Dynamische und große Variablen benutzen

Dieses Kapitel behandelt folgende Themen:

- Allgemeine Informationen zu dynamischen Variablen
- Zuweisungen mit dynamischen Variablen
- Initialisierung dynamischer Variablen
- String-Manipulation mit dynamischen alphanumerischen Variablen
- Logische Bedingungen (LCC) bei dynamischen Variablen
- AT/IF-BREAK dynamischer Kontrollfelder
- Parameter-Übertragung mit dynamischen Variablen
- Arbeitsdateizugriff bei großen und dynamischen Variablen
- DDM-Generierung und Editieren von Spalten mit variabler Länge
- Zugriff auf große Datenbankobjekte
- Performance-Aspekte bei dynamischen Variablen
- Ausgabe von dynamische Variablen
- Dynamische X-Arrays

Allgemeine Informationen zu dynamischen Variablen

Im Allgemeinen kann eine dynamische alphanumerische Variable immer dann benutzt werden, wenn ein alphanumerisches Feld zulässig ist. Ein dynamisches binäres Feld kann immer dann benutzt werden, wenn ein binäres Feld erlaubt ist. Ein dynamisches Unicode-Feld kann überall dort verwendet werden, wo ein Unicode-Feld erlaubt ist.

Ausnahme:

Dynamische Variablen sind nicht zulässig beim `SORT`-Statement. Um dynamische Variablen in einem `DISPLAY`-, `WRITE`-, `PRINT`-, `RREINPUT`- oder `INPUT`-Statement zu benutzen, müssen Sie entweder den Session-Parameter `AL` oder `EM` benutzen, um die Länge der Variable zu definieren.

Die benutzte Länge (siehe Natural-Systemvariable `*LENGTH`, *Aktuell für eine dynamische Variable benutzter Wertespeicher*) und die Größe des zugewiesenen Speicherplatzes der dynamischen Variablen sind gleich Null, bis auf die Variable als Ziel-Operand zum ersten Mal zugegriffen wird. Aufgrund von Zuweisungen oder anderen Operationen können dynamische Variablen zuerst zugewiesen oder auf die exakte Größe des Source-Operanden erweitert werden (neu zugewiesen).

Die Größe einer dynamischen Variable kann mit den folgenden Statements erweitert werden, wenn sie als ein änderbarer Operand (Ziel-Operand) in den folgenden Statements benutzt wird:

ASSIGN	*operand1* (Ziel-Operand in einer Zuweisung).
CALLNAT	Siehe *Parameter-Übertragung bei dynamischen Variablen* (außer wenn `AD=O`, oder wenn `BY VALUE` in der entsprechenden PDA vorhanden ist)
COMPRESS	*operand2*, siehe *Verarbeitung*.
EXAMINE	*operand1* in der `DELETE REPLACE`-Klausel.
MOVE	*operand2* (Ziel-Operand), siehe *Funktion*.
PERFORM	Außer wenn `AD=O`, oder wenn `BY VALUE` in der betreffenden PDA vorhanden ist.
READ WORK FILE	*operand1* und *operand2*, siehe *Verarbeitung großer und dynamischer Variablen*.
SEPARATE	*operand4*.
SELECT (SQL)	Parameter in der `INTO`-Klausel.
SEND METHOD	*operand3* (außer wenn `AD=O`).

Zur Zeit gibt es die folgende Beschränkung in Bezug auf die Verwendung von großen Variablen:

CALL	Parameter-Größe kleiner als 64 KB pro Parameter (keine Beschränkung für `CALL` mit INTERFACE4-Option).

In den folgenden Abschnitten wird die Benutzung der dynamischen Variablen detaillierter und mit Beispielen erörtert.

Zuweisungen mit dynamischen Variablen

Im Allgemeinen wird eine Zuweisung in der zurzeit benutzten Länge des Source-Operanden durchgeführt (siehe Natural-Systemvariable `*LENGTH`). Wenn der Ziel-Operand eine dynamische Variable ist, wird seine zurzeit zugewiesene Größe gegebenenfalls erweitert, um den Source-Operanden ohne Abschneiden zu verschieben.

Beispiel:

```
#MYDYNTEXT1 := OPERAND
MOVE OPERAND TO #MYDYNTEXT1
/* #MYDYNTEXT1 IS AUTOMATICALLY EXTENDED UNTIL THE SOURCE OPERAND CAN BE COPIED
```

`MOVE ALL` bzw. `MOVE ALL UNTIL` mit dynamischen Ziel-Operanden wird wie folgt definiert:

- `MOVE ALL` verschiebt den Source-Operanden wiederholt zum Ziel-Operanden, bis die benutzte Länge (`*LENGTH`) des Ziel-Operanden erreicht ist. `*LENGTH` wird nicht geändert. Wenn `*LENGTH` Null ist, wird das Statement ignoriert.

- `MOVE ALL` *`operand1`* `TO` *`operand2`* `UNTIL operand3` verschiebt *operand1* wiederholt zu *operand2*, bis die in *operand3* angegebene Länge erreicht ist. Wenn *operand3* größer als `*LENGTH(`*`operand2`*`)` ist, wird *operand2* erweitert, und `*LENGTH(`*`operand2`*`)` wird auf *operand3* gesetzt. Wenn *operand3* kleiner als `*LENGTH(`*`operand2`*`)` ist, wird die benutzte Länge auf *operand3* reduziert. Wenn *operand3* gleich `*LENGTH(`*`operand2`*`)` ist, entspricht das Verhalten dem bei `MOVE ALL`.

Beispiel:

```
#MYDYNTEXT1 := 'ABCDEFGHIJKLMNO'          /* *LENGTH(#MYDYNTEXT1) = 15
MOVE ALL 'AB' TO #MYDYNTEXT1              /* CONTENT OF #MYDYNTEXT1 = 'ABABABABABABABA';
                                          /* *LENGTH IS STILL 15
MOVE ALL 'CD' TO #MYDYNTEXT1 UNTIL 6      /* CONTENT OF #MYDYNTEXT1 = 'CDCDCD';
                                          /* *LENGTH = 6
MOVE ALL 'EF' TO #MYDYNTEXT1 UNTIL 10     /* CONTENT OF #MYDYNTEXT1 = 'EFEFEFEFEF';
                                          /* *LENGTH = 10
```

`MOVE JUSTIFIED` wird zur Kompilierungszeit zurückgewiesen, wenn der Ziel-Operand eine dynamische Variable ist.

`MOVE SUBSTR` und `MOVE TO SUBSTR` sind zulässig. `MOVE SUBSTR` führt zu einem Laufzeitfehler, wenn ein Substring hinter der benutzten Länge einer dynamischen Variable (`*LENGTH`) referenziert wird. `MOVE TO SUBSTR` führt zu einem Laufzeitfehler, wenn eine Substring-Position hinter `*LENGTH + 1` referenziert wird, weil dies zu einer undefinierten Lücke im Inhalt der dynamischen Variable führen würde. Wenn der Ziel-Operand von `MOVE TO SUBSTR` erweitert werden sollte (zum Beispiel, wenn der zweite Operand auf `*LENGTH+1` gesetzt wird), ist der dritte Operand zwingend.

Gültige Syntax:

```
#OP2 := *LENGTH(#MYDYNTEXT1)
MOVE SUBSTR (#MYDYNTEXT1, #OP2) TO OPERAND                /* MOVE LAST CHARACTER TO OPERAND
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #lEN_OPERAND)   /* CONCATENATE OPERAND TO #MYDYNTEXT1
```

Ungültige Syntax:

```
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE SUBSTR (#MYDYNTEXT1, #OP2, 10) TO OPERAND           /* LEADS TO RUNTIME ERROR; UNDEFINED SUB-STRING
#OP2 := *LENGTH(#MYDYNTEXT1 + 10)
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2, #EN_OPERAND)   /* LEADS TO RUNTIME ERROR; UNDEFINED GAP
#OP2 := *LENGTH(#MYDYNTEXT1) + 1
MOVE OPERAND TO SUBSTR(#MYDYNTEXT1, #OP2)                /* LEADS TO RUNTIME ERROR; UNDEFINED LENGTH
```

Zuweisungskompatibilität

Beispiel:

```
#MYDYNTEXT1   := #MYSTATICVAR1
#MYSTATICVAR1 := #MYDYNTEXT2
```

Wenn der Source-Operand eine statische Variable ist, wird die benutzte Länge des dynamischen Ziel-Operanden (`*LENGTH(#MYDYNTEXT1)`) auf die Format-Länge der statischen Variablen gesetzt, und der Source-Wert wird einschließlich nachfolgender Leerzeichen (alphanumerische und Unicode-Felder) oder binärer Nullen (für binäre Felder) in diese Länge kopiert.

Wenn der Ziel-Operand statisch und der Source-Operand dynamisch ist, wird die dynamische Variable in ihre zurzeit benutzte Länge kopiert. Wenn diese Länge kleiner als die Format-Länge der statischen Variable ist, wird der Rest mit Leerzeichen (für alphanumerische und Unicode-Felder) oder binären Nullen (für binäre Felder) aufgefüllt, sonst wird der Wert abgeschnitten. Wenn die aktuell benutzte Länge der dynamischen Variable Null (0) ist, wird der statische Ziel-Operand mit Leerzeichen (für alphanumerische und Unicode-Felder) oder binären Nullen (für binäre Felder) aufgefüllt.

Initialisierung dynamischer Variablen

Dynamische Variablen können mit einem RESET-Statement mit Leerzeichen (alphanumerische und Unicode-Felder) oder Nullen (binäre Felder) bis zur aktuell benutzten Länge (= *LENGTH) initialisiert werden. *LENGTH wird nicht geändert.

Beispiel:

```
DEFINE DATA LOCAL
  1 #MYDYNTEXT1   (A)  DYNAMIC
END-DEFINE
#MYDYNTEXT1 := 'SHORT TEXT'
WRITE *LENGTH(#MYDYNTEXT1)          /* USED LENGTH = 10
RESET #MYDYNTEXT1                   /* USED LENGTH = 10, VALUE = 10 BLANKS
```

Um eine dynamische Variable mit einem angegebenen Wert in einer angegebenen Länge zu initialisieren, kann das MOVE ALL UNTIL-Statement benutzt werden.

Beispiel:

```
MOVE ALL 'Y' TO #MYDYNTEXT1 UNTIL 15     /* #MYDYNTEXT1 CONTAINS 15 'Y'S, USED LENGTH = 15
```

String-Manipulation mit dynamischen alphanumerischen Variablen

Wenn ein änderbarer Operand eine dynamische Variable ist, wird ihre zurzeit zugewiesene Länge möglicherweise erhöht, um die Operation ohne Abschneidung oder Fehlermeldung auszuführen. Dies gilt für die Verkettung (COMPRESS) und Trennung von dynamischen alphanumerischen Variablen (SEPARATE).

Beispiel:

```
** Example 'DYNAMX01': Dynamic variables (with COMPRESS and SEPARATE)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1 (A)   DYNAMIC
1 #TEXT       (A20)
1 #DYN1       (A)   DYNAMIC
1 #DYN2       (A)   DYNAMIC
1 #DYN3       (A)   DYNAMIC
END-DEFINE
*
MOVE ' HELLO WORLD ' TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading and trailing blanks
*
MOVE ' HELLO WORLD ' TO #TEXT
*
```

```
MOVE #TEXT TO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with whole variable length of #TEXT
*
COMPRESS #TEXT INTO #MYDYNTEXT1
WRITE #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  dynamic variable with leading blanks of #TEXT
*
*
#MYDYNTEXT1 := 'HERE COMES THE SUN'
SEPARATE #MYDYNTEXT1 INTO #DYN1 #DYN2 #DYN3 IGNORE
*
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
WRITE #DYN1 (AL=25) 'with length' *LENGTH (#DYN1)
WRITE #DYN2 (AL=25) 'with length' *LENGTH (#DYN2)
WRITE #DYN3 (AL=25) 'with length' *LENGTH (#DYN3)
/*  #DYN1, #DYN2, #DYN3 are automatically extended or reduced
*
EXAMINE #MYDYNTEXT1 FOR 'SUN' REPLACE 'MOON'
WRITE / #MYDYNTEXT1 (AL=25) 'with length' *LENGTH (#MYDYNTEXT1)
/*  #MYDYNTEXT1 is automatically extended or reduced
*
END
```

Anmerkung:
Im Falle von nicht-dynamischen Variablen kann eine Fehlermeldung zurückgegeben werden.

Logische Bedingungen (LCC) bei dynamischen Variablen

Im Allgemeinen erfolgt eine Lese-Operation (z.B. ein Vergleich) bei einer dynamischen Variablen mit ihrer zurzeit benutzten Größe. Dynamische Variablen werden wie statische Variablen verarbeitet, wenn sie in einem Lese-Kontext (nicht änderbar) benutzt werden.

Beispiel:

```
IF  #MYDYNTEXT1  =  #MYDYNTEXT2  OR  #MYDYNTEXT1 =  "**"  THEN ...
IF  #MYDYNTEXT1  <  #MYDYNTEXT2  OR  #MYDYNTEXT1 <  "**"  THEN ...
IF  #MYDYNTEXT1  >  #MYDYNTEXT2  OR  #MYDYNTEXT1 >  "**"  THEN ...
```

Auch im Falle von nachfolgenden Leerzeichen oder führenden Nullen zeigen dynamische Variablen ein entsprechendes Verhalten.

Für dynamische Variablen ist der alphanumerische Wert AA gleich AA, und der binäre Wert `00003031` ist gleich `3031`.

Führende Nullen für alphanumerische und Unicode-Variablen oder führende binäre Nullen für binäre Variablen werden bei statischen und dynamischen Variablen gleich behandelt. Zum Beispiel werden alphanumerische Variable, die die Werte AA und AA (d.h. AA mit nachfolgendem Leerzeichen) enthalten, als gleich angesehen. Binäre Variablen, die beispielsweise die Werte H'0000031' und H'3031' enthalten, werden ebenso als gleich angesehen. Wenn ein Vergleichsergebnis nur im Falle einer exakten Kopie wahr (TRUE) sein sollte, müssen die benutzten Längen der dynamischen Variablen außerdem miteinander verglichen werden. Wenn eine Variable eine exakte Kopie der anderen ist, sind ihre benutzten Längen auch gleich.

Beispiel:

```
#MYDYNTEXT1  :=  'HELLO'                                    /* USED LENGTH IS 5
#MYDYNTEXT2  :=  'HELLO     '                               /* USED LENGTH IS 10
IF   #MYDYNTEXT1  =  #MYDYNTEXT2   THEN  ...                /* TRUE
IF   #MYDYNTEXT1  =  #MYDYNTEXT2 AND
     *LENGTH(#MYDYNTEXT1) = *LENGTH(#MYDYNTEXT2)  THEN  ...  /* FALSE
```

Zwei dynamische Variablen werden Position für Position miteinander verglichen (von links nach rechts bei alphanumerischen Variablen und von rechts nach links bei binären Variablen) bis zum Minimum ihrer benutzten Längen. Die erste Position, an der die Variablen nicht gleich sind, legt fest, ob die erste oder zweite Variable größer, gleich oder kleiner als die andere ist. Die Variablen sind gleich, wenn sie bis zum Minimum ihrer benutzten Längen gleich sind, und der Rest der längeren Variable nur Leerzeichen (bei alphanumerischen dynamischen Variablen) oder binäre Nullen (bei dynamischen binären Variablen) enthält. Um zwei dynamische Unicode-Variablen miteinander zu vergleichen, werden aus beiden Werten die führenden Nullen entfernt, bevor zum Vergleich der beiden resultierenden Werte der ICU-Collation-Algoritmus benutzt wird. Siehe auch *Logical Condition Criteria* in der *Unicode and Code Page Support*-Dokumentation.

Beispiel:

```
#MYDYNTEXT1  :=  'HELLO1'                   /* USED LENGTH IS 6
#MYDYNTEXT2  :=  'HELLO2'                   /* USED LENGTH IS 10
IF #MYDYNTEXT1  <  #MYDYNTEXT2 THEN  ...    /* TRUE
#MYDYNTEXT2  :=  'HALLO'
IF #MYDYNTEXT1  >  #MYDYNTEXT2 THEN  ...    /* TRUE
```

Vergleichskompatibilität

Vergleiche zwischen dynamischen und statischen Variablen sind gleichwertig mit Vergleichen zwischen dynamischen Variablen. Die Format-Länge der statischen Variable wird als ihre benutzte Länge interpretiert.

Beispiel:

```
#MYSTATTEXT1 :=  'HELLO'                     /* FORMAT LENGTH OF MYSTATTEXT1 IS A20
#MYDYNTEXT1  :=  'HELLO'                     /* USED LENGTH IS 5
IF  #MYSTATTEXT1  =  #MYDYNTEXT1  THEN  ...  /* TRUE
IF  #MYSTATTEXT1  >  #MYDYNTEXT1  THEN  ...  /* FALSE
```

AT/IF-BREAK dynamischer Kontrollfelder

Der Vergleich des Gruppenwechsel-Kontrollfeldes mit seinem alten Wert wird Position für Position von links nach rechts durchgeführt. Wenn der alte und der neue Wert der dynamischen Variable jeweils unterschiedliche Längen hat, dann wird zu Vergleichszwecken der Wert mit der kürzeren Länge nach rechts aufgefüllt (mit Leerzeichen für alphanumerisch und Unicode (dynamische Werte oder binäre Nullen für binäre Werte).

Im Falle eines alphanumerischen oder Unicode-Gruppenwechselkontrollfeldes sind nachfolgende Leerzeichen nicht bedeutend für den Vergleich, d.h. nachfolgende Leerzeichen bedeuten keine Änderung des Wertes, und es tritt kein Gruppenwechsel auf.

Im Falle eines binären Gruppenwechselkontrollfeldes sind nachfolgende binäre Nullen nicht bedeutend für den Vergleich, d.h. nachfolgende binäre Nullen bedeuten keine Änderung des Wertes, und es findet kein Gruppenwechsel statt.

Parameter-Übertragung mit dynamischen Variablen

Dynamische Variablen können als Parameter an aufgerufene Programmobjekte (CALLNAT, PERFORM) übergeben werden. Aufruf über eine Referenz (Call-by-Reference) ist möglich, weil der Wertespeicher einer dynamischen Variable zusammenhängend ist. Aufruf über Wert (Call-by-Value) führt zu einer Zuweisung der Variablen-Definition des Aufrufenden als Source-Operand und der Parameter-Definition als Ziel-Operand. Bei Aufruf über Wert (Ergebnis) (Call-by-Value (Result)) ist es umgekehrt.

Bei Aufruf über eine Referenz (Call-by-Reference) müssen beide Definitionen dynamisch (DYNAMIC) sein. Wenn nur eine von ihnen dynamisch ist, tritt ein Laufzeitfehler auf. Im Falle von Call-by-Value (Result), d.h. Aufruf über Wert (Ergebnis) sind alle Kombinationen möglich. Die folgende Tabelle veranschaulicht die gültigen Kombinationen:

Call-By-Reference

Caller	**Parameter**	
	Statisch	**Dynamisch**
Statisch	Ja	Nein
Dynamisch	Nein	Ja

Die Formate der dynamischen Variablen A oder B müssen miteinander im Einklang sein.

Call-by-Value (Result)

Caller	**Parameter**	
	Statisch	**Dynamisch**
Statisch	Ja	Ja
Dynamisch	Ja	Ja

Anmerkung:
Im Falle von statischen/dynamischen oder dynamischen/statischen Definitionen können gemäß der Datenübertragungsregeln der betreffenden Zuweisungen Werte abgeschnitten werden.

Beispiel 1:

```
** Example 'DYNAMX02': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX03' USING #MYTEXT
```

```
*
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
*
END
```

Subprogramm DYNAMX03:

```
** Example 'DYNAMX03': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC  BY VALUE RESULT
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                 /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                   /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                  /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                 /* *LENGTH(#MYPARM) = 8
*
/* content of #MYPARM is moved back to #MYTEXT
/* used length of #MYTEXT = 8
*
END
```

Beispiel 2:

```
** Example 'DYNAMX04': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA LOCAL
1 #MYTEXT (A) DYNAMIC
END-DEFINE
*
#MYTEXT := '123456'      /* extended to 6 bytes, *LENGTH(#MYTEXT) = 6
*
CALLNAT 'DYNAMX05' USING #MYTEXT
*
WRITE *LENGTH(#MYTEXT)          /* *LENGTH(#MYTEXT) = 8
                                /* at least 10 bytes are
                                /* allocated (extended in DYNAMX05)
*
END
```

Subprogramm DYNAMX05:

```
** Example 'DYNAMX05': Dynamic variables (as parameters)
************************************************************************
DEFINE DATA PARAMETER
1 #MYPARM (A) DYNAMIC
END-DEFINE
*
WRITE *LENGTH(#MYPARM)                 /* *LENGTH(#MYPARM) = 6
#MYPARM := '1234567'                   /* *LENGTH(#MYPARM) = 7
#MYPARM := '12345678'                  /* *LENGTH(#MYPARM) = 8
EXPAND DYNAMIC VARIABLE #MYPARM TO 10   /* 10 bytes are allocated
*
WRITE *LENGTH(#MYPARM)                 /* *LENGTH(#MYPARM) = 8
*
END
```

3GL-Progamm aufrufen

Dynamische und große Variablen können mit dem CALL-Statement sinnvoll benutzt werden, wenn die Option INTERFACE4 verwendet wird. Der Einsatz dieser Option führt zu einer Schnittstelle zum 3GL-Programm mit einer unterschiedlichen Parameter-Struktur.

Dazu sind einige geringfügige Änderungen an dem 3GL-Programm erforderlich, jedoch bringt es folgende signifikanten Vorteile im Vergleich zu der früheren FINFO-Struktur:

- Keine Einschränkung bei der Anzahl der übergebenen Parameter (früher begrenzt auf 40)
- Keine Einschränkung bei der Datengröße eines Parameters (früher auf 64 KB pro Parameter begrenzt)
- Die Parameterinformationen können vollständig an das 3GL-Programm übergeben werden (einschließlich Array-Informationen). Exportierte Funktionen sind verfügbar, die einen sicheren Zugriff auf die Parameterdaten ermöglichen (früher musste darauf geachtet werden, nicht den Speicher in Natural zu überschreiben).

Weitere Informationen zur FINFO-Struktur siehe CALL INTERFACE4-Statement.

Bevor Sie ein 3GL-Programm mit dynamischen Parametern aufrufen, ist es wichtig sicherzustellen, dass die erforderliche Puffergröße zugewiesen wird. Dies kann explizit über das EXPAND-Statement erfolgen.

Wenn ein initialisierter Puffer erforderlich ist, kann die dynamische Variable mittels des MOVE ALL UNTIL-Statements auf den Ausgangswert und auf die erforderliche Größe gesetzt werden. Natural stellt eine Reihe von Funktionen zur Verfügung, die es dem 3GL-Programm ermöglichen, Informationen über dynamische Parameter zu erhalten und die Länge zu ändern, wenn Parameterdaten zurückgeschrieben werden.

Beispiel:

```
MOVE ALL ' ' TO  #MYDYNTEXT1 UNTIL 10000
  /* a buffer of length 10000 is allocated
  /* #MYDYNTEXT1 is initialized with blanks
  /* and *LENGTH(#MYDYNTEXT1) = 10000
CALL INTERFACE4  'MYPROG'  USING  #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
  /* *LENGTH(#MYDYNTEXT1) may have changed in the 3GL program
```

Eine ausführlichere Beschreibung finden Sie beim CALL-Statement in der *Statements*-Dokumentation.

Arbeitsdateizugriff bei großen und dynamischen Variablen

Folgende Themen werden behandelt:

- PORTABLE und UNFORMATTED
- ASCII, ASCII-COMPRESSED und SAG
- Besondere Bedingungen für TRANSFER und ENTIRE CONNECTION

PORTABLE und UNFORMATTED

Große und dynamische Variablen können mit den beiden Arbeitsdateitypen PORTABLE and UNFORMATTED in Arbeitsdateien geschrieben werden oder aus Arbeitsdateien gelesen werden. Bei diesen Arbeitsdateitypen gibt es keine Größenbeschränkung für dynamische Variablen. Große Variablen dürfen die maximale Feld-/Datensatzlänge von 32766 Bytes jedoch nicht überschreiten.

Beim Arbeitsdateityp PORTABLE wird die Feldinformation in der Arbeitsdatei gespeichert. Bei dynamischen Variablen wird die Größe während des `READ` geändert, wenn die Feldgröße im Datensatz von der aktuellen Größe abweicht.

Der Arbeitsdateityp UNFORMATTED kann zum Beispiel dazu genutzt werden, ein Video aus einer Datenbank auszulesen und es in einer Datei abzulegen, die mit anderen Dienstprogrammen direkt abspielbar ist. Mit dem `WRITE WORK`-Statement werden die Felder mit ihrer Byte-Länge in die Datei geschrieben. Alle Datentypen (`DYNAMIC` oder auch nicht) werden gleich behandelt. Es werden keine strukturellen Informationen eingefügt. Da Natural einen Puffermechanismus benutzt, werden die Daten erst nach einem `CLOSE WORK` komplett geschrieben. Dies ist dann besonders wichtig, wenn die Datei mit einem anderen Dienstprogramm verarbeitet werden soll während Natural aktiv ist.

Mit dem `READ WORK`-Statement werden Felder mit einer festen Länge mit ihrer gesamten Länge gelesen. Wenn das Ende der Datei erreicht wird, wird der Rest des aktuellen Feldes mit Leerzeichen aufgefüllt. Die nachfolgenden Felder bleiben unverändert. Im Falle von `DYNAMIC`-Datentypen wird der komplette Rest der Datei gelesen, außer wenn sie 1073741824 Bytes überschreitet. Wenn das Ende der Datei erreicht wird, bleiben die übrigen Felder (Variablen) unverändert (normales Natural-Verhalten).

ASCII, ASCII-COMPRESSED und SAG

Die Arbeitsdateitypen ASCII, ASCII-COMPRESSED und SAG (binär) können keine dynamischen Variablen verarbeiten; in diesem Fall tritt ein Fehler auf. Große Variablen stellen kein Problem für diese Arbeitsdateitypen dar, außer wenn die maximale Feld-/Datensatzlänge von 32766 Bytes überschritten wird.

Besondere Bedingungen für TRANSFER und ENTIRE CONNECTION

In Verbindung mit dem Statement `READ WORK FILE` kann der Arbeitsdateityp TRANSFER dynamische Variablen verarbeiten. Es gibt keine Größenbeschränkung für dynamische Variablen. Der Arbeitsdateityp ENTIRE CONNECTION kann keine dynamischen Variablen verarbeiten. Beide Arbeitsdateitypen können jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 1073741824 Bytes verarbeiten.

In Verbindung mit dem Statement `WRITE WORK FILE` kann der Arbeitsdateityp TRANSFER dynamische Variablen mit einer maximalen Feld-/Datensatzlänge von 32766 Bytes verarbeiten. Der Arbeitsdateityp ENTIRE CONNECTION kann keine dynamischen Variablen verarbeiten. Beide Arbeitsdateitypen können jedoch große Variablen mit einer maximalen Feld-/Datensatzlänge von 1073741824 Bytes verarbeiten.

DDM-Generierung und Editieren von Spalten mit variabler Länge

Abhängig von den Datentypen wird entweder das entsprechende Datenbankformat A oder B generiert. Für den Datenbank-Datentyp VARCHAR wird die Natural-Länge der Spalte auf die maximale Länge des Datentyps gesetzt, und zwar so wie sie im DBMS definiert ist. Wenn ein Datentyp sehr groß ist, wird an der Position des Längenfeldes das Schlüsselwort DYNAMIC generiert.

Für alle Spalten mit variabler Länge wird ein LINDICATOR-Feld L@<*column-name*> generiert. Für den Datenbank-Datentyp VARCHAR wird ein LINDICATOR-Feld mit Format/Länge I2 generiert. Bei großen Datentypen (siehe die Liste unten) wird Format/Länge I4 benutzt.

Bei einem Datenbankzugriff bietet LINDICATOR die Chance, die Länge des zu lesenden Feldes zu bekommen oder die Länge des zu schreibenden Feldes unabhängig von der definierten Pufferlänge zu setzen (oder unabhängig von *LENGTH). Nach einer Abfragefunktion wird *LENGTH normalerweise auf den Wert des entsprechenden Längenindikators gesetzt.

Beispiel-DDM:

```
T  L  Name                     F   Leng        S   D   Remark
   :
   1  L@PICTURE1               I   4                                  /* length indicator
   1  PICTURE1                 B   DYNAMIC             IMAGE
   1  N@PICTURE1               I   2                                  /* NULL indicator
   1  L@TEXT1                  I   4                                  /* length indicator
   1  TEXT1                    A   DYNAMIC             TEXT
   1  N@TEXT1                  I   2                                  /* NULL indicator
   1  L@DESCRIPTION            I   2                                  /* length indicator
   1  DESCRIPTION              A   1000                VARCHAR(1000)
   :
   :
~~~~~~~~~~~~~~~~~~~~~~Extended Attributes~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/* concerning PICTURE1
 Header               :    ---
 Edit Mask            :    ---
 Remarks              :   IMAGE
```

Die generierten Formate sind von unterschiedlicher Länge. Ein Natural-Programmierer hat die Möglichkeit, die Definition von DYNAMIC in eine feste Längendefinition (erweiterte Feldbearbeitung) zu ändern und kann zum Beispiel die entsprechende DDM-Felddefinition für VARCHAR-Datentypen in ein Feld mit multiplen Werten abändern (alte Generierung).

Beispiel:

```
T  L  Name                     F   Leng          S   D   Remark
   :
   1  L@PICTURE1               I   4                                  /* length indicator
   1  PICTURE1                 B   1000000000          IMAGE
   1  N@PICTURE1               I   2                                  /* NULL indicator
   1  L@TEXT1                  I   4                                  /* length indicator
   1  TEXT1                    A   5000                TEXT
   1  N@TEXT1                  I   2                                  /* NULL indicator
   1  L@DESCRIPTION            I   2                                  /* length indicator
M  1  DESCRIPTION              A   100                 VARCHAR(1000)
     :
     :
~~~~~~~~~~~~~~~~~~~~~~Extended Attributes~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~/* concerning PICTURE1
 Header               :    ---
 Edit Mask            :    ---
 Remarks              :   IMAGE
```

Zugriff auf große Datenbankobjekte

Für den Zugriff auf eine Datenbank mit großen Objekten (CLOBs oder BLOBs) ist ein DDM mit entsprechenden großen alphanumerischen Feldern, Unicode-Feldern oder binären Feldern erforderlich. Wenn eine feste Länge definiert ist und in der Datenbank ein großes Objekt nicht in dieses Feld passt, wird das große Objekt abgeschnitten. Wenn der Programmierer die eindeutige Länge des Datenbankobjektes nicht kennt, dann ist es sinnvoll mit dynamischen Feldern zu arbeiten. So viele Neuzuteilungen wie nötig werden durchgeführt, um das Objekt zu halten. Es wird nichts abgeschnitten.

Beispielprogramm:

```
DEFINE DATA LOCAL

1 person VIEW OF xyz-person
  2 last_name
  2 first_name_1
  2 L@PICTURE1                  /* I4 length indicator for PICTURE1
  2 PICTURE1                    /* defined as dynamic in the DDM
  2 TEXT1                       /* defined as non-dynamic in the DDM

END-DEFINE

SELECT * INTO VIEW person FROM xyz-person                 /* PICTURE1 will be read completely
                            WHERE last_name = 'SMITH'     /* TEXT1 will be truncated to fixed length 5000

   WRITE  'length of PICTURE1: '  L@PICTURE1              /* the L-INDICATOR will contain the length
                                                          /* of PICTURE1 (= *LENGTH(PICTURE1)
   /* do something with PICTURE1 and TEXT1

   L@PICTURE1  :=  100000
   INSERT INTO xyz-person  (*) VALUES (VIEW person)       /* only the first 100000 Bytes of PICTURE1
                                                          /* are inserted
END-SELECT
```

Wenn im View eine Format-/Längendefinition weggelassen wird, wird sie aus dem DDM genommen. Im Reporting Mode ist es jetzt möglich, eine beliebige Länge zu definieren, wenn das entsprechende DDM-Feld als `DYNAMIC` definiert ist. Das dynamische Feld wird in einem Feld mit einer festen Pufferlänge abgebildet. Anders herum ist es nicht möglich.

DDM-Format-/Längendefinition	VIEW-Format-/Längendefinition	
(An)	-	gültig
	(An)	gültig
	(Am)	nur im Reporting Mode gültig
	(A) DYNAMIC	ungültig
(A) DYNAMIC	-	gültig
	(A) DYNAMIC	gültig
	(An)	nur im Reporting Mode gültig
	(Am / i : j)	nur im Reporting Mode gültig

(entsprechend für Format B-Variablen)

Parameter mit LINDICATOR-Klausel in SQL-Statements

Wenn das LINDICATOR-Feld als ein I2-Feld definiert ist, wird der SQL-Datentyp VARCHAR zum Senden oder Empfangen der entsprechenden Spalte benutzt. Wenn die LINDICATOR-Host-Variable als I4 angegeben ist, wird ein Datentyp für große Objekte (CLOB/BLOB) benutzt.

Wenn das Feld als DYNAMIC definiert ist, wird die Spalte bis zu ihrer tatsächlichen Länge in einer internen Schleife gelesen. Das LINDICATOR-Feld und *LENGTH werden auf diese Länge gesetzt. Bei einem Feld mit einer festen Länge wird die Spalte bis zur definierten Länge gelesen. In beiden Fällen wird das Feld bis zu dem im LINDICATOR-Feld definierten Wert geschrieben.

Performance-Aspekte bei dynamischen Variablen

Wenn eine dynamische Variable in kleinen Beträgen mehrmals (z.B. byteweise) erweitert werden soll, benutzen Sie das EXPAND-Statement vor den Schleifen-Iterationen, wenn die obere Grenze des erforderlichen Speichers (ungefähr) bekannt ist. Dadurch vermeiden Sie zusätzlichen Verarbeitungsmehraufwand zum Anpassen des erforderlichen Speicherplatzes.

Benutzen Sie das REDUCE- oder RESIZE-Statement, wenn die dynamische Variable nicht mehr erforderlich ist, insbesondere für Variablen mit einem hohen Wert von *LENGTH. Dadurch wird es Natural ermöglicht, den Speicherplatz wieder zu benutzen oder freizugeben. Somit kann die Gesamtleistung verbessert werden.

Die Größe des einer dynamischen Variable zugewiesenen Hauptspeichers kann mittels des REDUCE DYNAMIC VARIABLE-Statements auf eine angegebene Größe reduziert werden. Um eine Variable auf eine angegebene Größe (neu) zuzuweisen, kann das EXPAND-Statement benutzt werden.

Wenn die Variable initialisiert werden soll, benutzen Sie das MOVE ALL UNTIL-Statement.

Beispiel:

```
** Example 'DYNAMX06': Dynamic variables (allocated memory)
************************************************************************
DEFINE DATA LOCAL
1 #MYDYNTEXT1  (A)  DYNAMIC
1 #LEN         (I4)
END-DEFINE
*
#MYDYNTEXT1 := 'a'     /* used length is 1, value is 'a'
                       /* allocated size is still 1
WRITE *LENGTH(#MYDYNTEXT1)
*
EXPAND DYNAMIC VARIABLE  #MYDYNTEXT1 TO 100
                       /* used length is still 1, value is 'a'
                       /* allocated size is 100
*
CALLNAT 'DYNAMX05' USING #MYDYNTEXT1
WRITE *LENGTH(#MYDYNTEXT1)
                       /* used length and allocated size
                       /* may have changed in the subprogram
*
#LEN := *LENGTH(#MYDYNTEXT1)
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  #LEN
                       /* if allocated size is greater than used length,
                       /* the unused memory is released
*
```

```
REDUCE DYNAMIC VARIABLE  #MYDYNTEXT1  TO  0
WRITE *LENGTH(#MYDYNTEXT1)
                      /* free allocated memory for dynamic variable
END
```

Regeln:

- Benutzen Sie dynamische Operanden überall dort, wo es sinnvoll ist.
- Benutzen Sie EXPAND, wenn die obere Grenze der Hauptspeicher-Benutzung bekannt ist.
- Benutzen Sie REDUCE, wenn der dynamische Operand nicht mehr gebraucht wird.

Ausgabe von dynamische Variablen

Dynamische Variablen können innerhalb von Ausgabe-Statements wie folgt benutzt werden:

Statement	Anmerkungen
DISPLAY	Mit diesen Statements müssen Sie das Format der Ausgabe oder Eingabe von dynamischen Variablen setzen, indem Sie die Session-Parameter AL (Alphanumerische Länge für die Ausgabe) oder EM (Editiermaske) benutzen.
WRITE	
INPUT	
REINPUT	--
PRINT	Da die Ausgabe des PRINT-Statements unformatiert ist, muss die Ausgabe der dynamischen Variablen im PRINT-Statement nicht mittels der Parameter AL und EM gesetzt werden. Mit anderen Worten: diese Parameter können weggelassen werden.

Dynamische X-Arrays

Ein dynamisches X-Array kann zugewiesen werden, indem man zuerst die Anzahl der Ausprägungen angibt und dann die Länge der vorher zugewiesenen Array-Ausprägungen erweitert.

Beispiel:

```
DEFINE DATA LOCAL
 1 #X-ARRAY(A/1:*)  DYNAMIC
END-DEFINE
*
EXPAND  ARRAY   #X-ARRAY TO (1:10)  /* Current boundaries (1:10)
#X-ARRAY(*) := 'ABC'
EXPAND  ARRAY   #X-ARRAY TO (1:20)  /* Current boundaries (1:20)
#X-ARRAY(11:20) := 'DEF'
```