Zusatzangebot

ABAP für InfoSets

zum Buch

»Praxishandbuch Reporting im SAP®-Finanzwesen«
von Heinz Forsthuber, Abdarahman Fardas, Karin Bädekerl

Galileo Press

SAP PRESS 2011
ISBN 978-3-8362-1680-7

Bonn • Boston

In diesem Zusatzkapitel finden Sie eine kurze Einführung in die Programmiersprache ABAP speziell für den Einsatz in InfoSets, die im Buch in Kapitel 9 behandelt werden.

ABAP für InfoSets

Das SAP-System R/3 wurde zum überwiegenden Teil in der von SAP entwickelten Programmiersprache ABAP geschrieben, die ihren Ursprung in der Programmiersprache COBOL hat. Anfangs wurden vor allem Sprachelemente zur Erstellung von Drucklisten integriert. Davon zeugt die ursprüngliche Bezeichnung ***A**llgemeiner **B**erichts**a**ufbereitungs**p**rozessor*. Inzwischen sind unterschiedliche Sprachelemente eingeflossen.

Quelltexteditor

Der Quelltexteditor (siehe Abbildung 1) ist selbst eine mit ABAP erstellte Dialoganwendung. Seine Oberfläche ist ebenfalls ein Dynpro und deshalb zeilenorientiert. Jedoch wurden inzwischen viele Funktionen moderner, seitenorientierter Editoren aufgenommen.

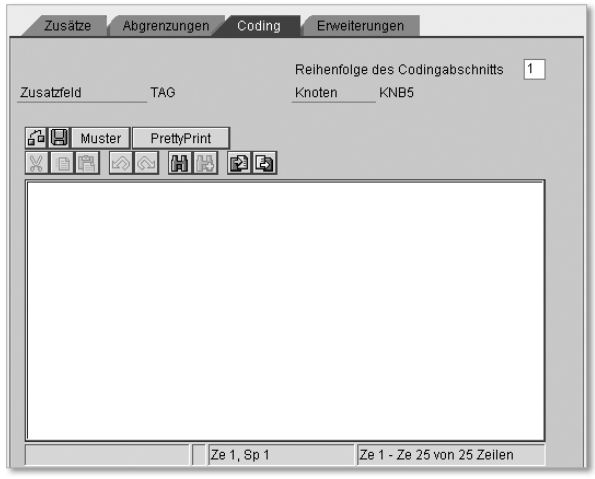

Abbildung 1 Quelltexteditor

Programmaufbau

Jedes Programm besteht aus einem Deklarationsteil und einem Operationsteil. Im Deklarationsteil werden alle Datenstrukturen, Tabellen und Parameter vereinbart. Der Operationsteil enthält die auszuführenden Programmanweisungen. ABAP-Befehle beginnen in der Regel mit einem Schlüsselwort und werden mit einem Punkt ».« abgeschlossen. Wörter werden durch mindestens ein Leerzeichen voneinander getrennt. Die Anweisungen können formatfrei eingegeben werden. Mehrere Anweisungen pro Zeile und einzelne Anweisungen über mehrere Zeilen sind erlaubt.

Es ist möglich, mehrere aufeinanderfolgende Anweisungen mit dem gleichen Schlüsselwort zu Kettensätzen zusammenzufassen. Nach dem Schlüsselwort muss dann ein Doppelpunkt »:« eingegeben werden. Die verketteten Anweisungsteile werden durch Kommata »,« getrennt. Vor und hinter den Trennzeichen können Leerzeichen stehen.

Die Anweisungen

```
WRITE 'Name: '.
WRITE NAME.
```

sind identisch mit dem Kettensatz

```
WRITE: 'Name: ', NAME.
```

Kommentare können auf zwei unterschiedliche Arten im Programm hinterlegt werden. Ein Stern '*' in Spalte 1 kennzeichnet eine gesamte Zeile als Kommentar. Nach einem Anführungszeichen '«' innerhalb einer Zeile wird der Text bis zum Zeilenende als Kommentar interpretiert.

```
* Das ist eine Kommentarzeile
DATA NAME. "Vereinbarung der Variablen NAME
```

Definition von Datenfeldern, Typen und Konstanten

Datenfelder (Variablen) müssen vor ihrer erstmaligen Benutzung im Deklarationsteil des Programms definiert werden. Dazu ist neben dem Schlüsselwort `DATA` die Angabe eines Datentyps und der Feldlänge oder einer Referenz erforderlich.

Feldnamen können bis zu 30 Zeichen lang sein und Sonderzeichen sowie Ziffern enthalten. Das erste Zeichen sollte ein Buchstabe sein. Die Zeichen ‚(' ‚)' ‚+' ‚.' ‚,' und ‚:' sind nicht erlaubt. Der Bindestrich ‚-' sollte nicht verwendet werden, da er als Trennstrich für die Bestandteile zusammengesetzter

Feldnamen verwendet wird. Empfehlenswert ist, zusammengesetzte Feldnamen mit einem Unterstrich ‚_' zu unterteilen und darüber hinaus auf Sonderzeichen im Feldnamen zu verzichten. Einige Namen sind in ABAP vordefiniert. Dazu gehören die Systemvariablen, die generell mit `SY` oder `SYST` beginnen. Tabelle 1 zeigt einige Beispiele.

Systemvariable	Bedeutung
SY-UNAME	Name des Benutzers, der das Programm startet
SY-DATUM	Aktuelles Datum in der Form YYYYMMTT
SY-UZEIT	Aktuelle Uhrzeit in der Form HHMMSS
SY-SUBRC	Return-Code

Tabelle 1 Systemvariablen

Die `DATA`-Anweisung verfügt über eine Reihe von Zusätzen, von denen nur einige dargestellt werden können. Eine einfache Datendeklaration hat folgenden Aufbau:

```
DATA <Variablenname>[(<Länge>)] {TYPE <Typ>| LIKE <Var>}[VALUE
<Standardwert>].
```

Dabei gilt:

- `<Variablenname>` ist eine Zeichenkette von bis zu 30 Buchstaben, die die Variable symbolisiert.
- `<Länge>` ist die Länge des Feldes, wenn eine andere als die Standardlänge gewünscht wird. Dies ist insbesondere für Zeichenketten interessant, da deren Standardlänge nur ein Zeichen beträgt. Eine Längenangabe ist nur für die Typen C, P und N zulässig.
- `<Typ>` ist einer der in Tabelle 2 aufgelisteten Type.

Typ	Bedeutung	Standard-länge	Initialwert
C	Text (Zeichenkette)	1	Leerzeichen
N	Numerischer Text	1	‚000…0'
D	Aktuelles Datum in der Form YYYYMMTT	8	‚00000000'
T	Aktuelle Uhrzeit in der Form HHMMSS	6	‚000000'
I	Ganze Zahl (Integer)	4	0
P	Gepackte Zahl	8	0

Tabelle 2 Datentypen

Der Typ gibt an, wie der Feldinhalt bei Berechnungen und Ausgaben interpretiert wird. Er legt darüber hinaus die Länge, einen Initialwert und den erlaubten Zeichenvorrat fest. Während der Typ C alle Zeichen enthalten kann, sind für den Typ N nur Ziffern als Eingabe möglich. Hier einige Beispiele:

```
* Definition eines Feldes vom Typ Datum.
DATA DATUM1 TYPE D.
* Definition einer Zeichenkette der Länge 10.
DATA NAME(10) TYPE C.
* Definition einer Zeichenkette der Länge 1.
DATA ZEICHEN.
* Definition von drei ganzen Zahlen.
DATA: I1,I2,I3 TYPE I.
* Definition einer gepackten Zahl mit acht Vor- und zwei
Nachkommastellen.
DATA VAL(10) TYPE P DECIMALS 2.
```

Soll eine Variable den gleichen Typ wie eine bereits deklarierte Variable besitzen, kann dies über den Zusatz `LIKE` definiert werden.

```
* Definition einer Zeichenkette der Länge 10.
DATA NAME(10) TYPE C.
* Definition einer Zeichenkette der Länge 10.
DATA NAME_2 LIKE NAME.
* Definition einer Variablen vom gleichen Typ wie SY-DATUM.
DATA DATUM LIKE SY-DATUM.
* Definition einer Variablen vom Typ wie das Tabellenfeld LFA1-NAME1.
DATA NAME LIKE LFA1-NAME1.
```

Außerdem besteht die Möglichkeit, direkt Konstanten mit dem Schlüsselwort `CONSTANTS` zu vereinbaren:

```
CONSTANTS DATUM TYPE D VALUE '20090416'.
```

Bei der Zuweisung von Initialwerten und Konstantenvereinbarungen müssen Sie für die Typen C und N in Apostrophe eingeschlossene Textliterale (etwa 'Name') angeben. Für die Typen P und I werden Zahlenliterale ohne Apostroph (etwa 2340) geschrieben. Für alle anderen Typen geben Sie Textliterale an, die vom SAP-System automatisch konvertiert werden.

Ausgabe und Formatierung

Mit Ausgabeanweisungen können die Inhalte von Datenfeldern auf ein Ausgabemedium übertragen werden. Standardmäßig erfolgt eine Ausgabe als Liste auf dem Bildschirm. Es ist aber auch möglich, die Ausgabe auf einen

Drucker umzuleiten. Mit der WRITE-Anweisung werden Feldinhalte ausgegeben. Dabei gilt folgende Syntax:

```
WRITE [AT] [/][<Spalte>][(<Ausgabelänge>)] <Wert> [<Format>].
```

Ein Schrägstrich '/' heißt, die Ausgabe startet in einer neuen Zeile, sofern die aktuelle nicht leer ist. Ist eine Spalte angegeben, fängt dort der Schreibvorgang an. Mit <Ausgabelänge> kann die Länge des Strings begrenzt werden. Alle drei Angaben müssen, soweit vorhanden, ohne trennendes Leerzeichen angegeben werden. So schreibt die Anweisung `WRITE /5(20) NAME.` den Namen in eine neue Zeile, in die 5. Spalte und mit einer maximalen Ausgabelänge von 20 Zeichen.

<Format> kann sehr vielschichtige Zusätze umfassen, die wichtigsten zeigt folgende Übersicht:

Zusatz	Bedeutung
DD/MM/YYYY	Datum im benutzerspezifischen Format
USING EDIT MASK <Maske>	Formatierung gemäß Maske, wobei '_' für ein Zeichen des einfachen Ausgabestrings steht: z. B. ZEIT USING EDIT MASK '__:__:__'
DECIMALS <Anz>	Limitieren der Dezimalstellen einer nichtganzen Zahl
NO-GAP	keinen Abstand (= Leerzeichen) hinter dem Feld ausgeben
LEFT-JUSTIFIED	linksbündig ausgeben
CENTERED	zentriert ausgeben (entsprechend der definierten Länge)
RIGHT-JUSTIFIED	rechtsbündig ausgeben

Tabelle 3 Zusatzformatierungen

Für die benutzerspezifische Ausgabe des Datums müssen Sie dessen Form in der Transaktion zur Festlegung der Benutzerfestwerte SU50 auswählen. Zur Formatierung der Ausgabe steht eine Reihe von Befehlen zur Verfügung. Im Folgenden sind die wichtigsten Ausgabeoperationen und -klauseln mit Beispielen aufgelistet:

```
* Ausgabe der Variablen NAME.
WRITE NAME.
* Ausgabe mehrerer Variablen in einer Zeile.
WRITE: NAME, NAME_ALT, NAME_ZUSATZ.
Ausgabe eines Textes in einer neuen Zeile.
WRITE / 'Name: ', NAME.
* Ausgabe von maximal 20 Zeichen der Variablen NAME in eine neue * Zeile
ab der Spalte 10.
```

```
WRITE /10(20) NAME.
* Ausgabe von NAME_ALT ab der gleichen Spalte wie NAME.
WRITE / NAME_ALT UNDER NAME.
* Benutzerspezifische Datumsausgabe.
WRITE DATUM TT/MM/JJJJ.
* Ausgabe einer Leerzeile.
SKIP.
* Ausgabe von fünf Leerzeilen.
SKIP 5.
* Ausgabe einer Linie.
ULINE.
* Erzwingt den Anfang einer neuen Seite.
NEW-PAGE.
```

Außerdem gibt es die Möglichkeit, Icons und Symbole mit `WRITE` anzuzeigen. Genaueres über die Zusätze `AS ICON` und `AS SMBOL` erfahren Sie in der Online-Hilfe zum Befehl `WRITE`.

Wertzuweisung und Operationen

Sollen Werte an Variablen (Datenfelder) übergeben werden, müssen diese zugewiesen werden. Bei Verwendung des Zuweisungsoperators '=' ist keine Steuerungsanweisung notwendig.

```
* Übertragung der Variablen NAME_ALT auf die Variable NAME.
NAME = NAME_ALT.
```

In vielen ABAP-Programmen wird das Schlüsselwort `MOVE` verwendet. Es hat dieselbe Bedeutung.

```
* Übertragung der Variablen NAME_ALT auf die Variable NAME.
MOVE NAME_ALT TO NAME.
```

Wertzuweisungen können Operationen enthalten. Als Standardoperatoren stehen +, –, / (Division), * (Multiplikation), ** (Potenzierung), `DIV` (ganzzahlige Division) und `MOD` (Rest der ganzzahligen Division) zur Verfügung. Weitere mathematische Funktionen können Sie der Online-Hilfe entnehmen. Klammersetzung ist erlaubt. Vor und nach jedem Operator und jeder Klammer muss ein Leerzeichen stehen.

```
* V1 erhält den Wert 4.
V1 = 12 / ( 7 - 4 ).
* Für V2 ergibt sich der Wert 3.
V2 = 7 MOD 4.
* Die Operation ergibt den Wert 7.
V3 = V1 + V2.
```

Die hier angegebene Schreibweise ist die Kurzform der Wertzuweisung mit dem optionalen Schlüsselwort COMPUTE. Die vollständige Syntax lautet:

```
COMPUTE <Feld> = <Ausdruck>.
```

Alternativ können die Anweisungen ADD TO, SUBTRACT FROM, MULTIPLY BY und DIVIDE BY verwendet werden. Zu beachten ist, dass dabei kein Ausdruck verwendet werden darf und die Ergebnisvariable am Ende der Anweisung steht:

```
ADD >Wert> TO <Feld>.
```

Das Operationszeichen '+' wird auch als Offsetangabe für Zeichenketten verwendet.

```
* Die Zeichen 3 bis 7 von C werden an C1 übergeben.
C1 = C+2(5).
```

Da nur die fehlenden Leerzeichen eine Addition von einer solchen Zeichenkettenoperation unterscheiden, führt »V1 = C+3.« anstelle von »V1 = C + 3.« nicht zu einem Syntaxfehler, aber zu einem falschen Ergebnis. Die Anweisung CLEAR setzt Variablen auf ihren durch den Typ festgelegten Initialwert zurück.

```
* Die Variable V1 wird auf 0, die Variable DATUM auf '00000000'
zurückgesetzt.
CLEAR: V1, DATUM.
```

Datenbankabfragen

Am häufigsten werden ABAP-Programme zur Verarbeitung von im SAP-System gespeicherten Datenbeständen genutzt. Alle Daten sind in Tabellen der angeschlossenen relationalen Datenbank gespeichert.

Deklarieren von Datenbanktabellen

Die Namen der Tabellen, die im Programm verwendet werden, müssen im Deklarationsteil in einer TABLES-Anweisung aufgelistet werden.

```
TABLES: LFA1, LFB1, LFBK.
```

Diese Deklaration ist mit einer Definition eines Datenaustauschbereichs zwischen Programm und Tabelle verbunden. Die einzelnen Felder werden über <Tabellenname>-<Feldname> angesprochen. LFA1-NAME1 bezeichnet das Feld NAME1 in der Tabelle LFA1.

Lesen von Werten aus einer Datenbanktabelle

Die SELECT-Anweisung dient dem Auslesen von Daten aus der Datenbank. Sie kann im Programm auf jede Datenbanktabelle angewendet werden, die mit TABLES deklariert wurde. Es gibt zwei grundlegende Ausprägungen der Anweisung. Die gebräuchlichere hat die folgende Form:

```
SELECT * FROM <Tabelle> [WHERE <Bedingungen>]
[ORDER BY <Felder>]
[UP TO <n> ROWS].
<Verarbeitung des Datensatzes>
ENDSELECT.
```

Die Anweisung arbeitet in einer Schleife alle Datensätze ab, die den Bedingungen der WHERE-Klausel genügen. Innerhalb der Schleife, die durch das Schlüsselwort ENDSELECT beendet wird, kann der jeweils aktuelle Datensatz verarbeitet werden. Die einzelnen Felder werden dabei über <Tabelle>-<Feldname> angesprochen.

Für die Zusätze der SELECT-Anweisung gilt Folgendes:

Die WHERE-Klausel erlaubt es Ihnen, den Datenbestand durch Bedingungen einzuschränken. Das bietet einen erheblichen Performancevorteil gegenüber einer eigenen Einschränkung innerhalb der SELECT-Schleife. Innerhalb einer WHERE-Klausel müssen die Felder der aktuellen Tabelle ohne den Zusatz <Tabelle>- angegeben werden.

Mit der ORDER BY-Klausel können die selektierten Daten nach einem oder mehreren Feldern sortiert werden. Bei Angabe mehrerer Felder werden die Daten zunächst nach dem ersten Feld sortiert, innerhalb der Gruppen mit demselben Feldinhalt nach dem zweiten etc. Die Syntax für die Angabe eines Feldes lautet jeweils

```
<Feld> [ASCENDING|DESCENDING].
```

Dabei steht der Zusatz ASCENDING für aufsteigende, DESCENDING für absteigende Sortierung. Ist keine Sortierung angegeben, wird aufsteigend sortiert. Der Zusatz PRIMARY KEY sortiert nach den Primärschlüsselfeldern der Tabelle. Mit dem Zusatz UP TO <n> ROWS kann die Bearbeitung auf die ersten n Datensätze beschränkt werden. Dazu zwei Beispiele:
Im ersten Beispiel

```
SELECT * FROM LFA1 WHERE ORT01 = 'Berlin' (...).
```

liest das Programm aus der Tabelle LFA1 alle Angaben über Lieferanten, die in Berlin ansässig sind.
Im zweiten Beispiel

```
SELECT * FROM UMTAB1 ORDER BY UMSATZ DESCENDING UP TO 10 ROWS.
```

liest das Programm aus der fiktiven Tabelle UMTAB1 die zehn umsatzstärksten Kunden, absteigend nach UMSATZ sortiert. Es gibt auch die Möglichkeit, SELECT-Schleifen zu schachteln. Dies ist insbesondere für Tabellenhierarchien mit 1:n-Beziehungen wichtig. Betrachten wir hierzu die beiden Tabellen KUNDEN (mit den Feldern KUNDENNR und ORT) sowie UMSATZ (mit den Feldern KUNDENNR, GJAHR und UMSATZ), könnte eine geschachtelte SELECT-Anweisung etwa so aussehen:

```
SELECT * FROM KUNDEN ORDER BY NAME.
WRITE: / KUNDEN-NAME, KUNDEN-ORT.
SELECT * FROM UMSATZ WHERE KUNDENNR = KUNDEN-KUNDENNR
ORDER BY GJAHR DESCENDING.
WRITE: / UMSATZ-GJAHR, UMSATZ-UMSATZ.
ENDSELECT.
ENDSELECT.
```

Die zweite Ausprägung von SELECT ist die SELECT SINGLE-Anweisung. Diese liest, wenn die Bedingung erfüllt ist, genau einen Datensatz. Sie hat die folgende Form:

```
SELECT SINGLE * FROM <Tabelle> WHERE <Bedingung>.
```

Ein ENDSELECT darf nicht geschrieben werden. SELECT SINGLE liest genau einen Datensatz über seinen Schlüssel aus. Dazu müssen alle Felder, die zum Schlüssel gehören, über einen Vergleich als Bedingung angegeben werden. Betrachten wir hierzu die zeitabhängige Tabelle KUNDE (mit den Feldern KUNDENNR, NAME, KONTAKT, DATAB, DATBIS). Es soll der aktuelle Satz zum Kunden '4711' ermittelt werden:

```
SELECT SINGLE * FROM KUNDE WHERE KUNDENNR = '4711'
AND DATAB <= SY-DATUM
AND DATBIS >= SY-DATUM.
```

Den Erfolg einer SELECT-Anweisung können Sie mithilfe der Systemvariablen SY-SUBRC ermitteln. Ist der Wert 0, war die Suche erfolgreich, und mindestens ein Datensatz (bei SELECT SINGLE genau ein Datensatz) wurde gefunden. Ist der Wert 4, wurde kein Datensatz selektiert, und der Wert 8 bedeutet, dass bei einer SELECT SINGLE-Anweisung mehrere Datensätze selektiert wurden.

Elemente der Programmsteuerung

Soll der Programmablauf in Abhängigkeit von Werten und Ereignissen beeinflusst werden, müssen Steuerstrukturen genutzt werden. ABAP bietet neben den auch in anderen Programmiersprachen verwendeten Grundstrukturen (Verzweigung und Schleife) die Möglichkeit, den Programmablauf ereignisgesteuert zu gestalten.

Grundlegende Steuerstrukturen

Mit ABAP lassen sich die in prozeduralen Sprachen üblichen Verzweigungen und Zyklen realisieren. Die Anweisung `IF <Bedingung>.` verzweigt entsprechend dem Resultat einer Bedingung. Ist die Bedingung erfüllt, werden die Anweisungen im Ja-(`IF`-)Zweig ausgeführt, ansonsten die des Nein-(`ELSE`-)Zweiges. Soll die Abarbeitung des Nein-Zweiges durch eine weitere `IF`-Anweisung aufgespalten werden, können `ELSE.` und `IF <Bedingung>.` zur Anweisung `ELSEIF <Bedingung>.` verbunden werden. Die Anweisung `ENDIF.` schließt die Verzweigung ab:

```
IF X > Y.
WRITE / 'X ist größer als Y'.
ELSEIF X = Y.
WRITE / 'X und Y sind gleich'.
ELSE.
WRITE / 'Y ist größer als X'.
ENDIF.
```

Sollen mehr als zwei alternative Anweisungsfolgen in Abhängigkeit von einem Feldinhalt ausgewählt werden, empfiehlt sich die `CASE`-Anweisung. Mit ihr wird der Feldinhalt mit mehreren Werten verglichen. Es werden dann die Anweisungen abgearbeitet, die auf `WHEN <Wert>.` mit dem übereinstimmenden Wert folgen. `WHEN OTHERS.` bezeichnet die Anweisungen, die ausgeführt werden sollen, wenn der Feldinhalt keinem der vorgegebenen Werte entspricht. `ENDCASE.` schließt die `CASE`-Anweisung ab.

```
CASE X.
WHEN 1.
WRITE / 'X ist 1'.
WHEN 2.
WRITE / 'X ist 2'.
WHEN 3.
WRITE / 'X ist 3'.
WHEN OTHERS.
WRITE / 'X ist weder 1, 2 noch 3'.
ENDCASE.
```

Die wiederholte Abarbeitung von Anweisungen wird mittels Zyklen realisiert. ABAP unterscheidet Zählzyklen und abweisende Zyklen. Die DO-Anweisung wird gemeinsam mit einem Parameter und dem Zusatz TIMES als Zählschleife in der Form DO <Parameter> TIMES. verwendet. Der Parameter gibt die Anzahl der Wiederholungen an. ENDDO. beendet den Zyklus. Die aktuelle Zahl der bisherigen Schleifendurchläufe wird in der Systemvariablen SY-INDEX hinterlegt.

```
* Schleife mit fünf Durchläufen.
DO 5 TIMES.
WRITE / SY-INDEX.
ENDDO.
```

Eine Endlosschleife entsteht, wenn die DO-Anweisung ohne Parameter und TIMES verwendet wird. Diese kann nur durch eine Abbruchanweisung verlassen werden. Diese Abbruchanweisungen sollten nur in Ausnahmefällen verwendet werden. Die Anweisung CONTINUE bewirkt, dass der aktuelle Zyklus nicht zu Ende geführt wird; es beginnt sofort der nächste Schleifendurchlauf. CHECK Bedingung wirkt analog zu CONTINUE, wenn die Bedingung nicht erfüllt ist. Mit der Anweisung EXIT wird eine Schleife ohne Bedingung verlassen.

Die Abarbeitung eines WHILE-Zyklus ist von der Bedingung abhängig, die in der WHILE-Anweisung formuliert wird: WHILE <Bedingung>. Sie wird ausgeführt, sofern und solange diese Bedingung erfüllt ist. Ein abweisender Zyklus wird mit ENDWHILE. abgeschlossen.

```
X = 2.
WHILE X < 1000.
WRITE / X.
X = X * 2.
ENDWHILE.
```

Die Ausführung vieler Anweisungen ist unabhängig von Bedingungen. Diese werden durch logische Ausdrücke definiert. Eine Möglichkeit ist der Vergleich von zwei Feldinhalten oder eines Feldinhalts mit einem festen Wert. Die Vergleichsoperatoren, die Sie im SAP-System nutzen können, sind in Tabelle 4 dargestellt.

Kurzbezeichnung	Zeichen	Bedeutung
EQ	=	gleich
NE	<>, ><	ungleich

Tabelle 4 Vergleichsoperatoren

Kurzbezeichnung	Zeichen	Bedeutung
GT	>	größer als
LT	<	kleiner als
GE	>=, =>	größer gleich
LE	<=, 0<	kleiner gleich

Tabelle 4 Vergleichsoperatoren (Forts.)

Hier einige Beispiele:

```
IF I <> 1.
IF X GE Y.
IF SY-UNAME = 'HFORSTHU'.
```

Mit der Klausel `BETWEEN ... AND ...` kann überprüft werden, ob der Inhalt eines Feldes innerhalb eines vorgegebenen Intervalls liegt. Die Klausel `IS INITIAL` vergleicht den Feldinhalt mit dem Standardinitialwert des entsprechenden Datentyps.

```
* anstelle von 10 <= X <= 20.
IF X BETWEEN 10 AND 20.
* enthält X seinen Initialwert?
IF X IS INITIAL.
```

Zeichenketten lassen sich mit den in Tabelle 5 aufgeführten Operatoren vergleichen.

Kurzbezeichnung	Langbezeichnung	Bedeutung
CO	Contains Only	Erfüllt, wenn der erste Operand nur Zeichen des zweiten Operanden enthält.
CN	Contains Not Only	Erfüllt, wenn der erste Operand nicht nur Zeichen des zweiten Operanden enthält.
CA	Contains Any	Erfüllt, wenn der erste Operand mindestens ein Zeichen des zweiten Operanden enthält.
NA	Contains Not Any	Erfüllt, wenn der erste Operand kein Zeichen des zweiten Operanden enthält.

Tabelle 5 Vergleichsoperatoren für Zeichenketten

Kurzbezeichnung	Langbezeichnung	Bedeutung
CS	Contains Strings	Erfüllt, wenn der erste Operand die Zeichenfolge des zweiten Operanden enthält.
NS	Contains No Strings	Erfüllt, wenn der erste Operand nicht die Zeichenfolge des zweiten Operanden enthält.
CP	Contains Pattern	Erfüllt, wenn der erste Operand das Muster des zweiten Operanden enthält.
NP	Contains No Pattern	Erfüllt, wenn der erste Operand nicht das Muster des zweiten Operanden enthält.

Tabelle 5 Vergleichsoperatoren für Zeichenketten (Forts.)

Dem Größenvergleich zwischen Strings wird eine lexikographische Ordnung zugrunde gelegt. Bei der Verwendung von CP und NP steht das Symbol + für genau ein beliebiges Zeichen und * für eine beliebige Zeichenfolge. Das Symbol # steht vor einem zeichengetreu zu vergleichenden Zeichen. CS und NS unterscheiden nicht zwischen Groß- und Kleinschreibung. Die angegebenen Stringvergleiche können bei der `WHERE`-Klausel allerdings nicht eingesetzt werden, diese sind nur bei `IF`- und `CHECK`-Anweisungen gültig.

Die einzelnen Bedingungen können über `AND` und `OR` logisch verknüpft werden. Sollte es nötig sein, einzelne Teile klar abzutrennen, ist eine Klammerung möglich. Die Verknüpfung `AND` hat zur Folge, dass der Datensatz nur dann selektiert wird, wenn beide verknüpften Bedingungen erfüllt sind. Bei einer Verknüpfung mit `OR` reicht es hingegen, wenn eine der beiden Bedingungen erfüllt ist.

Ereignisorientierte Programmsteuerung

ABAP-Programme werden verwendet, um Daten aus Datenbanktabellen zu lesen, zu verarbeiten und auszugeben. Während der Abarbeitung dieser Programme tritt eine Reihe von Ereignissen auf. Beispiele dafür sind der Programmbeginn, das Lesen eines Datensatzes oder das Beenden einer Bildschirmeingabe. ABAP-Programme können diese Ereignisse auswerten und damit den Programmablauf steuern. Die Ereignisse werden über Zeitpunkt- und Ereignisschlüsselwörter definiert. Diese teilen das Programm in meh-

rere Verarbeitungsblöcke auf. Ein Verarbeitungsblock beginnt mit einem Schlüsselwort.

Das Ereignis `INITIALIZATION.` tritt zu Programmbeginn ein. Mit den zugehörigen Anweisungen können Selektionskriterien und Parameter vor Ausgabe des Selektionsbildschirms verändert werden.

Das Ereignis `START-OF-SELECTION.` findet nach der Bearbeitung des Selektionsbildschirms und vor der ersten Leseoperation auf Datenbanktabellen statt. Seine Angabe ist optional. Nach `START-OF-SELECTION.` können Initialisierungen für die nachfolgenden Anweisungsblöcke programmiert werden.

`END-OF-SELECTION.` bezeichnet den Zeitpunkt nach dem Lesen und Verarbeiten aller Tabellen.

`TOP-OF-PAGE.` und `END-OF-PAGE.` sind Ereignisse bei der Ausgabe von Listen.

Zeichenkettenbearbeitung

Mit `WRITE <Ausdruck> TO <Variable>[+<Off>] [(<Len>)].` kann ein Ausdruck, z.B. eine Variable oder ein Text, statt auf den Bildschirm in eine Variable an der Stelle `<Off>` mit der Maximallänge `<Len>` geschrieben werden. Die meisten Formatzusätze von `WRITE` sind hierbei zugelassen.

`REPLACE <StrALT> WITH <StrNEU> INTO <Feld>.` ersetzt im Feld `<Feld>` das erste Vorkommen der Zeichenkette `<StrALT>` mit der Zeichenkette `<StrNEU>`.

`SHIFT <String> {[BY <Zahl> PLACES] {[LEFT]|[RIGHT} [CIRCULAR]|...}.` verschiebt den String innerhalb des Feldes nach links oder rechts, wahlweise um mehrere Zeichen oder rotierend.

`OVERLAY <Feld1> WITH <Feld2>.` setzt im `<Feld1>` an allen Stellen, an denen sich ein Leerzeichen befindet, das entsprechende Zeichen aus `<Feld2>` ein.

`<Variable> = STRLEN( <String> ).` bestimmt die Länge des Strings in Zeichen.

`CONDENSE <String> [NO-GAPS].` verdichtet den String so, dass Zwischenräume nur noch ein Leerzeichen umfassen bzw. beim Zusatz alle Zwischenräume gelöscht werden.

`TRANSLATE <String> {TO UPPER CASE|TO LOWER CASE|...}.` konvertiert den String in Groß- bzw. Kleinbuchstaben. Es gibt noch weitere Konvertierungsmöglichkeiten.

`CONCATENATE [<String1> <String2> ... <Stringn>] INTO <String> [SEPARATED BY <Zeichen>].` verkettet die angegebenen Strings zu einer in `<String>` gespeicherten Zeichenkette. Die einzelnen Teile können dabei durch ein Zeichen `<Zeichen>` getrennt werden.

`SPLIT <String> AT <Zeichen> INTO [<String1> <String2> ... <Stringn>| TABLE <Tab>].` zerlegt einen String jeweils am Trennzeichen `<Zeichen>` und speichert die einzelnen Teile wahlweise in eine Liste von Feldern (Strings) oder in eine »interne« Tabelle.

Unterprogramme

Parameter des Unterprogramms werden in Referenzparameter und Wertparameter unterschieden. Bei einem Referenzparameter wird nur die Adresse des Aktualparameters an das Unterprogramm übermittelt. Wird der Wert eines solchen Parameters im Unterprogramm verändert, wird die Änderung direkt auf der Variablen im Hauptprogramm vollzogen. Die Definition erfolgt mit dem Schlüsselwort `USING`.

```
FORM Unterprogramm1 USING FELD1 FELD2.
```

Wertparameter sind eigenständige Variablen des Unterprogramms. Beim Aufruf werden die Werte der zugehörigen Aktualparameter an die Formalparameter übergeben. Die Werte der Aktualparameter bleiben erhalten. Damit können diese Wertparameter nur als Eingabe- und nicht als Ausgabeparameter des Unterprogramms genutzt werden.

Die Definition erfolgt mit `USING VALUE`.

```
FORM Unterprogramm1 USING VALUE(FELD1) VALUE(FELD2).
```

Eine Mischform ist die Definition von Parametern mit dem Schlüsselwort `CHANGING VALUE(<Feld>)`. Mit `CHANGING VALUE(<Feld>)` wird ein Wertparameter vereinbart, dessen Wert am Ende des Unterprogramms wieder auf den zugehörigen Aktualparameter übertragen wird. Damit ist dieser Parameter sowohl für Eingaben als auch für Ausgaben aus dem Unterprogramm anwendbar. Eine Wertübergabe erfolgt auch, wenn das Unterprogramm mit `EXIT.` abgebrochen wird. Sie erfolgt nicht bei einem Abbruch mit einer Fehlernachricht.

```
FORM Unterprogramm1 CHANGING VALUE(FELD1) VALUE(FELD2).
```

Interne Tabellen können nicht mit USING übergeben werden. Für sie ist der Zusatz TABLES <Tabellenname> notwendig. Dieser Zusatz muss als erster nach den Schlüsselwörtern FORM oder PERFORM angegeben werden.

```
FORM Unterprogramm1 TABLES TAB1.
```

Bei der Übergabe von Feldleisten und Tabellen an Unterprogramme ist es möglich, zusätzlich deren Struktur mitzuteilen. Dazu fügen Sie in der FORM-Anweisung direkt hinter dem entsprechenden Parameternamen den Zusatz TYPE <Strukturangabe> an. Durch diesen Zusatz wird gesichert, dass die Typen der Formalparameter mit den Typen der Aktualparameter kompatibel sind. Falsche Typangaben führen zu Syntaxfehlern. Typkonvertierungen werden nicht ausgeführt.

Ohne diese Angaben kann auf die Tabellen nur mit zeilenorientierten Operationen, wie APPEND oder READ, zugegriffen werden. Auch auf Feldleisten kann nur als Ganzes zugegriffen werden. Ihr Inhalt wird als Zeichenkette interpretiert. Mit der Strukturangabe können Sie wie gewohnt mit Einzelfeldern der Tabellen und Feldleisten arbeiten.

```
TYPES:BEGIN OF TABTYP OCCURS 10,
NAME(10) TYPE C,
ORT(15) TYPE C,
END OF TABTYP.
DATA TAB1 TYPE TABTYP.
...
PERFORM Unterprogramm1 TABLES TAB1.
...
FORM Unterprogramm1 TABLES TAB1 TYPE TABTYP.
...
```

Werden mehrere Parameter übergeben, müssen die Zusätze TABLES, USING und CHANGING in folgender Reihenfolge angegeben werden:

```
FORM Unterprogramm1TABLES TAB1
USING FELD1
CHANGING VALUE(FELD2).
```

Soll im Unterprogramm auf Datenbanktabellen zugegriffen werden, müssen diese mit der Anweisung LOCAL <Tabellenname>. vereinbart werden.

```
FORM Unterprogramm1
LOCAL USR03.
...
ENDFORM.
```

Funktionsbausteine

Funktionsbausteine sind in einer Zentralbibliothek verwaltete, logisch abgeschlossene Unterprogramme mit einer fest definierten Schnittstelle zum Nutzer. Aufgerufen werden Funktionsbausteine mit der `CALL FUNCTION`-Anweisung.

```
CALL FUNCTION <Baustein>
{EXPORTING [<Formalparameter>=<Aktualparameter>]1:}
{IMPORTING [<Formalparameter>=<Aktualparameter>]1:}
{CHANGING [<Formalparameter>=<Aktualparameter>]1:}
{TABLES [<Formalparameter>=<Aktualparameter>]1:}
{EXCEPTIONS [<Ausnahme>=<Fehlercode>]1:
    [OTHERS=<Fehlercode>]}.
```

Die Option `EXPORTING` wird für Eingabeparameter verwendet. Mit `IMPORTING` werden Ausgaben von Formal- an Aktualparameter deklariert. `CHANGING` definiert Ein- und Ausgabeparameter. Mit `TABLES` werden Tabellen übergeben. `EXCEPTIONS` dient der Ausnahmebehandlung. Damit können bei vorzeitigem Abbruch der Abarbeitung zusätzliche Verarbeitungsschritte eingeleitet werden.

Da jeder Funktionsaufruf spezifische Optionen hat, ist auch dessen Aufruf spezifisch. Deshalb ist es am einfachsten, über die Funktion `Muster` und die Eingabe des Bausteinnamens den Aufruf in den aktuellen Programmcode einfügen zu lassen. Dabei müssen Sie nur noch die Aktualparameter ergänzen.