Loop Processing

A processing loop is a group of statements which are executed repeatedly until a stated condition has been satisfied, or as long as a certain condition prevails.

This chapter covers the following topics:

- Use of Processing Loops
- Limiting Database Loops
- Limiting Non-Database Loops - REPEAT Statement
- Example of REPEAT Statement
- Terminating a Processing Loop - ESCAPE Statement
- Loops Within Loops
- Example of Nested FIND Statements
- Referencing Statements within a Program
- Example of Referencing with Line Numbers
- Example with Statement Reference Labels

Use of Processing Loops

Processing loops can be subdivided into database loops and non-database loops:

- **Database processing loops**
  are those created automatically by Natural to process data selected from a database as a result of a `READ`, `FIND` or `HISTOGRAM` statement. These statements are described in the section *Database Access*.

- **Non-database processing loops**
  are initiated by the statements `REPEAT`, `FOR`, `CALL FILE`, `CALL LOOP`, `SORT`, and `READ WORK FILE`.

More than one processing loop may be active at the same time. Loops may be embedded or nested within other loops which remain active (open).

A processing loop must be explicitly closed with a corresponding `END-...` statement (for example, `END-REPEAT`, `END-FOR`)

The `SORT` statement, which invokes the sort program of the operating system, closes all active processing loops and initiates a new processing loop.

Limiting Database Loops

The following topics are covered below:

- Possible Ways of Limiting Database Loops
- LT Session Parameter
- LIMIT Statement
- Limit Notation
- Priority of Limit Settings

Possible Ways of Limiting Database Loops

With the statements READ, FIND or HISTOGRAM, you have three ways of limiting the number of repetitions of the processing loops initiated with these statements:

- using the session parameter LT,
- using a LIMIT statement,
- or using a limit notation in a READ/FIND/HISTOGRAM statement itself.

LT Session Parameter

With the system command GLOBALS, you can specify the session parameter LT, which limits the number of records which may be read in a database processing loop.

Example:

```
GLOBALS LT=100
```

This limit applies to all READ, FIND and HISTOGRAM statements in the entire session.

LIMIT Statement

In a program, you can use the LIMIT statement to limit the number of records which may be read in a database processing loop.

Example:

```
LIMIT 100
```

The LIMIT statement applies to the remainder of the program unless it is overridden by another LIMIT statement or limit notation.

Limit Notation

With a READ, FIND or HISTOGRAM statement itself, you can specify the number of records to be read in parentheses immediately after the statement name.

Example:

```
READ (10) VIEWXYZ BY NAME
```

This limit notation overrides any other limit in effect, but applies only to the statement in which it is specified.

Priority of Limit Settings

If the limit set with the LT parameter is smaller than a limit specified with a LIMIT statement or a limit notation, the LT limit has priority over any of these other limits.

Limiting Non-Database Loops - REPEAT Statement

Non-database processing loops begin and end based on logical condition criteria or some other specified limiting condition.

The REPEAT statement is discussed here as representative of a non-database loop statement.

With the REPEAT statement, you specify one or more statements which are to be executed repeatedly. Moreover, you can specify a logical condition, so that the statements are only executed either until or as long as that condition is met. For this purpose you use an UNTIL or WHILE clause.

If you specify the logical condition

- in an UNTIL clause, the REPEAT loop will continue *until* the logical condition is met;
- in a WHILE clause, the REPEAT loop will continue *as long as* the logical condition remains true.

If you specify *no* logical condition, the REPEAT loop must be exited with one of the following statements:

- ESCAPE terminates the execution of the processing loop and continues processing outside the loop (see below).
- STOP stops the execution of the entire Natural application.
- TERMINATE stops the execution of the Natural application and also ends the Natural session.

Example of REPEAT Statement

```
** Example 'REPEAX01': REPEAT
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW VIEW OF EMPLOYEES
  2 NAME
  2 SALARY (1:1)
*
1 #PAY1    (N8)
END-DEFINE
```

```
*
READ (5) MYVIEW BY NAME WHERE SALARY (1) = 30000 THRU 39999
  MOVE SALARY (1) TO #PAY1
  /*
  REPEAT WHILE #PAY1 LT 40000
    MULTIPLY #PAY1 BY 1.1
    DISPLAY NAME (IS=ON) SALARY (1)(IS=ON) #PAY1
  END-REPEAT
  /*
  SKIP 1
END-READ
END
```

Output of Program REPEAX01:

```
Page      1                                                  04-11-11  14:15:54

        NAME           ANNUAL     #PAY1
                       SALARY
------------------- ---------- ---------

ADKINSON                   34500     37950
                                     41745

                           33500     36850
                                     40535

                           36000     39600
                                     43560

AFANASSIEV                 37000     40700

ALEXANDER                  34500     37950
                                     41745
```

Terminating a Processing Loop - ESCAPE Statement

The ESCAPE statement is used to terminate the execution of a processing loop based on a logical condition.

You can place an ESCAPE statement within loops in conditional IF statement groups, in break processing statement groups (AT END OF DATA, AT END OF PAGE, AT BREAK), or as a stand-alone statement implementing the basic logical conditions of a non-database loop.

The ESCAPE statement offers the options TOP and BOTTOM, which determine where processing is to continue after the processing loop has been left via the ESCAPE statement:

- ESCAPE TOP is used to continue processing at the top of the processing loop.
- ESCAPE BOTTOM is used to continue processing with the first statement following the processing loop.

You can specify several ESCAPE statements within the same processing loop.

For further details and examples of the ESCAPE statement, see the *Statements* documentation.

Loops Within Loops

A database statement can be placed within a database processing loop initiated by another database statement. When database loop-initiating statements are embedded in this way, a "hierarchy" of loops is created, each of which is processed for each record which meets the selection criteria.

Multiple levels of loops can be embedded. For example, non-database loops can be nested one inside the other. Database loops can be nested inside non-database loops. Database and non-database loops can be nested within conditional statement groups.

Example of Nested FIND Statements

The following program illustrates a hierarchy of two loops, with one `FIND` loop nested or embedded within another `FIND` loop.

```
** Example 'FINDX06': FIND (two FIND statements nested)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 PERSONNEL-ID
1 VEH-VIEW VIEW OF VEHICLES
  2 MAKE
  2 PERSONNEL-ID
END-DEFINE
*
FND1. FIND EMPLOY-VIEW WITH CITY = 'NEW YORK' OR = 'BEVERLEY HILLS'
  FIND (1) VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FND1.)
    DISPLAY NOTITLE NAME CITY MAKE
  END-FIND
END-FIND
END
```

The above program selects data from multiple files. The outer `FIND` loop selects from the `EMPLOYEES` file all persons who live in New York or Beverley Hills. For each record selected in the outer loop, the inner `FIND` loop is entered, selecting the car data of those persons from the `VEHICLES` file.

Output of Program `FINDX06`:

```
        NAME                 CITY                 MAKE

-------------------- -------------------- --------------------

RUBIN                NEW YORK             FORD
OLLE                 BEVERLEY HILLS       GENERAL MOTORS
WALLACE              NEW YORK             MAZDA
JONES                BEVERLEY HILLS       FORD
SPEISER              BEVERLEY HILLS       GENERAL MOTORS
```

Referencing Statements within a Program

Statement reference notation is used for the following purposes:

- Referring to previous statements in a program in order to specify processing over a particular range of data.
- Overriding Natural's default referencing.
- Documenting.

Any Natural statement which causes a processing loop to be initiated and/or causes data elements in a database to be accessed can be referenced, for example:

- READ
- FIND
- HISTOGRAM
- SORT
- REPEAT
- FOR

When multiple processing loops are used in a program, reference notation is used to uniquely identify the particular database field to be processed by referring back to the statement that originally accessed that field in the database.

If a field can be referenced in such a way, this is indicated in the Referencing Permitted column of the *Operand Definition Table* in the corresponding statement description (in the *Statements* documentation). See also *User-Defined Variables*, *Referencing of Database Fields Using (r) Notation*.

In addition, reference notation can be specified in some statements. For example:

- AT START OF DATA
- AT END OF DATA
- AT BREAK
- ESCAPE BOTTOM

Without reference notation, an AT START OF DATA, AT END OF DATA or AT BREAK statement will be related to the *outermost* active READ, FIND, HISTOGRAM, SORT or READ WORK FILE loop. With reference notation, you can relate it to another active processing loop.

If reference notation is specified with an ESCAPE BOTTOM statement, processing will continue with the first statement following the processing loop identified by the reference notation.

Statement reference notation may be specified in the form of a *statement reference label* or a *source-code line number*.

- **Statement reference label**
  A statement reference label consists of several characters, the last of which must be a period (.). The period serves to identify the entry as a label.

A statement that is to be referenced is marked with a label by placing the label at the beginning of the line that contains the statement. For example:

```
0030 ...
0040 READ1. READ VIEWXYZ BY NAME
0050 ...
```

In the statement that references the marked statement, the label is placed in parentheses at the location indicated in the statement's syntax diagram (as described in the *Statements* documentation). For example:

```
AT BREAK (READ1.) OF NAME
```

- **Source-code line number**
  If source-code line numbers are used for referencing, they must be specified as 4-digit numbers (leading zeros must not be omitted) and in parentheses. For example:

  ```
  AT BREAK (0040) OF NAME
  ```

  In a statement where the label/line number relates a particular field to a previous statement, the label/line number is placed in parentheses after the field name. For example:

  ```
  DISPLAY NAME (READ1.) JOB-TITLE (READ1.) MAKE MODEL
  ```

Line numbers and labels can be used interchangeably.

See also *User-Defined Variables*, *Referencing of Database Fields Using (r) Notation*.

Example of Referencing with Line Numbers

The following program uses source code line numbers (4-digit numbers in parentheses) for referencing.

In this particular example, the line numbers refer to the statements that would be referenced in any case by default.

```
0010 ** Example 'LABELX01': Labels for READ and FIND loops (line numbers)
0020 ************************************************************************
0030 DEFINE DATA LOCAL
0040 1 MYVIEW1 VIEW OF EMPLOYEES
0050   2 NAME
0060  2 FIRST-NAME
0070  2 PERSONNEL-ID
0080 1 MYVIEW2 VIEW OF VEHICLES
0090  2 PERSONNEL-ID
0100  2 MAKE
0110 END-DEFINE
0120 *
0130 LIMIT 15
0140 READ MYVIEW1 BY NAME STARTING FROM 'JONES'
0150  FIND MYVIEW2 WITH PERSONNEL-ID = PERSONNEL-ID (0140)
0160    IF NO RECORDS FOUND
0170      MOVE '***NO CAR***' TO MAKE
0180    END-NOREC
0190    DISPLAY NOTITLE NAME       (0140) (IS=ON)
0200                    FIRST-NAME (0140) (IS=ON)
```

```
0210                    MAKE       (0150)
0220  END-FIND /* (0150)
0230 END-READ   /* (0140)
0240 END
```

Example with Statement Reference Labels

The following example illustrates the use of statement reference labels.

It is identical to the previous program, except that labels are used for referencing instead of line numbers.

```
** Example 'LABELX02': Labels for READ and FIND loops (user labels)
************************************************************************
DEFINE DATA LOCAL
1 MYVIEW1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 MYVIEW2 VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
RD. READ MYVIEW1 BY NAME STARTING FROM 'JONES'
  FD. FIND MYVIEW2 WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      MOVE '***NO CAR***' TO MAKE
    END-NOREC
    DISPLAY NOTITLE NAME       (RD.) (IS=ON)
                    FIRST-NAME (RD.) (IS=ON)
                    MAKE       (FD.)
  END-FIND /* (FD.)
END-READ   /* (RD.)
END
```

Both programs produce the following output:

```
        NAME              FIRST-NAME              MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               GENERAL MOTORS
                     LAUREL               GENERAL MOTORS
                     KEVIN                DATSUN
                     GREGORY              FORD
JOPER                MANFRED              ***NO CAR***
JOUSSELIN            DANIEL               RENAULT
JUBE                 GABRIEL              ***NO CAR***
JUNG                 ERNST                ***NO CAR***
JUNKIN               JEREMY               ***NO CAR***
KAISER               REINER               ***NO CAR***
KANT                 HEIKE                ***NO CAR***
```