCESSING OF RESULTS FROM PHASE-I
2. CONVERT QUERY PARAMETERS IN MATHEMATICAL FORM
3. ASSEMBLE INFORMATION AS PER QUERY
4. CONVERT INTO ADL FORMAT
5. INVOKE MDOS TO COMPARE ABOVE INFORMATION
6. STORE OUTPUT IN OBJECT FILE

IS INVOKING OK

NO

CHECK PROCEDURE

YES

STORE RESULTS

DISPLAY RESULTS

STORE RESULTS IN FILE

TO PHASE I AT FIG.4-1

(FIG.4-2)
(5) Example's of Query Identifiers/types are 'STATION', 'STATE', 'REGION', 'UNIT', 'COAL', 'OUTAGE', 'SECTOR' etc. The value of query identifier to be searched in query must come before the identifier. i.e. the name of state or station or region must precede the word STATE or STATION or REGION respectively.

Consider following phrase used by user in his query:

(a) BADARPUR STATION or BAD STATION
(b) DELHI STATE or DEL STATE
(c) CENTRAL SECTOR or CEN SECTOR
(d) NORTHERN REGION or N REGION
(e) I.P. STATION or IP Station or INDRA STATION
(f) BADARPUR
(g) STATION OF BADARPUR
(h) STATE IN DELHI
(i) SECTOR Central
(j) INDRA PRASTHA STATION

In above examples, (a) to (e) are valid phrases; (f) is invalid as after BADARPUR, there is no identifier; (g), (h) & (i) are invalid as value of query identifier 'STATE', 'STATION' and 'SECTOR' are appearing after 'STATION', 'STATE' & 'SECTOR' respectively; '(j)' is invalid as 'PRASTHA'(used before 'STATION') is not a valid station. Also, as the space character is treated as separator between words, INDRA PRASTHA station will be taken as 2 words. In all such cases either the words should be written in continuation or only first word or its synonyms may be used (refer Appendix I(B)). Second word (if missing) will be substituted by the system.

For Example: INDRA PRASTHA should be written as - 'INDRAPRASTHA' OR 'INDRA' (first word) or 'IP'(synonym).

(6) The VALID identifiers in the operation of power projects are 'CAPACITY' for 'INSTALLED GENERATING CAPACITY'; GENERATION
FOR 'GROSS ELECTRICITY GENERATION', 'PLF' for 'PLANT LOAD FACTOR', 'PM' for 'PLANNED MAINTENANCE', 'FO' for FORCED OUTAGE, 'RS' for RESERVE SHUTDOWN, 'COAL', RESERVOIR, 'ADVICE' for EXPERT ADVICE; 'LIST' for listing.

(7) Date must be used in the form 'YYMMDD' where YY is Year, MM is Month and DD is Date. If year and month is to be used, it should be written as YYMM. Date in a query can appear after "AS ON" or after "FROM" or after "TO". Example of a valid date is 910519 and valid Year-Month is 9105.

Criteria used for handling dates in a query is described as under:

(i) If starts-date or end-date or both are missing, system assumes

Date = current date (default date)

(ii) If only one date is to be given,"AS ON" clause should be used. For example:

Query : "GIVE CAPACITY OF IP STATION AS ON 890101"

(9) END of LINE of a QUERY/REQUEST is sensed by Carriage Control Character/Carriage Return <CR>.

4.4 Facilities provided by IA to user

IA provides the following facilities to user.

(1) User is free to use words of his choice in any order. System scans query from left to right, top to bottom. If user types an error in input of a value (say an identifier name), system corrects it and displays to user as shown in the following Example:

Query : "GIVE ME CAPACITY OF BADARXXX STATION"
System corrects the value of identifier 'BADARXXX' as BADARPUR and displays following to user:

QUERY CATEGORY - CAPACITY
STATION NAME - BADARPUR
DATE - MISSING

(2) If the value of an identifier is in two words, the user is given freedom to enter the first word. The second word is automatically appended in query and will be displayed to user before processing a query. For example, the system, on coming across the following query:

"DISPLAY GENERATION OF INDRA STATION"

(wherein only the first word of the station (INDIRA) is used), will return following to user before processing the query

QUERY CATEGORY - GENERATION
STATION NAME - INDRA PRASTHA

The word "PRASTHA" has been appended by the system.

(3) If the value of an identifier is in more than two words, user can give value by putting the words together or can also use short abbreviation or synonyms [refer Appendix I(B)]. For example:

PLANT LOAD FACTOR can be written as 'PLANTLOADFACTOR' OR 'PLF'

(4) If the user wishes to give value of identifier in a string of few characters, IA accepts and displays to user all the names which start with that string as illustrated in the following example:

"GIVE ME CAPACITY OF HARD STATION"
System while processing above query, finds two names starting with string HARD. It displays both names along with code and advises user to select one code from the display shown below;

<table>
<thead>
<tr>
<th>CODE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>HARDUAGANJ-A</td>
</tr>
<tr>
<td>2.</td>
<td>HARDUAGANJ-B</td>
</tr>
</tbody>
</table>

(5) If user is interested to know the current status of any category (say generation) and has forgotten to provide date, then the system provides the information using the current date (from system). Also, it assumes duration as one day for calculation purpose.

4.5 Design Of IA: Query Analyzer

IA accepts and scans a query submitted by user with the help of various parsing routines described under Section 4.5.1.4. The logic and algorithm developed are elaborated under each routine. User inputs a query in English. The parser parses the query and converts into sub-parts in program understandable form. This form is transferred to IRS. The IRS converts query into mathematical form, displays query to user in user understandable form, obtains permission from user before sending it for further processing, links user query to query entry point, then it assembles information required to link database records and items as per query and submits to MDQS. The MDQS (a vendor supplied software) compiles output of IRS, links database records as per user query and stores it in an object file ‘OBJ’. Then begins the post processing of query where the parameters earlier passed by IRS are used to fetch data from database. Report writer program is
used if report is required in formatted way. The items retrieved from various databases are linked and the result of the query is displayed to user.

Query handling can be divided into two stages

(a) Preprocessing of query
(b) Post-processing of query

4.5.1 Preprocessing of query

Preprocessing of query is handled by IA in various stages. These stages are query acceptance, query scanning, query understanding and query conversion to a form acceptable to IRS. Primary effort in various stages is to scan the words in sequence. The parsing and scanning involves development of a process based on an algorithm devised for the purpose. The logic used for handling a natural language query are described under sections 4.5.1.1 to 4.5.1.3 (refer fig 4-3 to 4-6).

4.5.1.1 Query Acceptance Phase and Parsing methodology

(1) Query from a user terminal is accepted in Natural Language in either an interactive or a batch mode. Interactive usage mode invokes IA (Intelligent Advisor) and prints following message on the terminal:

'Please type your query......'

User can now enter his query/request in English in a field free format.

4.5.1.1.1 Query parsing Methodology

A sample user query has been used by user in four ways in the illustration given below. However, even though all the four examples are framed differently, they will all have similar results as junk words are ignored by the system.
Example 1 (a):

User Query

GIVE ME THE CAPACITY OF BADARPUR STATION AS ON 920101

Query Category

keyword value of station

date pattern (keyword) value

JUNK WORDS

Example 1(b):

User Query

DISPLAY AS ON 920101 BADARPUR STATION CAPACITY

JUNK WORD DATE PATTERN (KEYWORD)

DATE (KEYWORD VALUE)

STATION (KEYWORD VALUE)

KEYWORD QUERY CATEGORY

Explanation

This query is similar to query shown in Example 1 above.

Example 2(a):

User Query

BADARPUR STATION CAPACITY

STATION KEYWORD QUERY CATEGORY

Explanation

It may noted that this query is similar to the queries in Example 1(a) and 1(b) except for omission of a specific date. Since, the date has not been supplied, the program will fetch (by
default) the current date from the system. This query, in fact, represents a case of minimal query (i.e. a query with minimum number of words required in it).

Example 2(b)

User Query

```
BADA
---
STATION
---
CAPACITY
---
KEYWORD
VALUE

MATCHED BY
FIRST 4 CHARACTERS

QUERY
KEYWORD

MATCHED BY
FULL STRING

QUERY
CATEGORY

MATCHED BY
PRONUNCIATION
```

Explanation

This query is identical in its output to the result of the query of example 2(a) above. The difference in the two pairs shall only be in respects of the dates.

4.5.1.2 Query Scanning & understanding Phase

(1) The text of the query is scanned character by character in array bfr[] by get_token routine until a <CR> is encountered. Scanning bfr proceeds from left to Right. If a blank character or <CR> is encountered, IA transfers the string of characters encountered so far to the first word of array token[]. It continues scanning further and keeps transferring in a sequential manner all the words encountered to array token[] till a <CR> is encountered. The count of number of words stored in array ‘token’ is kept in ‘no_tok’. The logic followed is shown in figure 4-3.

(2) Now array token is scanned by the IA for predefined key words, or patterns that indicate known objects or
FUNCTION GET_TOKEN

This routine accepts a user query/request, reads it in 'BUFFER', separates words encountered in the query, stores them in the array 'TOKEN', and counts the number of words in 'NO_TOK'.

**Algorithm:**

1. **GET_TOKEN()**
2. **PLEASE TYPE YOUR QUERY**
3. **READ USER QUERY/REQUEST IN BUFFER TILL A CARRIAGE RETURN (CR) IS ENCOUNTERED**
4. **I ← 0, NO ← 1, J ← 0, NO_TOK ← 0**
5. **IF BUFFER(I) = “CR”**
   - **NO**
   - **ELSE IF BUFFER(I) = “.”**
     - **NO**
     - **ELSE**
       - **RETURN**
6. **TEMP(J) ← BUFFER(I)**
   - **I ← I+1, J ← J+1**
7. **IF BUFFER(I) ≠ “.” OR CR**
   - **NO**
   - **ELSE**
     - **TOKEN(J) ← TEMP**
     - **J ← 0**
     - **NO_TOK ← NO_TOK + 1**

**Flowchart:**

- **Fig. 4-3**
relationships. In the first scan, the query category is determined by matching words of array 'token' with category word (CW) by function find_cat. QRYID is set to 0 if CW is capacity; set to 1 if CW is generation; set to 2 if CW is PLF; set to 3 if CW is outage; set to 4 if CW is FO (Forced outage); set to 5 if CW is F_M (forced and miscellaneous) outage; set to 6 if CW is miscellaneous outages; set to 7 if CW is OA (Operating Availability); set to 8 if CW is Advice; set to 9 if CW is coal; set to 10 if CW is reservoir and so on. If it is not possible to find CW, user is sent suitable message to amend the query appropriately.

(3) Array token is also scanned for finding query type(s). Functions namely KEYST, KEYSTN, KEYRG, KEYOUT, KEYSYS, and KEYSEC, help in ascertaining whether the words in array token is a state or a station or a region or a system or a sector or an outage. The logic followed by Function KEYST() has been shown in figure 4-4.

(4) The combination of keywords on query type are checked by the function CDSELE. The functions CODEST, CODESTN, CODERG, CODEOUT, CODESYS and CODESEC are used to find a detail query type within STATE, STATION, REGION, OUTAGE, COAL respectively. Functions KEYST, KEYSTN, KEYRG, KEYOUT, KEYSYS, KEYSEC are called from CODEST, CODESTN, CODERG, CODEOUT, CODESYS, CODESEC respectively.

The identifier value as given by the user in query is compared with the value stored in the system and if they match, the processing takes place. The calling routine is returned with values for various parameters as explained
FUNCTION KEYST()

This routine scans array TOKEN for finding keyword state or states and return state-code, state-name and region-code.

(CAT. 4-4)
in steps (a) to (f) described below:

(a) Function KEYST after matching value of a state used by user, with the database value on state, returns region code, state code, state name to function CODEST. If state is not found, an error flag is set and user is suitably informed.

(b) Function KEYSTN after matching value of station name used by the user, with the database value on station name, returns value of region code, state code, station code, station name and its type to function CODESTN.

(c) Function KEYRG after matching value of region, with the database value on region name, returns value of region code and region name to CODERG.

(d) Function KEYSYS after matching value of system, with the database value on system name, returns value of system code, system name, system category to CODESYS.

(e) Function KEYSEC after matching value of sector, with the database value on sector name, returns value sector code and sector name to CODESEC.

(f) Function KEYOUT after matching value of outage, with database value on outage name, returns outage code, outage main classification and outage name. Function CODESTN, CODEST, CODERG, CODESYS, CODESEC, CODEOUT returns a value under flag 'QRYOPT' after identifying the detail query type. The logic followed by function CODEST is described under figure 4-5.
(5) The query is to run for a fixed date or for a fixed duration. For finding duration, array token is again scanned to find the word 'FROM' or 'TO' or 'AS ON' or 'TILL' or 'MONTHOF' or 'MONTH'. SYSDT function helps in finding date and time but if dates are missing, it is fetched from system and converted into a form YEAR-MONTH-DATE (YYMMDD).

(6) As earlier stated, IA expects date in the form YYMMDD. 'START-DATE' is entered after 'from' and 'END-DATE' after 'TO'. For example:

GIVE ME CAPACITY OF BADARPUR STATION FROM 870131.

Above, 'END-DATE' is missing, system assumes 'END-DATE' as 'CURRENT-DATE' and it is fetched by function SYSDT() as shown in figure 4-6.

4.5.1.3 Query Formulation Phase

During query acceptance and scanning, the routines get_token, KEYSTN, KEYST, KEYRG, KEYSYS, KEYSEC, and KEYOUT are used to find query identifier and return various codes to main program. Function CODESTN, CODEST, CODERG, CODESYS, CODESEC, CODEOUT, returns different values of QRYOPT. Function find_cat is used to find Query category. SYSDT routine returns the date used for calculation of duration. IA passes all above parameters to IRS for converting intermediate query into mathematical form, assembling information to link database, post processing of query and printing query result. The function, logic and algorithm of various routines used during parsing of a user query are described under section 4.5.1.4.
FUNCTION CODEST()

FUNCTION CODEST CALLS FUNCTION KEYST TO CHECK WHETHER KEYWORD STATE OR STATES HAS BEEN USED IN USER QUERY. IT RETURNS QRYOPT VALUE FOR THE QUERY TYPE (REF. QRYOPT TABLE)

1. CODEST()
2. KEYST()
3. I ← 0
4. TEMP ← TOKEN(1)
5. IS I EQ 0
6. YES
7. IS TEMP IN REGIONS OR REGIONS
8. NO
9. I ← I + 1
10. TEMP ← TOKEN(I-1)
11. IS TEMP = TOKEN(I-1)
12. YES
13. GET REGION NAME
14. PICK UP FIRST REG-NAME
15. IS TEMP EQ REG-NAME
16. YES
17. REGION KEYWORD NOT IN INPUT STRING
18. NO
19. REGION NAME DOES NOT EXIST IN REGION
20. RETURN
21. WRONG OPTION
22. GET NEXT RECORD OF REG-NAME
23. END OF REGION DIR
24. YES
25. STATE NAME DOES NOT EXIST IN REGION
26. NO
27. QUERY OF STATE IN A REGION
28. QUERY OF STATE-STATE VALUENG
29. WRONG OPTION
30. STATE NAME DOES NOT EXIST IN REGION
31. RETURN
32. (FIG. 4-5)
FUNCTION SYSDT()

This routine scans array TOKEN for the keywords 'AS ON', 'AS ON', 'MONTH OF', 'MONTH'
'FROM', 'TO', 'TILL' and returns two dates as STARTING-DATE and END-DATE in YYMMDD format.

I <- 1, ID <- 1

TEMP <- TOKEN(1), TEMP1 <- TOKEN(1+i), TEMP2 <- TOKEN(1+i+2)

IF TEMP EQ AS NO IF TEMP EQ AS ON NO IF TEMP EQ MONTH NO IF TEMP EQ MONTH OF NO IF TEMP EQ FROM NO IF TEMP EQ ID' OR TILL' NO

YES YES YES YES YES YES NO

IF TEMP1 EQ ON NO IF TEMP1 EQ OF NO

DAT1 <- TEMP1

YES YES

DAT1 <- TEMP2

DATE MISSING

ERROR IN STRING

ID <- 2

IF I LE NO_TOT NO

YES

DAT2 <- YYMMDD

YES

IF ID EQ 2

CONVERT YYMMDD (- YYDDMM) YYDDMM

CALL SYSTEM DATE FUNCTION Datime()

RETURN
4.5.1.4 **Parser Routines**

4.5.1.4.1 **get-token ()**

(a) **FUNCTION** : 1. The user’s query on energy sector (as entered by user in English) is read character by character in array bfr[].

2. It is used to parse query into separate words/tokens.

(b) **INPUT** : User query in English.

(c) **OUTPUT** : Array token[] (array of string and variables),

   no_tok (number of words parsed).

(d) **ASSUMPTIONS** : 1. The user’s query is stored in bfr (a character array, buffer size = 20 characters).

2. Space character and/or <CR> ('/n') (carriage return) are separators.

3. If a non-blank character is found before <CR>, then <CR> is treated as separator, otherwise it signifies end of the query.

(e) **LOGIC** : 

   a) get-token breaks the user’s query into separate words (tokens).

   b) A stream of continuous characters (or a single character) excluding the space, tab, new line and other special characters is defined to be a token.

   c) These tokens are stored in a two dimensional array token and a counter no_tok indicates the total number of tokens in the query.
Algorithm:

```
initialize array token[];
i <- 0;
j <- 0;
no_tok <- 0;
do while bfr[i] not = CR
    do while bfr [i] = space
        i <- i + 1;
        if bfr [i] = CR;
            return;
        endif;
    enddo;
j <- 0;
no_tok <- no_tok + 1;
do while (while bfr[i] not = space) or (bfr[i] not = CR)
    token[no_tok-1][j] <- bfr[i];
j <- j + 1;
i <- i + 1;
enddo;
enddo;
```

4.5.1.4.2 Streq()

(a) FUNCTION : It is used for comparison of string of user's query with the string stored in directories.

(b) INPUT : Two strings S1, S2 which need to be matched

(c) OUTPUT : Returns TRUE, if matches successful; FALSE, if match is not found.

(d) ASSUMPTIONS : The two strings S1, S2 are passed as pointers.
(e) LOGIC : 1. First use a direct string comparison to check equality.

2. If (1) above, is not successful, convert both strings into respective Soundex codes and check their equality (pronunciation match)

3. If both (1) & (2) are not successful, then match only the first 4 characters for equality.

(f) ALGORITHM :

```
if string S1 = S2 : /* use string matching function
    strcmp (s1,s2) -a standard routine supplied with C */
    return TRUE;
endif;

Sdx_1 <- SOUNDEX (S1); /*SOUNDEX returns the soundex code */
Sdx_2 <- SOUNDEX (S2);
if Sdx_1 = Sdx_2
    return TRUE;
endif;
if first 4 char of S1 = first 4 char of S2
    /* strnicmp (S1,S2,4) */
    return TRUE;
endif;
return FALSE;
```

It may be noted that SOUNDEX routine returns a code depending on the pronunciation of the string. Thus, if the following are matched for equality: 'ADVICE', 'ADWYS', 'ADVISE', they will all return the same soundex code.
4.5.1.4.3 find_CAT()

(a) FUNCTION : Find the query category from user's query.
(b) INPUT : ARRAY token[], no_tok;
(c) OUTPUT : If match is successful, it returns query category otherwise returns an error code.
(d) LOGIC : 1) Based on the user's requirement, following list of possible query categories are created.
   1) CAPACITY
   2) GENERATION
   3) PLANT LOAD FACTOR
   4) OUTAGE
   5) FORCED OUTAGE
   6) FORCED_AND_MISCELLANEOUS OUTAGE / F_M OUTAGE
   7) MISCELLANEOUS OUTAGE
   8) OPERATING AVAILABILITY
   9) ADVISE
   10) COAL
   11) RESERVOIR

2) Tokens in the user query (and now in the array token[]) are matched against above list using the string equality logic of function STREQ (refer section 4.5.1.4.2)

3) If query category is found, the category code is returned otherwise an error code is generated.

4) query[] array contains categories. The total no of query categories are contained in a variable called no-of-query-topics.

(e) ALGORITHM

initialize the array query[] /* see above list */
i <- 0; /* query index */
j <- 0; /* token index */
no_of_query_topics <-- 11;

query_topic <- 0;  /* no query topic as yet */
for i = 0 to no_of_query_topics - 1 do
    for j = 0 to no_tok - 1 do
        if token[j] = query[i]
            query-topic <- i+1;
            return;
        endif;
    enddo;
enddo;

display "error : Unable to find a query category" ;
return;

4.5.1.4.4 cdsele()

(a) FUNCTION : It checks the combination of query types used by user in his query

(b) ASSUMPTIONS : The following hierarchies are defined in the system.

```
UNIT -> STATION -> STATE -> REGION -> INDIA
UNIT -> STATION -> SYSTEM -> SECTOR -> INDIA
```

(c) INPUT : 1. Uses the predefined arrays (directories) which have a list of all valid stations, units, sectors, states, Regions, systems.

2. Uses the global 2-D array token[].

(d) OUTPUT : Returns TRUE or FALSE (for valid and invalid query respectively).
(e) LOGIC: 1. For the current limited application of the energy database, two important hierarchies have been presented. (See ASSUMPTIONS (b) above).

2. The query contains at least one keyword from one of the above hierarchies. Otherwise, the query is rejected and will return a FALSE value.

3. All the keywords identified (if more than one) in the query, must fall only in one hierarchy. If they do not, then the query is treated invalid and is rejected (RETURN FALSE). For example: A query containing both the keywords STATE and SYSTEM or REGION and SYSTEM is rejected as it is not contained in hierarchy.

(f) ALGORITHM:

    h <- 0; /* h=0 means no hierarchy found yet */
    if keysec() /* "SECTOR" found */
        if h = 1
            display "error" ; /* as already we have a different hierarchy */
        endif;
    h <- 2;
    endif;
    if keyrg() /* "REGION" found */
        if h = 2
            display "error" ; /* as we already have a different hierarchy */
        endif;

144
h <- 1;
endif;
if keysys() /* "SYSTEM" found
    if h = 1
        display "error" ; /* as we already have a
different hierarchy */
    endif;
    h <- 2;
endif;
if keyst() /* state found */
    if h = 2
        display "error" ;
    endif;
    h <- 1;
endif;
if keystn() /* Station found */
    if h = 0
        h <- 1;
    endif;
endif;
if keyword "INDIA" found /* all India basis */
    h <- 9;
endif;
if h = 0 /* if no hierarchy found */
    display "error: no keyword found" ;
endif;
return;

4.5.1.4.5 \texttt{KEYxxx()} \\

(a) \textbf{FUNCTION} : The general format of routines which search state, station, system, region, sector etc. in array token are represented by \texttt{KEYxxx}, where \texttt{xxx} can be ST, SYS, RG, STN, SEC & OUT.\\

\textbf{e.g.} \texttt{KEYST()} --- Search for "STATE" \texttt{KEYSYS()} --- Search for "SYSTEM" \texttt{KEYRG()} --- Search for "REGION" \texttt{KEYSTN()} --- Search for "STATION" \texttt{KEYSEC()} --- Search for "SECTOR" \texttt{KEYOUT()} --- Search for "OUTAGE"

Since, the logic followed by above routines are similar so we have chosen function \texttt{KEYST()} which search for key word 'state'. Each of the above routines after comparison of relevant words, return various parameters as shown below:

\begin{tabular}{|l|l|}
\hline
\textbf{Function} & \textbf{Value returned} \\
\hline
\texttt{KEYST()} & State code, State name, Region code \\
\texttt{KEYSTN()} & State code, Station code, Station name, Region code \\
\texttt{KEYRG()} & Region code, Region name, \\
\texttt{KEYSYS()} & System code, System name, System category \\
\texttt{KEYSEC()} & Sector Code, Sector name \\
\texttt{KEYOUT()} & Outage code, Outage name, Outage type \\
\hline
\end{tabular}

4.5.1.4.6 \texttt{KEYST()}

(a) \textbf{INPUT/OUTPUT:} This function is called from function \texttt{CODEST} and returns various parameters described under the logic of \texttt{KEYST}.

(b) \textbf{FUNCTION} : Searches the array token [] for the keyword "STATE"
(c) LOGIC: 1. Search the array token for keyword "STATE"
   2. if not found, return control to main program
   3. If found, but available in the first token, then signal error as the state name should have preceded this keyword, return.
   4. Search the token before the keyword in the state-directory for a match.
   5. if found, return with the values of the following parameter.
      1. state name
      2. state code
      3. Region code
      4. Error flag
   6. if not found, signal error as invalid state name.

(d) ALGORITHM:
   i <-- 0;
   do while i < no_tok
      temp <-- token [i];
      if temp = "STATE"
         if i = 0
            display "error : State name missing" ;
            return;
         else
            temp = token [i-1];
            if temp = "ALL"
               STATE-CODE = 99;
               return;
            endif;
         endif;
      j <-- 0;
do while  j < no-of-state
    if temp = stdr[j].STNM
        STATE-CODE <-- stdr[j].STCD;
        STATE-NAME <-- stdr[j].STNM;
        REGION-CODE <-- stdr[j].REGD;
        /* State keyword and related information was found */
        return;
    endif;
    j <- j + 1;
enddo;

display "error : State name not found in state directory" ;
endif;
return;
endif;
i <-- i + 1;
enddo;
return;    /* state keyword not found */

4.5.1.4.7 CODEXXX()

(a) FUNCTION : (a) The format of the function is CODEXXX, where xxx is ST or STN or RG or OUT or SEC or SYS. The function of the COSEST, CODESTN, CODERG, CODEOUT, CODESEC and CODESYS is to find a detail query type within a State or Station or Region or Outage or Sector or System.

(b) Function KEYST, KEYSTN, KEYRG, KEYSYS, KEYSEC, KEYOUT are called from CODEST, CODESTN, CODERG, CODESYS, CODESEC, CODEOUT
and return various parameters after matching identifier as explained under functions of the KEYxxx. (refer section 4.5.1.4.5).

(c) This function returns a value under QRYOPT after identifying query type.

(b) **INPUT**: Array Token, no_tok

(c) **OUTPUT**: (a) QRYOPT

(b) Values returned by routines KEYST, KEYSTN, KEYRG, KEYSEC, KEYSYS, KEYOUT, KEYSYS. Since, the logic followed by CODExxx routines are similar so the logic of CODEST routine is explained in detail.

**4.5.1.4.8 CODEST()**

(a) **FUNCTION**: This routine will search the array token so as to find whether it will satisfy hierarchy (2) i.e., it first looks for the keyword STATE by calling the function KEYST. If the keyword is not found, CODEST returns an error signifying that the hierarchy is not satisfied. If the keyword state is found, search continues for the keyword REGION. If it is found, the keyword values are validated and the function returns a value under QRYOPT.

(b) **INPUT**: Array Token, no_tok

(c) **OUTPUT**: Query type value under QRYOPT

(d) **LOGIC**: 1. Search token[] for region.

2. if it is found in the first token, error.
3. The token before "REGION" is assumed to be region-name.

4. if region-name = "ALL", error.

5. Search region-name in region-directory.

6. if found 'Query of state in a region'.

7. if not found, error : State does not exist in region.

8. After identifying the Query type, the QRYOPT is set to different value described under QRYOPT table.

<table>
<thead>
<tr>
<th>QRYOPT CODE</th>
<th>QUERY NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. &quot;xxxxxx STATION STATUS IN DELHI STATE&quot;</td>
<td></td>
</tr>
<tr>
<td>2. &quot;ALL STATIONS IN XXXXX STATE&quot;</td>
<td></td>
</tr>
<tr>
<td>3. &quot;ALL STATIONS IN xxxxx REGION&quot;</td>
<td></td>
</tr>
<tr>
<td>4. &quot;ALL STATIONS IN INDIA&quot;</td>
<td></td>
</tr>
<tr>
<td>5. &quot;xxxxxx STATE IN NORTHERN REGION&quot;</td>
<td></td>
</tr>
<tr>
<td>6. &quot;ALL STATES IN xxxxx REGION&quot;</td>
<td></td>
</tr>
<tr>
<td>7. &quot;ALL STATES OF INDIA&quot;</td>
<td></td>
</tr>
<tr>
<td>8. &quot;xxxxx REGION OF INDIA&quot;</td>
<td></td>
</tr>
<tr>
<td>9. &quot;ALL REGIONS OF INDIA&quot;</td>
<td></td>
</tr>
<tr>
<td>10. &quot;QUERY OF ALL INDIA&quot;</td>
<td></td>
</tr>
<tr>
<td>11. &quot;QUERY OF xxxxxxx STATION BTPS SYSTEM&quot;</td>
<td></td>
</tr>
<tr>
<td>12. &quot;QUERY OF xxxxx STATION OF xxxxx SECTOR&quot;</td>
<td></td>
</tr>
<tr>
<td>13. &quot;QUERY OF ALL STATIONS OF xxxx SYSTEM&quot;</td>
<td></td>
</tr>
<tr>
<td>14. &quot;QUERY OF ALL STATIONS IN xxxxx SECTOR&quot;</td>
<td></td>
</tr>
<tr>
<td>15. &quot;Capacity of XXXXX system&quot;</td>
<td></td>
</tr>
<tr>
<td>16. &quot;All states in a sector&quot;</td>
<td></td>
</tr>
<tr>
<td>17. &quot;All system in country&quot;</td>
<td></td>
</tr>
<tr>
<td>18. &quot;All units in XXXXX sector&quot;</td>
<td></td>
</tr>
<tr>
<td>19. &quot;All units in a system&quot;</td>
<td></td>
</tr>
<tr>
<td>20. &quot;Capacity of XXXXX sector&quot;</td>
<td></td>
</tr>
<tr>
<td>21. &quot;xxxxx sector in XXXXX region&quot;</td>
<td></td>
</tr>
<tr>
<td>22. &quot;All sectors in XXXXX region&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Where XXXXX is the value of key item available in user query.
(e) ALGORITHM:

```plaintext
i <-- 0;
do while i < no_tok
    temp <-- token [i];
    if temp = "REGION"
        if i = 0
            display "error: region name missing";
            return;
        else
            temp <-- token [i-1];
            j <-- 0;
            if temp = "ALL"
                display "error: Invalid option";
                return;
            endif;
            do while j < NO-OF-REGIONS /* no. of Region =
                total Regions*/
                if temp = RGDIR [j].RGNM
                    if RGDIR[j].RECD = REGION-CODE
                        /* region code is obtained from region
directory while checking state */
                        /* query = query of state in region */
                        QRYOPT <-- 5;
                        return;
                    else
                        display "error: invalid region given";
                        return;
                    endif;
                endif;
            endif;
            j <-- j+1;
```
enndo;

display "error: region name not found";
enendif;
enendif;
i <-- i+1;
enndo;

4.5.1.4.9 SYSDT ()

(a) FUNCTION : This routine helps in searching date or dates used by user in his query. If dates are missing, it returns current date equal to system date.

(b) INPUT : token[]

(c) OUTPUT : Returns DATE1 and DATE2

(d) ASSUMPTIONS : DATIME is a system function which return the current date and time in required format.

(d) LOGIC : (a) For finding duration, array token is scanned to find the word 'FROM' or 'TO' or 'ASON' or 'AS ON' or 'TILL' or 'MONTHOF' or 'MONTH OF' or 'UP TO'.

(b) If pattern is missing, it assumes pattern as 'AS ON' and date supplied by user is treated junk word. The missing date is obtained from the system by calling function DATIME() which returns the current system date.

(c) The missing date is assumed as current date and is obtained from system in the YYMMDD format.
(e) ALGORITHM :

\begin{verbatim}
i <-- 0;
dat1 <-- 0;
dat2 <-- 0;
do while i < no_tok
    temp <-- token [i];
    temp1 <-- token [i+1];
    temp2 <-- token [i+2];
    if (temp = "ASON") OR (temp = /* We use streq for for testing string equality */
        "MONTHOF") OR (temp = "FROM")
        DATE1 <-- temp1;
    endif;
    if (temp = "AS" and temp1 = "ON") OR (temp = "MONTH" and temp1 = "OF")
        DATE1 <-- temp1;
    endif;
    if (temp = "TO") OR (temp = "TILL")
        OR (temp = "UP TO")
        DATE1 <-- temp2;
    endif;
    i <-- i + 1;
enddo;
if DATE1 = 0
    DATE1 <-- datetime ()
endif;
if DATE2 = 0
    DATE2 <-- datetime ()
endif;
\end{verbatim}
endif;
return;

4.5.2 Post Processing of Query

If stages in preprocessing (query-acceptance, scanning, understanding, formulation) is completed successfully, IA invokes IRS which is responsible for post processing of intermediate query as well as for providing an interface between NLCS/MBQS and database. The function of IRS is described as under

4.5.2.1 Information Retrieval System (IRS)

IRS invokes a procedure called query analyzer, it displays query (as understood by system) to user for user's concurrence on a user's console, then begins the post processing of query which requires:

(a) Conversion of parsed query into form acceptable to database.

(b) Assemble the information on parameters like name of database records, access path, owner-member relation (SET), in the format accepted by MDQS.

(c) Passing the information assembled through a module of IRS called ASSEM to MDQS which links this information and creates an object module containing users view.

(d) Retrieving records from database using RETRIEVE macro.

(e) Parameters required by RETRIEVE macro are passed to it by CRUN() function of IRS (refer section 4.5.2.1.1).

(f) Linking the information received from databases for printing as a result of query.
4.5.2.1.1 CRUN() functions of IRS

a) FUNCTION: (i) The function of CRUN is to convert the output of parser into mathematical for passing to various programs.
   (ii) The output of CRUN is used for linking database record and its associated information.

b) INPUT: Query category, Query type, query value, code values of owner and member record (The parameters are the output of parser routines).

c) OUTPUT: CRUN generates an output in the form of statements as described below:
   CRUN xxxx(a1, a2, a3, ......an)
   Where XXXX name of program procedure for linking parameters of user query
   XXXX - CAPA for capacity category,
   GEN for generation category,
   PLF for Plant Load Factor,
   OUT for outage,
   (a1, a2, ......an) - List of arguments as per the user's query.

d) LOGIC: During query scanning phase, query category is determined by function find_cat(). Also, query type is obtained from the following six sets of routines:
   (a) KEYST(), CODEST() - query is on STATE
   (b) KEYSTN(), CODESTN() - query is on STATION
   (c) KEYSEC(), CODESEC() - query is on SECTOR
   (d) KEYSYS(), CODESYS() - query is on SYSTEM
   (e) KEYRG(), CODERG () - query is on REGION
   (f) KEYOUT(), CODEOUT() - query is on OUTAGE

155
The output of above routine are various parameters (refer 4((a) to (f)) of section 4.5.1.2). These parameters are concatenated into a single line statement as shown under subtitle "OUTPUT" at (c) above.

e) ALGORITHM:

OUTPUT of routines described under logic of CRUN() uses following logic to convert sub-queries into mathematical form.

if query category = "CAPACITY"
    date <-- DAT1;
    if query on "STATION"
        cd1 <-- STATE-CODE;
        cd2 <-- STATION-CODE;
        opt <-- 1;
    elseif query on "STATE"
        cd1 <-- STATE-CODE;
        cd2 <-- 0;
        opt <-- 2;
    elseif query on "REGION"
        cd1 <-- REGION-CODE;
        cd2 <-- 0;
        opt <-- 3;
    elseif query on "SYSTEM"
        cd1 <-- SYSTEM-CODE;
        cd2 <-- 0;
        opt <-- 4;
    elseif query on "ALL-INDIA"
        cd1 <-- 0;
        cd2 <-- 0;
        opt <-- 5;
else
    display "error : Invalid topic on capacity query";
    return;
endif;

CRUN CAPA(opt, cd1, cd2, date);

elseif query category = "GENERATION"
    date_from <-- DAT1;
    date_to <-- DAT2;
    if query on "STATION"
        cd1 <-- STATE-CODE;
        cd2 <-- STATION-CODE;
        opt <-- 1;
    elseif query on "STATE"
        cd1 <-- STATE-CODE;
        cd2 <-- 0;
        opt <-- 2;
    elseif query on "REGION"
        cd1 <-- REGION-CODE;
        cd2 <-- 0;
        opt <-- 3;
    elseif query on "ALL-INDIA"
        cd1 <-- 0;
        cd2 <-- 0;
        opt <-- 4;
    elseif query on "SYSTEM"
        cd1 <-- SYSTEM-CODE;
        cd2 <-- 0;
        opt <-- 5;
else
    elseif query on sector
else
    display "error: Invalid topic on generation query";
    return;
endif;

CRUN GENA(opt, cd1, cd2, date_from, date_to)

elseif query category = "PLANT CODE FACTOR"
    date_from <-- DAT1;
    date_to <-- DAT2;
    if query on "UNIT"
        cd1 <-- STATE-CODE;
        cd2 <-- STATION-CODE;
        cd3 <-- UNIT-CODE;
        opt <-- 1;
    elseif query on "STATION"
        cd1 <-- STATE-CODE;
        cd2 <-- STATION-CODE;
        cd3 <-- 0;
        opt <-- 2;
    elseif query on "STATE"
        cd1 <-- STATE-CODE;
        cd2 <-- 0;
        cd3 <-- 0;
        opt <-- 3;
    elseif query on "REGION"
        cd1 <-- REGION-CODE;
        cd2 <-- 0;
cd3 <-- 0;
opt <-- 4;

elseif query on "ALL-INDIA"
   cd1 <-- 0;
   cd2 <-- 0;
   cd3 <-- 0;
   opt <-- 5;

elseif query on "SYSTEM"
   cd1 <-- SYSTEM-CODE;
   cd2 <-- 0;
   cd3 <-- 0;
   opt <-- 6;
else
   display "error : Invalid topic on PLF query";
   return;
endif;

CRUN PLF(opt, cd1, cd2, cd3, date_from, date_to)

elseif query category = "OUTAGES"

date_from <-- DAT1;

date_to <-- DAT2;

cd2 <-- OUTAGE_CAT;
   /* cd2 = 01 for Planning outage category */
   /* cd2 = 02 for Reserved outage category */
   /* cd2 = 03 for forced outage category */
   /* cd2 = 04 for outage category */

   cd3 <-- outage-type;
   if query on "STATE"
      cd1 <-- STATE-CODE;
      opt <-- 1;
elseif query on "SYSTEM"
    cd1 <-- SYSTEM-CODE;
    opt1 <-- 2;
elseif query on "SECTOR"
    cd1 <-- SECTOR-CODE;
    opt <-- 3;
elseif query on "REGION"
    cd1 <-- REGION-CODE;
    opt <-- 4;
elseif query on "ALL INDIA"
    cd1 <-- 0;
    cd2 <-- 0;
    opt <-- 5;
else
    display "error : Invalid query topic";
    return;
endif;

CRUN OUTAGE (opt, cd1, cd2, cd3,
        date_from, date_to);

else
    display "error : Invalid topic on query category" ;
    return;
endif;
return;

4.5.2.1.2 Assembling of Information to link Database

Assembling of information required to satisfy a query is achieved by IRS through ASSEM module which picks the data from
data file depending upon query, converts it (as per syntax of Application Definition Language) in the format required by MDQS. The details of data file and the steps to link this information to query, are described as under.

1. The data file consists of query category/query type, query entry point name, record(s) and its storage mode or set name. Consider following query for illustration:

"LIST ALL STATIONS IN DELHI STATE"

In above query, parser identifies query type and invokes IRS to get station information of Delhi state. To satisfy above query, following two records are needed

(a) STATE Record to search Delhi state
(b) STATION RECORD to get station record corresponding to Delhi state.

2. Accessing the above Database records, need the information about its mode of storage i.e. VIA Mode or through a SET or VIA an entry point.

3. The information required for a query is available in a data file against an entry point.

4. During post processing of query, Query is linked to query entry point.

5. Query entry point and information defined against it along with database name is passed to MDQS (vendor supplied) which picks up the record and its associated information and creates an object module.

6. The query parameters which are already converted to a mathematical form (using CRUN function), are linked with the above object module for the retrieval of information.
For Example

(a) RETRIEVE REGION-STATE-ENTRY
    WHERE REGION-CODE <= 2

(b) RETRIEVE REGION-STATE-ENTRY
    WHERE REGION-CODE = 02

The execution of statement at (a) above will list all state name pertaining to Region 1 (Northern INDIA) and REGION 2 (Western INDIA). While the execution of statement 2, will list states of only Region 2 i.e. Western region. As already stated, the value '2' of region code is obtained from region directory and is passed through CRUN function to appropriate procedure used for generating report.

4.5.2.2 Creation Of Object module using MDQS

MDQS uses assembled information of IRS, checks the syntax of statement (i.e., scanning each statement of the file containing assembled information) and finally, it builds a symbols table for data item, levels, subscripts, arrays and routine in object file called Application Definition File (ADF). This object ADF is used during post processing.

4.5.2.3 Procedure for making a formatted reports

Following procedure is followed while generating a report for a query submitted on a terminal

1. Invoke the object ADF file and use the clauses/statements described below under serial (2) to (13).
2. REPORT Statement is used to define characteristics of report, name, file etc.
3. PAGE HEADING for defining lines and space along with heading to be produced at the top of page.
4. PAGE FOOTING for defining the line and space to be produced at the bottom of page along with required foot note.

5. PAGE LENGTH is used to define number of print lines required to be printed on a page.

6. PAGE WIDTH is used to define the number of characters that may be printed across a line.

7. STARTING PAGE NUMBER clause is used to define starting page number of the report.

8. COLUMN clause is used to define the spacing between the columns.

9. TITLE clause is used to define column titles.

10. PICTURE clause is used to define format of elements on a report line.

11. RETRIEVE statement is used to access a database records defined under a ENTRY-NAME.

12. PRINT statement is used to define the destination of report and number of elements to be printed on a print line.

13. 'END' statement is used to show the physical end of report procedure.

14. Steps 2 to 13 described above are used in ADF file for report preparation.

4.5.2.4 Query Processing

An interactive session between user and IA has been described through sample dialogues. The query processing is divided in two parts (i) Query through Interactive mode (ii) Query through a batch file submitted in batch mode. The query is processed as per the sequence described in figure 4-7:
4.5.2.4.1 Steps required for Query Processing

The steps which are followed in query parsing, scanning and processing are summarised as under:

(i) Process user query/requests from user terminal or through a file in array bfr.

(ii) Call get_token() for parsing query and ignoring junk words.

(iii) CALL CDSELE() to identify query Hierarchy and FIND-CAT() to find query category.

(iv) CALL CODEXXX() and KEYXXX() repeatedly to find:

(a) QUERY type-Main TYPE

(b) Query sub_type within MAIN-TYPE

Where xxx is STN or ST or SEC or SYS or REG or OUT depending upon if query is on a) station, b) state, c) sector, d) system, e) region and f) outage.

If error, return error code.

(v) If KEYxxx() not successful, it returns with a message "INVALID QUERY TYPE".

(vi) Call SYSDT() which performs following:

(a) extraction of system date if date is missing in user's query,

(b) validating the date(s) of query and providing it to IA.
(vii) CALL IRS() which performs following:
   (a) taking user's consent before processing,
   (b) to convert query into a mathematical form,
   (c) Assemble the information required for formation of user's view from the data file and convert it in ADL format and store it in a file for later use.

(viii) Creation of object module using MDQS
Call MDQS compiler for compiling the information already stored in a file [refer step vii (c) above] and store the compiled information as object module (OBJ) for use at later stage.

(ix) Processing of Query
- Invoke object module (OBJ).
- Use retrieve macro to get the required database records.
- Parameters passed by the CRUN function of IRS are used in retrieve statement to match with the values of items of database records.
- Link the items retrieved from databases as per query.
- PRINT the items linked as query results.

4.5.2.5 Illustrative example on retrieval of data

The steps used in retrieval of data corresponding to a query, is explained using an illustrative example shown below:
Query - List thermal stations in country?

4.5.2.5.1 Logic
(a) Parser will parse above query and send its output to IRS.
(b) IRS will convert it into mathematical form and points to query-entry-point of array structure for assembling
information on database records corresponding to query.
(c) Output of IRS is sent to MDQS which generates an object file - OBJ.
(d) Information supplied by IRS to MDQS is again used for accessing information along with OBJ file.
(e) In above case, IRS assembles following three entry points from array structure as per the structure of database - ENEDB.
   (1) REGN-ENT
   (2) REGN-STAT-ENT
   (3) STN-TYPE-ENT
(f) For retrieving information as per above query, user has to navigate from country (AI-record) to station (STN-record) through Region and state Record (refer figure 2-5A) and output will be consisting of state code, state name, station code, station name under each region. Let us examine, how the assembled information is used for processing above query.
   (i) REGN-ENT will provide information of owner [ALL INDIA record (ai-R)], member [REGION record (regn-R)] and SET (AI-REGN-S) for linking owner and member record.
   (ii) REGN-STAT-ENT will provide owner [REGION record regn-R], member [STATE record (state-R)], SET (REGN-STATE-S) for linking REGION record to STATE record.
   (iii) STN-TYPE-ENT will provide information on 3 sets which will be used for linking the records of the following 3 groups.
(g) Use OBJ file (already created by MDQS) and RETRIEVE macro to retrieve information using entry-point defined above. The OBJ file makes available to user all records. The SET information (assembled by IRS) is used for linking the records required in following sequence for the above query:

- (a) AI to REGION
- (b) REGION to STATE
- (c) STATE to STATION
- (d) STATION TO STATION-TYPE

(h) Logic of linking
(1) Start from All India record and get all regions VIA AI-REGN-S (AI-REGN-S is a set created in between All-India and Region records).
(2) We pick up first region code from REGION record and using REGN-STATE-S we get all states corresponding to region code (REGN-STATE-S is a set created in between Region and state record).
(3) We pick up first state and using entry STN-TYPE-ENT, STATE is linked to STATION and STATION to STATION-TYPE. We pick up first station and print it. This process is repeated till all the station records within a state are over.
(4) We pick up next state and steps described in h(3) are repeated.
(5) When all states are over, we pick up next region and steps h(1) to h(4) are repeated. The logic of preparation of report and printing of retrieved items are written using report writer.

4.5.2.5.2 Algorithm

In section 4.5.2.5.1, (g) describes the sequence in which records are to be accessed, (f) describes the steps to be followed. Here, we are describing the assumption, records required and logic used in data retrieval.

Assumption

if the content of db_status is zero at any stage, then the operation performed by the macro (like FIND etc) is treated successful.

(ii) Record names used in above query

<table>
<thead>
<tr>
<th>Record Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ai-R</td>
<td>All-India record</td>
</tr>
<tr>
<td>region-R</td>
<td>Region record</td>
</tr>
<tr>
<td>state-R</td>
<td>State record</td>
</tr>
<tr>
<td>station-R</td>
<td>Station record</td>
</tr>
<tr>
<td>station-type-R</td>
<td>Station type record</td>
</tr>
<tr>
<td>station-subtype-R</td>
<td>Station sub-type record</td>
</tr>
</tbody>
</table>

(iii) Logic used in retrieval

```
db_status=0;

RETRIEVE REGN-ENT where ai-cd = 1;
    /* ai-cd = All India code */

reg <-- regn-cd;    /* transfer first value of region-code to temporary variable reg */
    /* ai-r and region-r available */

RETRIEVE REGN-STATE-ENT where regn-cd = reg;

get region-r;
    /* region-r and state-r is available */
    /* pick all state records which has regn-cd=reg */
```

168
statecd <- state-cd; /* store the first
  state-code into statecd */

RETRIEVE STN-TYPE-ENT where state-cd = statecd;
and station-type = "t" or "c";

/* make available state-r, station-r, state-sub-
  type, station-type */

/* picks up a station record
  which has station type = thermal */

L-40 : get station-record;

display region state-cd state-name
  station-cd station name;
find next station-r;
if db_status not=zero /* all station within state over */
  go to L-100;
else
  go to L-40;
endif;

L-100 : find next state-r; /* through reg-state set */

L-150 : if db_status not=zero /* all state are over */
  go to L-200;
else
  pick up first station corresponding to st_cd;
  go to L-40;
endif;

L-200 : find next region-r;
if db_status not=zero /* all region over */
  go to L-400;
else
  find state-r; /*picks up first state to
    region code */
  go to L-150;
endif;
L-400: exit;

4.5.2.6 Query Session

Query analyzer can be invoked by any valid user by giving command IA against system prompt as shown below:

SYSTEM ? GET IA
SYSTEM ? IA

System responds to user with following
"Please type your Query"

= 

Now user can enter a query in English as described in the following queries. The notation 'Exp' is used for explanation of query and 'Result' for output of query.

QUERY 1

User: DISPLAY CAPACITY OF BADARPUR

Exp: In this query, user has not indicated what is BADARPUR? Whether it is STATION or STATE or SYSTEM or REGION. Hence query is rejected and returned to user for correcting as shown below.

RESULT: QUERY CATEGORY = CAPACITY
QUERY incomplete, needs correction

QUERY 2

User: LIST CAPACITY OF BADARPUR STATION IN DETAIL

Exp: In this query, USER has forgotten to give date on which he wants to see status. SYSTEM fetches current date from system. Query analyzer scans query and displays following to user for his concurrence:
QUERY CATEGORY = CAPACITY
STATION = BADARPUR
DATE = MISSING
DATE ASSUMED = 910595

SHOULD I continue processing (enter YES or NO) ?

If the user types 'NO', query is rejected and control is returned to user for re entering the query.

If the option 'Yes' is selected, query is processed and following result is returned to user.

RESULT

STATION WISE DETAILS OF CAPACITY
AS ON : 90/10/25
STATION : BADARPUR

<table>
<thead>
<tr>
<th>UNIT NO</th>
<th>CAPACITY (MW)</th>
<th>INS-DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.000</td>
<td>73/11/ 1</td>
</tr>
<tr>
<td>2</td>
<td>100.000</td>
<td>74/ 8/ 1</td>
</tr>
<tr>
<td>3</td>
<td>100.000</td>
<td>75/ 3/29</td>
</tr>
<tr>
<td>4</td>
<td>210.000</td>
<td>78/12/ 2</td>
</tr>
<tr>
<td>5</td>
<td>210.000</td>
<td>81/12/25</td>
</tr>
<tr>
<td>TOTAL</td>
<td>720.000</td>
<td></td>
</tr>
</tbody>
</table>

QUERY 3
User : LIST CAPACITY OF I.P. STATION AS ON 910101
Exp : The query is on CAPACITY as on 1st Jan. '1991. IA permits to use abbreviation I.P. for INDRA PRASTHA and replaces it with actual station name during the processing of the query.

RESULT

STATION WISE DETAILS OF CAPACITY
AS ON : 91/01/01
STATION : INDRA PRASTHA STATION
CAPACITY : 282.000

QUERY 4
User : WHAT IS THE GENERATION OF IP STATION FOR THE PERIOD FROM 8704 TO 8706
Exp : IA permits abbreviation "IP" for INDRA PRASTHA and
extracts information on generation and other relevant parameters. For example: The fields 'TARGET' and '% of ACHIEVEMENT' in the result shown below are not asked by user in his query but are useful from the point of monitoring requirement.

RESULT

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>STATION WISE DETAILS ENERGY GENERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>87/04 TO 87/06</td>
<td>STATION: INDRA PRASTHA STATION</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNIT NO</th>
<th>TARGET (GWH)</th>
<th>ACTUAL (GWH)</th>
<th>% OF ACHIEVEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>36.00</td>
<td>40.42</td>
<td>112</td>
</tr>
<tr>
<td>2</td>
<td>91.00</td>
<td>94.83</td>
<td>104</td>
</tr>
<tr>
<td>3</td>
<td>91.00</td>
<td>93.34</td>
<td>103</td>
</tr>
<tr>
<td>4</td>
<td>30.00</td>
<td>111.76</td>
<td>373</td>
</tr>
<tr>
<td>5</td>
<td>91.00</td>
<td>91.33</td>
<td>100</td>
</tr>
</tbody>
</table>

| TOTAL   | 339.00       | 431.68       | 127             |

QUERY 5

User: PERIOD FROM 87/01 TO 87/01 GENERATION OF ROPER STATION

Exp: In this query, English grammar rules has been violated but query conveys the meaning, hence it is processed.

RESULT

<table>
<thead>
<tr>
<th>PERIOD</th>
<th>STATION WISE DETAILS ENERGY GENERATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>87/01 TO 87/01</td>
<td>STATION: ROPER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UNIT NO</th>
<th>TARGET (GWH)</th>
<th>ACTUAL (GWH)</th>
<th>% OF ACHIEVEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>107.00</td>
<td>154.01</td>
<td>144</td>
</tr>
<tr>
<td>2</td>
<td>116.00</td>
<td>143.39</td>
<td>124</td>
</tr>
</tbody>
</table>

| TOTAL   | 223.00       | 297.40       | 133             |

172
QUERY 6
User :  GIVE ME CAPACITY OF BADARXXX STATION
Exp  :  In this query, value of station is wrong but since first few characters are correct, it is accepted and results are processed as shown in Query 2. STATION NAME IS CORRECTED AS ‘BADARPUR’.

QUERY 7
User :  DISPLAY CAPACITY OF HAR STATION
Exp  :  In this question, value of station is HAR. There are 2 stations starting with this string, namely ‘HARDUAGANJ A’ and ‘HARDUAGANJ B’. Both the stations along with codes are displayed to user and the user is asked to select one out of the two choices. The result is processed as per Query 2 (also refer 4 of section 4.4)

QUERY 8
User :  GIVE GENERATION OF ALL UNITS OF DELHI STATE
Exp  :  Query is for all unit of Delhi State. System gets information from ‘STATE’ record, ‘STATION’ record and ‘Unit’ record for answering this Query. The result of this query is similar to query 5 except generation figure for all units will appear.

QUERY 9
User :  WHAT IS PLF OF DELHI state from 8706 to 8707 ?
Exp  :  Query is ON PLF. PLF calculation requires information from ‘Generation’, ‘Capacity’, ‘No-PLF-Date’ and ‘Unit’ record. The records corresponding to above are invoked and following report is generated.
RESULT

PLF REPORT FOR THE STATE DELHI
PERIOD: FROM - 87/06 TO 87/07

<table>
<thead>
<tr>
<th>MONTH</th>
<th>TOTAL</th>
<th>CAPACITY</th>
<th>GENERATION</th>
<th>PLF(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>07</td>
<td>1196.500</td>
<td>441.24</td>
<td>60.65</td>
</tr>
<tr>
<td>07</td>
<td>07</td>
<td>1196.500</td>
<td>433.30</td>
<td>56.85</td>
</tr>
</tbody>
</table>

STATE TOTAL FOR THE PERIOD

874.54 58.72

Query 10

User: list plf of Badarpur station 5th Unit from 87/01 to 87/01

Exp: Query is on unit number 5 of station BADARPUR. PLF has been asked for JAN 87.

RESULT

STATION WISE PLF REPORT FOR THE
PERIOD: FROM - 87/01 TO 87/01
STATION: BADARPUR

<table>
<thead>
<tr>
<th>MONTH</th>
<th>UNIT-NO</th>
<th>INS-CAP (MW)</th>
<th>INS-DATE</th>
<th>CAPACITY (MW)</th>
<th>GENERATION (GWH)</th>
<th>P-L-F (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>05</td>
<td>210.000</td>
<td>81/12/25</td>
<td>210.000</td>
<td>137.01</td>
<td>87.69</td>
</tr>
</tbody>
</table>

QUERY 11

User: list all station of Delhi state

Exp: Delhi is valid state and has four stations.

RESULT

State name: Delhi, State code: 1

Station code Station name

1 badarpur
2 d.e.s.u. gas turbine.
3 i.p. station
4 rajghat

QUERY 12

User: give me advice on badarpur station in the period from 87/04 to 87/05

174
Exp : SYSTEM asks user -

'Do you want to see criteria stored ? YES

When User types YES, following is displayed

(if plf <= 45% - performance is poor; If plf > 45% & < 70% - performance is good; If plf >= 70% - performance is very good.)

DO you agree with above advise ? YES

In this case, user replies YES, hence experts advice are generated based on the criteria displayed.

RESULT :

UNIT WISE PERFORMANCE REPORT FOR THE
PERIOD : FROM - 87/04 TO 87/05
STATION : BADARPUR

<table>
<thead>
<tr>
<th>MONTH</th>
<th>UNIT NO.</th>
<th>PLF (%)</th>
<th>PERFORMANCE</th>
<th>PLANNED MAINTENANCE</th>
<th>EXPERT'S ADVISE</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>01</td>
<td>80.47</td>
<td>VERY GOOD</td>
<td>87/04/01 87/04/30</td>
<td>POSTPONE MAINTENANCE</td>
</tr>
<tr>
<td>04</td>
<td>02</td>
<td>32.15</td>
<td>POOR</td>
<td>87/12/01 87/12/31</td>
<td>NEEDS PREPONEMENT</td>
</tr>
<tr>
<td>04</td>
<td>03</td>
<td>56.92</td>
<td>GOOD</td>
<td>88/02/01 88/02/28</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>63.61</td>
<td>GOOD</td>
<td>87/09/01 87/10/31</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>41.79</td>
<td>POOR</td>
<td>87/06/01 87/06/15</td>
<td>NEEDS PREPONEMENT</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>55.48</td>
<td>GOOD</td>
<td>87/06/01 87/06/15</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
</tbody>
</table>

Query 13

User : give me advice on badarpur station in the period from 8704 to 8705.

Exp : This query is similar to query 12 but in this query, user does not want to use the stored knowledge, he wants to use his own expertise hence, he says NO in reply to following question :

"Do you want to see advice stored ?" NO

System then asks user to enter new criteria interactively as
shown below :

Enter new advise for category=poor in 2 digits specifying lower and upper limits
= 0, 35

Enter new advise for category=good in two digits specifying lower and upper limits
= 36, 75

Enter new advise for category=very good in two digits specifying lower and upper limits
= 76, 99

SHOULD I CONTINUE Processing? YES

In this query, the criteria of evaluation is different, although the data is same as of query 12, hence the expert advise shown in result is different in comparison to query 12.

RESULT:

UNIT WISE PERFORMANCE REPORT FOR THE
PERIOD : FROM - 87/04 TO 87/05
STATION : BADARPUR

<table>
<thead>
<tr>
<th>MONTH</th>
<th>UNIT NO.</th>
<th>PLF (%)</th>
<th>PERFORMANCE</th>
<th>PLANNED MAINTENANCE SCHEDULE DATES</th>
<th>EXPERT'S ADVICE</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>01</td>
<td>80.47</td>
<td>VERY GOOD</td>
<td>87/04/01 87/04/30</td>
<td>POSTPONE MAINTENANCE</td>
</tr>
<tr>
<td>04</td>
<td>02</td>
<td>32.15</td>
<td>POOR</td>
<td>87/12/01 87/12/31</td>
<td>NEEDS PREPONEMENT</td>
</tr>
<tr>
<td>04</td>
<td>03</td>
<td>56.92</td>
<td>GOOD</td>
<td>88/02/01 88/02/28</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>63.61</td>
<td>GOOD</td>
<td>87/09/01 87/10/31</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
<tr>
<td>04</td>
<td>05</td>
<td>41.79</td>
<td>GOOD</td>
<td>87/06/01 87/06/15</td>
<td>PROCEED AS PER SCHEDULE</td>
</tr>
</tbody>
</table>

Query 14

User: give the status of plf of Delhi state from 8705 to 8705

Exp: System displays PLF of all stations in May 1987.
RESULT:

PLF REPORT FOR THE STATE DELHI
PERIOD: FROM - 87/05 TO 87/05

<table>
<thead>
<tr>
<th>STATION</th>
<th>MONTH</th>
<th>CAPACITY (MW)</th>
<th>GENERATION (GWH)</th>
<th>PLF(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BADARPUR</td>
<td>05</td>
<td>720.000</td>
<td>322.00</td>
<td>60.11</td>
</tr>
<tr>
<td>DESU GAS TURBINES</td>
<td>05</td>
<td>180.000</td>
<td>00.20</td>
<td>0.00</td>
</tr>
<tr>
<td>I.P. STATION</td>
<td>05</td>
<td>282.500</td>
<td>149.35</td>
<td>71.06</td>
</tr>
<tr>
<td>RAJCHAT</td>
<td>05</td>
<td>14.000</td>
<td>00.00</td>
<td>0.00</td>
</tr>
<tr>
<td><strong>STATE TOTAL</strong></td>
<td></td>
<td><strong>1196.500</strong></td>
<td><strong>471.55</strong></td>
<td><strong>63.20</strong></td>
</tr>
</tbody>
</table>

4.5.2.7 **Query termination**

Once report is generated, IA again asks user whether he would like to ask more queries? If user types 'YES', system waits for another query, otherwise database is closed and session is terminated.