Schleifenverarbeitung

Eine Verarbeitungsschleife ist eine Gruppe von Statements, deren Ausführung so oft wiederholt wird, bis eine bestimmte Bedingung erfüllt ist, oder solange eine bestimmte Bedingung gegeben ist.

Dieses Kapitel behandelt folgende Themen:

- Verwendung von Verarbeitungsschleifen
- Schleifendurchläufe bei Datenbankzugriffen begrenzen
- Durchläufe bei Nicht-Datenbankzugriffsschleifen begrenzen — das REPEAT-Statement
- Beispiel für Verarbeitungsschleife mit REPEAT-Statement
- Verarbeitungsschleife verlassen — das ESCAPE-Statement
- Schleifen innerhalb von Schleifen
- Beispiel für geschachtelte FIND-Statements
- Statements innerhalb eines Programms referenzieren
- Beispiel für das Referenzieren mit Zeilennummern
- Beispiel mit Statement-Labels

Verwendung von Verarbeitungsschleifen

Verarbeitungsschleifen lassen sich in Datenbankschleifen und Nicht-Datenbankschleifen unterteilen:

- **Datenbankschleifen**
  werden von Natural automatisch erzeugt, um die Daten, die mit einem `READ`-, `FIND`- oder `HISTOGRAM`-Statement von einer Datenbank gelesen werden, zu verarbeiten. Diese Statements sind im Kapitel *Datenbankzugriffe* beschrieben.

- **Nicht-Datenbankschleifen**
  (d.h. Schleifen ohne Datenbankzugriff) werden mit folgenden Statements erzeugt: `REPEAT`, `FOR`, `CALL FILE`, `CALL LOOP`, `SORT` und `READ WORK FILE`.

Es können mehrere Verarbeitungsschleifen gleichzeitig aktiv sein. In einer gerade aktiven, d.h. noch nicht abgeschlossenen Schleife können weitere Schleifen eingebettet werden.

Jede Verarbeitungsschleife muss durch ein entsprechendes `END-...`-Statement beendet werden (z.B. `END-REPEAT`, `END-FOR` usw.).

Das `SORT`-Statement, mit dem das Sortierprogramm des Betriebssystems aufgerufen wird, beendet alle aktiven Schleifen und löst eine neue Schleife aus.

Schleifendurchläufe bei Datenbankzugriffen begrenzen

Die folgenden Themen werden behandelt:

- Möglichkeiten der Begrenzung von Datenbankschleifen
- LT-Session-Parameter
- LIMIT-Statement
- Limit-Notation
- Priorität der Limit-Einstellungen

Möglichkeiten der Begrenzung von Datenbankschleifen

Bei den Statements READ, FIND oder HISTOGRAM haben Sie drei Möglichkeiten, die Anzahl, wie oft eine Verarbeitungsschleife durchlaufen werden soll, zu begrenzen:

- mit dem Session-Parameter LT
- mit einem LIMIT-Statement
- oder mit einer Limit-Notation im READ-/FIND-/HISTOGRAM-Statement selbst.

LT-Session-Parameter

Mit dem Systemkommando GLOBALS können Sie den Session-Parameter LT angeben, der die Anzahl der Datensätze, die in einer Datenbank-Verarbeitungsschleife gelesen werden sollen, begrenzt.

Beispiel:

```
GLOBALS LT=100
```

Dieses Limit gilt für alle READ-, FIND- und HISTOGRAM-Schleifen in der gesamten Session.

LIMIT-Statement

In einem Programm können Sie die Anzahl der Datensätze, die in einer Datenbank-Verarbeitungsschleife gelesen werden sollen, mit einem LIMIT-Statement begrenzen.

Beispiel:

```
LIMIT 100
```

Das LIMIT-Statement gilt für alle nachfolgenden READ-, FIND- oder HISTOGRAM-Schleifen im Programm, es sein denn, es wird durch ein anderes LIMIT-Statement oder eine Limit-Notation außer Kraft gesetzt.

Limit-Notation

In einem READ-, FIND- oder HISTOGRAM-Statement selbst können Sie die Anzahl der Datensätze, die gelesen werden sollen, in Klammern unmittelbar hinter dem Statement-Namen angeben.

Beispiel:

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

Diese Limit-Notation hat Vorrang vor allen anderen Limits, gilt aber nur für das betreffende Statement.

Priorität der Limit-Einstellungen

Wenn das mit dem LT-Parameter angegebene Limit kleiner ist als ein mit einem LIMIT-Statement oder einer Limit-Notation angegebenes, dann hat das LT-Limit Vorrang vor diesen anderen Limits.

Durchläufe bei Nicht-Datenbankzugriffsschleifen begrenzen — das REPEAT-Statement

Anfang und Ende von Verarbeitungsschleifen, die keinen Datenbankzugriff beinhalten, basieren auf einer logischen oder sonstwie die Schleife begrenzenden Bedingung. Sie werden mit einem der folgenden Statements erzeugt: REPEAT, FOR, CALL FILE, CALL LOOP, SORT und READ WORK FILE.

Stellvertretend für Nicht-Datenbankschleifen-Statements wird hier das Statement REPEAT behandelt.

Mit dem REPEAT-Statement geben Sie ein oder mehrere Statements an, die wiederholt ausgeführt werden sollen. Außerdem können Sie eine logische Bedingung angeben, so dass die Statements nur ausgeführt werden, solange oder bis diese Bedingung erfüllt ist. Die Bedingung geben Sie in einer UNTIL-Klausel oder in einer WHILE-Klausel an:

- Bei einer UNTIL-Klausel wird die Schleife so oft ausgeführt, bis (UNTIL) die logische Bedingung erfüllt ist, d.h. die Schleife wird beendet, sobald der in der Bedingung angegebene Zustand erreicht ist.
- Bei einer WHILE-Klausel wird die REPEAT-Schleife ausgeführt, während (WHILE) der in der Bedingung angegebene Zustand besteht, d.h. die Schleife wird beendet, sobald die Bedingung nicht mehr erfüllt wird.

Wenn Sie *keine* logische Bedingung angeben, muss die REPEAT-Schleife mit einem der folgenden Statements verlassen werden:

- ESCAPE (siehe nächsten Abschnitt) beendet die Verarbeitung der Schleife und setzt die Verarbeitung außerhalb der Schleife fort.
- STOP bricht die Ausführung der gesamten Natural-Anwendung ab.
- TERMINATE bricht die Ausführung der Natural-Anwendung ab und beendet die Natural-Session.

Beispiel für Verarbeitungsschleife mit 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
```

Ausgabe des Programms 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
```

Verarbeitungsschleife verlassen — das ESCAPE-Statement

Mit dem ESCAPE-Statement können Sie die Ausführung einer Verarbeitungsschleife abbrechen, und zwar aufgrund einer logischen Bedingung.

Das ESCAPE-Statement kann Teil eines IF-Statements sein oder an eines der Statements AT END OF DATA, AT END OF PAGE oder AT BREAK geknüpft sein; es kann aber auch als eigenständiges Statement in Ausführung der einer Verarbeitungsschleife zugrundeliegenden logischen Bedingungen stehen.

Mit dem ESCAPE-Statement haben Sie die Optionen TOP und BOTTOM, mit denen Sie festlegen, wo die Verarbeitung fortgesetzt werden soll, nachdem die Schleife mit ESCAPE verlassen wurde:

- Bei ESCAPE TOP wird die Verarbeitung am Anfang der Schleife, d.h. mit dem nächsten Schleifendurchlauf, fortgesetzt.
- Bei ESCAPE BOTTOM wird die Verarbeitung mit dem ersten Statement, das nach der Schleife kommt, fortgesetzt.

Sie können innerhalb einer Verarbeitungsschleife auch mehrere ESCAPE-Statements angeben.

Weitere Informationen und Beispiele zum ESCAPE-Statement finden Sie in der *Statements*-Dokumentation.

Schleifen innerhalb von Schleifen

Mit Natural haben Sie die Möglichkeit, Schleifen innerhalb von Schleifen auszulösen und so eine ganze "Hierarchie" ineinander verschachtelter Schleifenkonstruktionen aufzubauen. Sind mehrere Datenbankzugriffsschleifen ineinander verschachtelt, so durchläuft jeder gelesene Datensatz, der die Auswahlkriterien erfüllt, nacheinander die einzelnen Schleifen, bevor der nächste Datensatz verarbeitet wird.

Mehrere Datenbankzugriffs- und Nicht-Datenbankzugriffsschleifen können ineinander verschachtelt werden. Verarbeitungsschleifen können auch Teil einer bedingten Verarbeitung sein.

Beispiel für geschachtelte FIND-Statements

Das folgende Programm zeigt eine Hierarchie zweier Verarbeitungsschleifen, wobei sich eine FIND-Schleife innerhalb einer anderen FIND-Schleife befindet.

```
** 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
```

Das obige Programm liest Daten von mehreren Dateien. Die äußere FIND-Schleife wählt von der EMPLOYEES-Datei alle Personen aus, die in New York oder Beverley Hills wohnen. Für jeden in der äußeren Schleife ausgewählten Datensatz wird die innere FIND-Schleife durchlaufen, in der die Fahrzeugdaten der betreffenden Personen von der VEHICLES-Datei gelesen werden.

Ausgabe des Programms 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
```

Statements innerhalb eines Programms referenzieren

Statement-Referenzierung dient dazu

- in einem Programm auf ein vorhergehendes Statement zu verweisen (d.h. dieses Statement zu "referenzieren"), um eine Verarbeitung für einen bestimmten Bereich von Daten auszuführen,
- Naturals Standard-Referenzierung (die bei jedem betroffenen Statement in der Dokumentation beschrieben ist) aufzuheben
- oder zu Programmdokumentationszwecken.

Sie können jedes Natural-Statement referenzieren, das eine Verarbeitungsschleife initiiert und/oder auf Datenelemente in einer Datenbank zugreift:

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

Enthält ein Programm mehrere Verarbeitungsschleifen, so kann man ein bestimmtes Datenbankfeld eindeutig identifizieren, indem man das Statement referenziert, welches zuerst auf das entsprechende Feld in der Datenbank zugriff.

Welche Felder bei welchem Statement referenziert werden dürfen, ersehen Sie in der Statements-Dokumentation in den *Operandentabellen* der einzelnen Statements aus der Spalte *Referenzierung erlaubt*. Siehe auch *Benutzervariablen*, *Datenbankfelder mit der (r)-Notation referenzieren*.

Außerdem kann eine Referenzierungsnotation in einigen Statements angegeben werden, z.B. bei:

- AT START OF DATA
- AT END OF DATA

- AT BREAK
- ESCAPE BOTTOM

Normalerweise bezieht sich bei einem AT START OF DATA-, AT END OF DATA- oder AT BREAK-Statement die schleifenbeendende Gruppenwechsel-Bedingung auf die jeweils äußerste aktive READ-, FIND-, HISTOGRAM-, SORT- oder READ WORK FILE-Schleife. Mit einer Referenzierungsnotation können Sie die Bedingung auf eine beliebige andere aktive Schleife beziehen.

Wenn Sie bei einem ESCAPE BOTTOM-Statement ein Statement referenzieren, wird die Verarbeitung unmittelbar nach der durch das referenzierte Statement identifizierten Schleife fortgesetzt.

Zur Statement-Referenzierung können Sie entweder ein sogenanntes *Statement-Label* oder die *Sourcecode-Zeilennummer* verwenden.

- **Statement-Label**
  Ein Statement-Label ist eine Zeichenkette, deren letztes Zeichen ein Punkt (.) sein muss. Der Punkt identifiziert die Zeichenkette als Label.

  Ein Statement, das referenziert werden soll, wird mit einem Label markiert, indem das Label an den Anfang der Zeile gestellt wird, in der das Statement steht, zum Beispiel:

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

  In dem Statement, das das markierte Statement referenziert, wird das Label in Klammern an der in der Statement-Syntax dafür vorgesehenen Stelle (siehe Syntaxdiagramme in der *Statements*-Dokumentation) eingefügt, zum Beispiel:

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

- **Sourcecode-Zeilennummern**
  Wenn Sie Sourcecode-Zeilennummern zur Referenzierung verwenden, müssen Sie diese immer vierstellig (vorangestellte Nullen dürfen nicht weggelassen werden) und in Klammern angeben, zum Beispiel:

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

  Bezieht sich in einem Statement ein bestimmtes Feld auf ein vorhergegangenes Statement, so wird das Label bzw. die Zeilennummer in Klammern hinter dem jeweiligen Feldnamen angegeben, zum Beispiel:

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

Sourcecode-Zeilennummern und Statement-Labels können wahlweise verwendet werden.

Siehe auch *Benutzervariablen, Datenbankfelder mit der (r)- Notation referenzieren*.

Beispiel für das Referenzieren mit Zeilennummern

Das folgende Programm verwendet Sourcecode-Zeilennummern (vierstellige Ziffern in Klammern) zur Referenzierung.

In diesem Beispiel beziehen sich die Zeilennummern auf Statements, die aufgrund der Programmstruktur ohnehin, auch ohne explizite Referenzierung, referenziert worden wären.

```
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
```

Beispiel mit Statement-Labels

Das folgende Beispiel zeigt die Verwendung von Statement-Labels.

Es ist mit dem vorigen Beispielprogramm identisch bis auf die Tatsache, dass zur Referenzierung der Statements Labels anstelle von Zeilennummern verwendet werden.

```
** 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
```

Beide Programme erzeugen folgende Ausgabe:

```
        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***
```