IMSQL

IMS – Query Language

Funktionsbeschreibung

VERSION 4.1

FUNKTION DES PROGRAMMES

Das Programm IMSQL ist ein funktionaler Compiler von SQL- bzw. BASIC- ähnlichen Programmsteuerkommandos zur vollständigen Wartung beliebiger IMS - Datenbanken, OS-Files und VSAM - Cluster. Läuft das Programm unter IMS sind IMS- DB2 und OS-Zugriffe beliebig kombinierbar, unter nativem-z/OS sind nur OS-Zugriffe möglich.

Alle Felder der Datenbanken/Files können namentlich angesprochen oder formatfremd oder überlagernd bearbeitet oder gelistet, die Segmente geprüft, ersetzt, gelöscht oder eingefügt werden.

Zusätzlich besteht die Möglichkeit sowohl den Programm- wie auch den Informationsfluss über dem PSB bekannten Input-GSAM-Datenbanken zu steuern, bzw. über Standard-OS-Files oder VSAM.

Die jeweiligen DCB's können unter Berücksichtigung des Satzformates beliebig überschrieben werden.

Steuerfunktionen:

Zur Steuerung des Programms dient ein frei formatierter Kommandofile unter dem DD-Namen SYSCMD, der, 80-stellig, als TSO - Dataset angelegt, oder direkt in die JCL eingefügt werden kann. Signifikant sind, wie bei JCLs oder Host-Programmen üblich, nur die Stellen 1 bis einschließlich 72. Da der Kommandofile frei formatierbar ist, sind weder Positionierung noch etwaige Unterstützungen durch CLIST, REXX oder ähnliches nötig.

Mit (i) gekennzeichnete Befehle sind nur unter IMS verfügbar, und haben unter z/OS allein keine Funktion.
Mit (m) gekennzeichnete Befehle können nur auf OS-Files, jedoch in jedem Environment angewandt werden.

- Checkpointschreibung (i)

Uneingeschränkte Restartfähigkeit ist durch eine automatische Checkpointschreibung realisiert. Optional kann ein vollautomatisches Wiederaufsetzen mittels einer eigenen Checkpointdatenbank bei der Implementierung aktiviert werden. Für MPP-PSBS oder DLI- Anwendungen ist diese Funktion deaktivierbar. Automatische Checkpoints werden ausschließlich auf Level 1 (= äußerste Programmschleife) gezogen, das heißt, weder in Unterprogrammen noch beim Bearbeiten von Dependent - Segmenten oder geschachteltem Lesen von Roots. Die Frequenz ist bei Implementierung des vollautomatischen Restarts von der momentanen Auslastung der Maschine, den konkurrierenden DB-Prorammen (MPP's und BMP's), von der Puffer-Anzahl in der JCL, von der Anzahl der gleichzeitig positionierten Datenbanken, von der Anzahl der datenbankverändernden Zugriffe und von der Execution-Time abhängig, bei nativem Checkpoint nur von der Zugriffsanzahl und der Execution-Time.

Falls Sync-Points nur an bestimmten Stellen im Programm zugelassen sein sollen, kann die Checkpointschreibung mittels der CHECK <n> - Anweisung auf diese beschränkt werden.

Mit NOCHECK wird jede Checkpointschreibung unterbunden (z/OS-Standard)

BEFEHLSVORRAT

I nicht ausführbare Kommandos

REM / REMARK / * / - Kommentar, kann überall im Programm kodiert werden

nicht ausführbare Kommandos, die vor dem ersten ausführbaren Befehl kodiert werden müssen oder nach dem das Programm abschließende END Statement:

NOUPDATE	**(i)**	- Zurücksetzen der DB-Veränderungen
FILE		- zur Definition GSAM - DB / OS- / VSAM-File
OSFILE	**(i)**	- zur Definition OS- / VSAM-File, unter IMS obligat
VAR / VARIABLE		- Variablendeklaration
DBD	**(i)**	- DBD-Deklaration für unqualifizierten DB-Zugriff
USES		- Segmentbeschreibung oder Include explizit laden
MAP		- Segmentbeschreibung oder Include explizit generieren
ALTER		- Segment - Attributänderung
NOCHECK	**(i)**	- Keine Checkpointschreibung (z/OS-Standard)
NOIMS	**(i)**	- IMS_Zugriffe unterbinden (z/OS-Standard)

II Testhilfen

TRACE	- verschiedene Testhilfen je nach Level
NOTRACE	- ausschalten des Trace
SHOW	- Segment/Variablenausgabe unformatiert auf SYSPRINT

III Unterprogramme und externe Calls

PROC	- Beginn Unterprozedur muss vor dem ersten ausführbaren Befehl kodiert werden
LOAD	- Ladebefehl externes Unterprogramm / externe Tabelle aus STEPLIB - Verkettung oder explizit angegebenem Dataset

IV Datenzugriffe

SELECT	- Datenbank oder File lesen
REPLACE	- DB - Segment / VSAM ersetzen
DELETE	- DB - Segment / VSAM löschen
INSERT	- DB - Segment einfügen,
LIST	- Ausgabe auf File
SEARCH	- Segment aus interner Speicherkette lesen

V System Zugriffe

CHECK (i)	- Spezieller Checkpoint
SYNC (i)	- Spezieller Syncpoint
ROLB (i)	- Explizites Rollback
OPEN (m)	- Explizites Open auf File (z/OS)
CLOSE (m)	- Explizites Close auf File (z/OS)

VI Bereichssteuerung

WHERE	- Gültigkeitstest d. vorausgehenden SELECT - Statements, auf alle Keyed- DB-/VSAM-Segmente anwendbar

VII Datenmanipulation

LET	- Wertzuweisung
PUSH	- Wertsicherung
POP	- Wertwiederherstellung
STORE	- Segmentspeicherung
RESTORE	- Update gespeichertes Segment
FIRST	- Positions-Call suf STORED-Bereich
RELEASE	- Freigabe STORE - Bereich

VIII Konditionale

IF	- Prüfung auf Wertübereinstimmung
AND	- detto
OR	- detto
ELSE	- Bei Nicht-Übereinstimmung der vorausgehenden IF/AND/OR - Kombination
YES	- spezieller Condition-Test, siehe unten
NO	- spezieller Condition-Test, siehe unten
ENDIF	

SWITCH	- Switchgruppe (RPF/PLI: SELECT)
WHEN	- entsprechend IF - Konditional,
<AND>	- 'OR' nicht unterstützt
WHEN	
...	
WHEN NONE	
ENDSWITCH / ENDSW	

IX Programmfluss-Steuerung

LOOP	- entspricht einer BCT-Schleife im Assembler
DO	- Unterprogramm Aufruf
RETURN	- entspr. Rücksprung
CALL	- Aufruf externes Unterprogramm
GOTO	- unbedingter Sprung
LABEL	- Ansprungadresse

X Block- und Programmabschlüsse

ENDLOOP	- Abschluss jeder LOOP - Schleife
ENDSEL	- Abschluss jedes SELECT-Blockes
ENDSRCH /	- Abschluss jedes SEARCH-Blockes
ENDIF	- Abschluss jedes IF-Blockes
ENDPROC	- Abschluss einer Unterprogrammdeklaration
ENDSW / **ENDSWITCH**	- Abschluss jeder SWITCH - Gruppe
ENDFILE	- optionaler Abschluss einer Filedeklaration
END	- Reguläres Beenden des Programms, muss zumindest als letztes Statement im Kommandofile kodiert werden.

XI Builtin - Funktionen

LENGTH	- Länge einer Variablen / eines Bereiches
INDEX	- Position String in String
SUBSTR	- Teilzeichenfolge
VERIFY	- String-Inhalt-Prüfung
COUNT / **COUNTER**	- Zugriff auf interne DB-Zähler
SCALE	- Änderung der Kommastellenanzahl,
ADDR	- Adresse einer(s) Variablen / Bereiches
ABS	- retourniert Absolutwert (nur gepackte Argumente!)
PCB (i)	- retourniert die Adresse des mit dem Segment-Argument verbundenen PCB
DECNUM	- Prüfung auf gepackten Feldinhalt
(name**)**	- zusätzliche Klammerung bei CHAR-Variablen bewirkt die Substitution durch den Wert der Variablen deren Name zur Laufzeit Inhalt der Variablen 'name' ist.

XII Systemvariable nicht überschreibbar, wenn nicht anders vermerkt

_SEGINT	- Pointer auf interne Segmentverwaltung
_SEGMAX	- Anzahl der generierten / geladenen Segmente
_CURSEG	- Name des Segmentes der letzten IO - Operation
_CURSLVL	- Hierarchischer Level des _CURSEG
_CURKFB	- aktuelle Key-Feedback-Area
_CURSTC	- aktueller IMS - Status-Code
_CURFLD	- zuletzt verarbeiteter Feldname
_CURLINE	- aktuelle Verarbeitungszeile (überschreibbar)
_ISCHKP	- Flag erstes Statement nach Checkpoint
_DATUM	- aktuelles Datum

INTERNE FUNKTION IMSQL

Die Abarbeitungsbasis des Programms ist ein vom User erstellter Kommandofile, bestehend aus einer mehr oder weniger langen Kette von Einzelbefehlen (zur Zeit sind maximal 1000 Zeilen zugelassen). Die Commands müssen vollständig auf je einer Zeile kodiert werden, Fortsetzungszeilen existieren nur bei Deklarationen (siehe unten). Das Compilat fußt, ähnlich Basic, auf der sequentiellen Abarbeitung der gelabelten Einzelzeilen.

Dieser Kommandoinput wird zeilenweise je nach Befehl in eine oder mehrere funktionale Instruktion(en) (Token) übersetzt. Diese werden dann zur Laufzeit abgearbeitet und bieten, da sie einheitlich formatiert sind und alle Informationen zur Abarbeitung des (Teil-) Befehles in einem ’Systemwort' beinhalten, durch direkten Sprung in den jeweiligen Funktions-Entry eine Laufzeit-Performance, die sich nur wenig von einem speziell kodierten PLI-Programm zur Lösung des anstehenden Problems unterscheidet.

Die Variablenspeicherung

Alle Variablen, das sind Segmentfelder, die userdefinierten Variablen, die Felder der GSAM-Bereiche und auch die Konstanten, die innerhalb der Kommandozeilen eingegeben wurden, werden einheitlich und eindeutig durch jeweils eine Attributstruktur beschrieben.

Userdefinierte Variable, seien sie in File-Bereichen oder frei allocatiert, erhalten den ihnen zugeteilten Namen, die Segmentfelder werden via MAP aus ASM-, COB- oder PLI- Includes generiert. und die im Kommando-Source-Code entaltenen Konstanten (Zahlen oder Strings) werden mit 'K#nnn' bezeichnet, wobei 'nnn' die fortlaufende Konstantennummer bedeutet und die Attribute aus dem Zeichen-Stream genommen werden.

Weitere systemgenerierte Variable sind:

A#nnn -	Pseudovariable der ADDR-Funktion
S#nnn -	Pseudovariable der SUBSTR-Funktion
W#nnn -	Pseudovariable der LENGTH- und SCALE-Funktionen

Segmentfelder können nicht lokal im Programm selbst definiert werden, sondern sind am Programmbeginn via MAP aus ASM- oder PLI-Includes zu generieren, bzw. aus einer speziellen PO-Datei zu laden (proprietäres Format). Die interne Darstellung entspricht jedenfalls auch bei diesen der aller anderen Variablen.

Der Aufbau des Kompilates.

Es handelt sich bei dem Kompilat nicht um ein ausführbares Programm im IBM - 370 Maschinencode, sondern um parametrisierte Funktionsaufrufe (Token), die zur Laufzeit mit einem Minimum an Instruktionen realisiert sind.

Der das Kompilat unterstützende Pseudoprozessor ist als Akkumulatorprozessor ausgelegt. Als Arbeitsregister stehen zur Verfügung:

ein Akkumulator mit maximaler Speicherkapazität von 32767 Bytes,
ein Arbeitsregister in derselben Länge sowie ein Indexregisterstack beliebiger Tiefe (Halbworte).

Syntax und Anwendung: (siehe auch Programmbeispiele am Ende der Dokumentation)

- Allgemein:

  Alle Befehle, Operanden, Variablen etc. müssen immer durch ’ ’ (Blank) voneinander getrennt werden, damit das Programm eine korrekte Trennung durchführen kann! Das gilt auch für die mathematischen Operatoren mit Ausnahme des '-' (Minus) zur Kennzeichnung einer negativen Konstanten, das direkt vor der ersten Ziffer stehen muss! Kommas (',') als Trennzeichen zwischen den Kommandoteilen sind nicht unterstützt, mit Ausnahme der impliziten SET - Angabe bei IF - Statements und als Zeilenfortsetzung bei Deklarationen.

  Als Kennzeichnung einer Zahl mit 'Komma'-stellen ist nur der Dezimalpunkt zulässig, z.B. 47.11 .

- Wichtig:

  Nicht unterstützt sind, in Ermangelung eines Coomand-Stacks, Klammerungen sowie die ansonsten selbstverständliche algebraisch richtige Operationspriorisierung wie Punkt- vor Strichrechnung! Das heisst: 3 + 4 * 2 ist 14 und nicht 11 !! Die Operationszuweisung funktioniert strikt von rechts nach links, die Akkumulierung mehrfacher Berechnungen aber von links nach rechts in der Reihenfolge der Kodierung.

- RC:

  Bei syntaktisch richtigem Input und unter der Bedingung, daß auch zur Laufzeit keine Fehler aufgetreten sind, beendet das Programm mit Returncode 0,.

  Wurden Syntaxfehler im Inputstream erkannt, dann wird vor der Laufzeitabarbeitung mit Returncode 16 beendet.

  Alle anderen Returncodes kommen aus Fehlern während der Laufzeit und müssen auf Grund der entsprechenden Fehlerhinweise auf SYSPRINT analysiert werden.

Sämtliche Befehle auch in Deutsch eingegeben werden, die z.Z. implementierte Synonymtabelle entnehmen Sie bitte dem Anhang.

I DIE NICHT - AUSFÜHRBAREN BEFEHLE :

- Diese müssen, mit Ausnahme des Kommentars immer am Anfang des Programms stehen.

REM / REMARK / *

Kommentar
Es können beliebig viele Kommentarzeilen definiert werden, da kein Code generiert wird.

Bei Originaldurchführung sollte die Funktion des Steuerfiles immer im Header kurz beschrieben werden.

NOUPDATE (i)

Vor jedem Checkpoint, bzw. am Ende der Verarbeitung werden alle Datenbankveränderungen zurückgesetzt. Das Rücksetzen erfolgt via Roll-Back somit sind die internen IMS Prüfroutinen auch bei REPLACE, DELETE und INSERT vollständig aktiv.

Nur bei IMS - Datenbanken unterstützt, Änderungen an OS-Files und VSAM-Clustern bleiben erhalten!

FILE filnr/gsamdbd/DDname <INP/OUT/UPD> <VSAM/GSAM> <KEY offs lng>
<LENGTH ll> > <DSN=dataset.name>
<NOOPEN>

Vor der Programmausführung müssen die verwendeten OS- und GSAM - Files deklariert werden. Dies geschieht mit dem FILE - Befehl.

filnr (i) - immer GSAM, Filenummer 1 bis n
die relative Position der GSAM-PCBs im PSB
d.h.: 1 .. erste GSAM-DB
2 .. zweite GSAM-DB
..
n .. n-te GSAM-DB nur unter IMS

oder

gsamdbd (i) - NAME der GSAM-Datenbank laut verwendetem PSB
z.B.: GSAMDB1, DPPRINT ...

oder

DDname (m) - DD-Name OS-File

<INP/INPUT/ <UPD> (m) - Datenrichtung, im IMS optional, fuer OS-Files vorgeschrieben
UPD bedeutet immer auch VSAM

<VSAM> (m) - VSAM-Bestand, optional bei UPD, bei Input-Output-Beständen zur Unterscheidung von sequentiellen Beständen vorgeschrieben.

<GSAM> (i) - GSAM-Datenbank, optional bei filnr sonst zur Kennzeichnung einer Gsamdatenbank vorgeschrieben.

<LENGTH ll> maximale interne Verarbeitungslänge des Files, sollte um unnötig lange Moves zu vermeiden immer angegeben werden
ll = Anzahl Bytes. Maximale Länge bei I/O Operationen sind 32760 Bytes!

<KEY offs lng> **(m)** - Key-Definition bei VSAM-Beständen in der Form ab Offset offs vom Beginn des Satzes (+0) in der Länge lng.

<DSN=... > (m) - Datasetname, der dynamisch mit dem DD-Namen verknüpft werden soll.

<NOOPEN> (m) - kein automatisches Open bei Laufzeitbeginn, File muss explizit via OPEN geöffnet werden.

Zur Definition des Fileaufbaues genügt es direkt nach der jeweiligen Filedeklaration die zugehörigen Felder in jeweils eigenen Zeilen anzuführen (Syntax siehe VARIABLE).

```
FILE 1 INPUT oder FILE FSAMDB1 GSAM
 BLZ         FIXED(5)
 KTO         FIXED(11)
 NAME        CHAR(28)
 KENNZ       BIT(8)
*
FILE 2 OUTPUT
 SATZART     BIN(7)
 KONTO       CHAR(9)
 PAD1        CHAR(2)
 NAME        CHAR(28)
 KAPI        CHAR(15,2)
 KZ          CHAR(1)
*
LET SATZART = 9
...
```

Die Definition des jeweiligen Fileaufbaues endet immer mit der nächsten Zeile, die ein gültiges Kommando enthält, oder mit einem ENDFILE - Kommando. Die Filestrukturen müssen nicht ausdefiniert werden, da die aktuelle LRECL immer aus dem DCB des damit verknüpften DD - Statements genommen wird. Es muss nur sichergestellt sein, dass die mit INPUT definierten Files auch mit dem richtigen DD-Namen und dem richtigen Dataset verbunden sind. Analoges gilt für die Outputs. Vor allem muss der DCB in der JCL immer explizit angegeben werden!

Unter z/OS benötigt man keine explizite DCB-Angaben in der JCL, dafür sollte die maximale Filelänge mittels LENGTH-Option bekanntgegeben werden.

Wenn nach der File-Deklaration USES oder MAP - Befehle gesetzt werden, so interpretiert das Programm diese Deklaration als zum File gehörig, d.h. die Feldinhalte der Struktur werden im I/O-Buffer des Files angelegt. Ist dies nicht gewünscht, so ist die Filedeklaration explizit mittels ENDFILE abzuschließen.

Bei der Deklaration eines Files wird ebenfalls ein Pseudosegment mit dem angegebenen Namen angelegt und die nachfolgenden Felder auch diesem zugeordnet, d.h. die Feldinhalte können auch in der Form filnam.feld angesprochen werden. Feldnamen können sich daher in jeder Strukturen gleichnamig wiederfinden. Dies ermöglicht dann eine einfache feldweise Zuweisung mittels einer einzigen BY NAME - Anweisung. Außerdem kann der gesamte File-Bereich wie auch jedes DB-Segment, als zusammenhängender String bearbeitet und zugewiesen werden.

Dem aufmerksamen Leser werden die zwei Besonderheiten im obigen Beispiel nicht entgangen sein:

SATZART	BIN(7)	und
KAPI	CHAR(15,2).	

Die Erklärung und die allgemeinere Syntax der Variablendefinition erfährt er im folgenden Kapitel.

VAR / VARIABLE name(ind) attr(länge<,komma>) <ORG * / name1<+nnn>>
<HOLD>
<INIT(..)>

Die VARIABLE - Deklaration dient zur Generierung eines Datenspeichers mit den zugehörigen Attributen. Aber anders als bei Datenbank - Feldern, die ihre zugehörigen Wert-Bereiche über die PSB-Pointer adressieren oder die File - Variablen, die über den I/O-Bereichen liegen, werden bei expliziter Variablendeklaration die Bereiche für die Wertaufnahme frei im Hauptspeicher in angeforderter Variablenlänge allocatiert. Variable erhalten bei der Anlage den Wert X'0', bzw. gepackte Felder den Wert P'0' in der angegebenen Länge (ausgenommen 'ORG'-Variable). Sie können jedoch auch initialisiert werden.

Wichtig:
Bei Kombination von ORG * und INIT ist nicht zugelassen!

name - eindeutiger Bezeichner, der allerdings in den DB-Feldern oder Filebereichen bereits verwendet worden sein kann (Lokale Variable sind Level-1 - Variable und haben keinen node). Mehrfach gleiche Level-1-Variable oder Felder innerhalb einer Filedeklaration werden nicht erkannt, es wird immer nur der erste verwendet! Hier ist auf Eindeutigkeit zu achten!

ind - Index bei Array-Deklaration. Nur ein Index kann angegeben werden. Dieser jedoch in beliebiger Tiefe, Mehrdimensionale Deklarationen wie VAR(3,4) sind nicht zugelassen.

attr - Attribut der Variablen, gültig sind:

CHAR / CHARACTER	-	Zeichenvariable
BIN / BINARY	-	Binärzahl
DEC / DECIMAL	-	Fixpunkt Dezimalzahl (Float ist nichtvimplementiert)
FIX / FIXED	-	alleine wie DEC, sonst wie 2. Attribut
BIT	-	Bitstring
BYTE	-	Binärer Code, Darstellung des Inhaltes als Zahl

länge - Länge in Abhängigkeit vom Attribut:

CHAR	-	Bytes
DEC	-	(länge + 1) / 2 Bytes
BIN	-	länge/8 + 1 Bytes, daher auch BIN(7) und BIN(23) mögl.
BIT	-	(länge-1)/8 + 1 Bytes
BYTE	-	immer 1

Komma - Anzahl Kommastellen (wahlweise)

CHAR	-	Bytes Die Angabe einer Kommastelle bei CHAR ermöglicht die Formatierung der Zahlen im Picture - Format
DEC	-	Halbbytes
BIN	-	nicht implementiert, Ganzzahl
BIT	-	nicht implementiert, wird für die Position verwendet
BYTE	-	nicht implementiert

ORG - Feldüberlagerung (wahlweise, Keywort), wenn spezifiziert muß name1 angegeben werden. Der Wert der Variablen name liegt somit über dem Wert einer anderen bereits definierten (entspricht BASED(ADDR(name1)))

name1 -Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert.

+nn../-nn.. Offset vom Beginn des Wertes des Feldes name1 in Bytes (wahlweise). Es ist darauf zu achten, dass das '+'-Zeichen direkt auf den Namen des Zielfeldes folgt. Der Beginn des Zielfeldes liegt bei +0. Vorsicht bei negativen Offsets!

*** -** Für die Variable wird kein Speicherplatz allocatiert. Enstpricht BASED in PL/1.Es ist darauf zu achten, dass diese Variablen vor der Verwendung mittels der Pseudovariablen ADDR(name) auf einen gültigen Speicherbereich positioniert wird.

HOLD -Der Inhalt der Variablen wird in speziellen Datenbereichen abgelegt, die beim Checkpoint gesichert werden. Im Falle eines Restarts werden diese Variableninhalte wiederhergestellt. Speziell für interne Zähler oder Kennzeichen. Die Gesamtkapazität ist jedoch beschränkt, insgesamt stehen 32760 * 4 Bytes zur Verfügung, die einzelne Variable oder Tabelle kann nicht größer als 32760 Bytes lang sein.

HOLD und ORG schließen einander aus. Siehe auch ADDR().

INIT(..) Wie in PL/1 kann der Variablen bereits zur Deklarationszeit ein Wert zugewiesen werden. Die Initialisierungen werden am Beginn der Laufzeit in der Reihenfolge der Codierung durchgeführt. Bei Overlayvariablen (ausgenommen ORG *) zieht somit der letzte auf diesen Speicherbereich zugewiesene Wert.

Bsp.:
```
VAR OVLINDB BIN(7) ORG DBSEG1.BITLEISTE
VAR NAME CHAR(28)  INIT('HUGO HABICHT')
VAR VNAME CHAR(14) ORG NAME INIT('PETER')
VAR NNAME CHAR(14) ORG NAME+14 INIT('PAN')
      ...
```

Inhalt der Variablen NAME: 'PETER PAN'

VARIABLEN - ARRAYS

Wurden Variable als Array angelegt oder sind sie als solche in der Datenbankbeschreibung deklariert, dann kann auf die indizierte Variable sowohl mit einem konstanten Index, z.B. LAGER.FACH(3), wie auch über variable Indizierung zugegriffen werden. Der Typ der Indexvariablen ist frei wählbar, solange deren Inhalt auf eine Ganzzahl konvertiert werden kann

z.B INCOME.UMSAETZE(LAUF).

Die Schachtelungstiefe der Indizierung ist im Prinzip nur auf die maximale Zeilenlänge von 72 Stellen begrenzt, Konstrukte wie LET INCOME.UMSAETZE(I1(I2(I3 * 2(I4 + 3(I5(I6(In....))))))) = A(B(C...)) sind möglich, doch ist die Laufzeitberechnung der Indizes etwas zeitraubend. Pro Index werden zusätzlich 2 Funktionsaufrufe benötigt, es empfiehlt sich daher im Sinne der Performance Indizierungen wenn möglich zu vermeiden, bzw. mit Konstanten zu indizieren, da diese keine Laufzeitverzögerungen bedeuten. Andererseits bewirkt dies meist eine Erhöhung der Statementanzahl, was auch wieder Zeit kosten.

Bei der Indexauflösung sind mathematischen Funktionen im Rahmen der allgemeinen Syntax gestattet. Alle Arrays beginnen mit Index 1, Bereiche wie 0 .. 15 sind nicht unterstützt.

Arrays von Strukturen sind nur bei Strukturdefinitionen via USES oder MAP möglich.

USES segment <ORG bez1> <segment <segment <...>>>

Werden Segmente dynamisch angesprochen (SELECT (variable)) müssen diese bei Programm-durchführung bereits geladen sein. Dies erreicht man durch das Dummy - Statement
'USES segmentname'.

segment - gültiger Segmentname des unterstützenden PSBs, oder beliebiges vorausgeneriertes Strukturmember. Geladen wird via DD-SYSSEG.

ORG bez1 - analog den Variablen können auch geUSEte Segmente andere Bereiche überlagern.

Es empfiehlt sich, alle angesprochenen Datenbanksegmente aus Dokumentationsgründen prinzipiell dem Programm mittels einer oder mehrerer USES-Anweisungen im Voraus bekanntzugeben.
Zusätzlich zu den Systemvariablen wie Akkumulator, Index- und Rechenregister wird für jedes geladene Segment eine Variable mit der Bezeichnung segmentname im Characterformat angelegt, deren Inhaltpointer auf den Segmentbuffer zeigt. Jedes Segment ist daher direkt namentlich als Zeichenvariable ansprechbar und kann komplett oder partiell bearbeitet oder ausgegeben / eingelesen werden.

Achtung:
Felder aus im File-Bereich generierten Segmenten sind nur via neuem Segmentnamen und nicht über den File-Namen als 'First Qualifier' oder deren Kombination ansprechbar:

```
FILE INPUTDB INP GSAM
 ANFANG CHAR(10)
 USES SEGMENT
 ...
ENDFILE
*
LET ANFANG = 'BEGINN' - oder
LET INPUTDB.ANFANG = 'BEGINN'
LET SEGMENT.Feld = ...  und nicht INPUTDB.Feld oder INPUTDB.SEGMENT.Feld
```

Alternativ zu USES können die Segment- oder Strukturbeschreibungen auch mittel MAP-Anweisung (sehe unten) dynamisch aus den Originalincludes (PLI, COB oder ASM) generiert werden. Bei der Auflösung komplexerer Macro-Anweisungen kommt es jedoch hie und da zu Problemen.

MAP segment <<LAN=/LANG=/LANGUAGE=>ASM/PLI/COB> <,>
<DSN=data.set.name> / <DD=/DDN=DD-Statement>
<NAME=name2>
<MEM=/MEMBER=mem>
<DB=/DATABASE=database>
<OCC=/OCCURENCY=nn>
<SEQ=/PROCSEQ=/INDEX=Indexdb>
<KEY=keyfeld>
<EXIT=exitname>
<PARENT=parent>
<ORG feldname<+Offset / LENGTH(variable/segment)>>
<NOSRT/NOSORT/UNSORT>
<FIX>

Erstellung der Strukturbeschreibung ohne SYSSEG zur Laufzeit aus ASM-/PLI-/COBOL - Include.

Segment Strukturmember, das generiert werden soll (Positionsparameter, muss angegeben werden)

<LAN=/LANGUAGE=/LANG=> ASM, PLI oder COB - Default = ASM
(zur Zeit nur ASM, PLI und COB unterstützt).

<,> Die MAP-Instruktion kann über mehrere Zeilen gehen, Abschluss jeder Zeile durch ´,´

<DSN=/DATASET=/DA=> Datasetname in dem sich das Include befindet (Default kann bei der Implementierung angegeben werden) -oder-

<DD=/DDN=> DD-Name unter dem das Include-Dataset in der JCL verknüpft ist

<NAME=> Segmentname lt. Datenbankbeschreibung, wenn von 'segment' unterschiedlich

<MEM=/MEMBER=> Membername lt. DSN/DDN, wenn von 'segment' unterschiedlich, idente Strukturen können hiermit unter unterschiedlichen ’segment’-Namen mehrfach geladen werden.

<DB=/DATABASE=> Datenbankname bei DB-Segmenten

<OCC=/OCCURENCY=> Position der DB im PSB wenn mehrfach ident vorhanden (default = 1)

<SEQ=/PROCSEQ=/ INDEX=> Name der Sekundaerindex-DB wenn indiziert

<EXIT => Exitprogramm, das bei jedem Datenbankzugriff sowohl als Vor - als auch als Nachexit aufgerufen wird. Die Parametrisierung entnehmen Sie bitte dem Anhang.

<KEY=> Bekanntgabe des Segment-Keys bei DB-Segmenten der Bezeichner keyfeld muss im Include deklariert sein

<PARENT=> Bekanntgabe eines etwaigen Parents bei Dependent-Segmenten. Das entsprechende Segment muss entweder über SYSSEG ladbar oder mittels MAP bereits generiert sein.

Optionen: **Spezielle Keywortfunktionen:**

<ORG feldname<+Offset / LENGTH(variable,segment)>>Das Segment wird als Overlay über eine bereits definierte Variable feldname extl. + Offset angelegt (siehe VAR) das zugeordnete Muttersegment wird als Vorgänger einer CHILD-Kette markiert, Änderungen an deren Speicherzuordnung (ADDR(muttter) = ...) bedingen auch eine Änderung der Adressen der 'CHILDREN' und deren Felder. Als Offset ist auch die relative Angabe einer Feld- oder Segmentlänge (LENGTH(x)) zulässig. Allerdings ist dann die automatische CHILD - Adressabhängigkeit nicht mehr aktiv.

<NOSRT/NOSPRT/UNSORT> Segment ist auf DB nicht aufsteigend sortiert, nur auf DB-Segmente anwendbar

<FIX> Segment ist fix lang (keine Satzlänge am Anfang)

MAP und USES können beliebig gemischt werden, z. B.:

MAP a PARENT=b (implizites USES auf B)

USES b
MAP a PARENT=b (explizites USES auf B)

MAP b
MAP a PARENT=b (explizites MAP auf B)

Implizit oder explizit zu ladende Strukturen müssen in der SYSSEG-Verkettung vorhanden sein.

Zum völlig unqualifizierten Lesen von IMS-Datenbanken kann jede Datenbank auch immer als reiner DBD- IO-Bereich implementiert werden:

DBD database <OCC=/OCCURENCY=nn>
<NOSEG>

via SELECT database <MULT>

sind unqualifizierte Segmentzugriffe ohne Segmentspezifikation (bei Angabe von NOSEG) möglich. Der Datenbankzugriff erfolgt sequentiell ohne SSA bei der aktuellen Position. Der Name des erhaltenen Segmentes wird in der Systemvariablen _CURSEG retourniert. Ist NOSEG nicht angegeben, wird der gelesene Segmentinhalt auch in den zugehörigen Segmentspeicher des Segments lt. _CURSEG kopiert. Das heißt, die betroffenen Segmente müssen vorab bereits deklariert sein (MAP, USES oder implizit durch Feldangaben). Der Zugriff entspricht dem dynamischen

SELECT (name)

Bsp.:
```
USES DBSEG1 DBSEG2 DBSEG3
DBD DBSEGDB
SELECT DBSEGDB MULT        < befüllt sowohl Speicher DBSEGDB als auch den des
    SHOW _CURSEG           betroffenen Segmentes
ENDSEL
```

Problem: Sind im verwendeten PSB für diese Datenbank DBSEGDB noch weitere Segmente lesesensitiv, so würde beim ersten Zugriff auf eines derselben das Programm abstürzen, weil kein expliziter Segmentspeicher dafür allokatiert wurde.

ALTER segment <EXIT <=> NONE/exitname>
<NOEXIT>
<NAME>
<SORT/NOSORT>

Änderung von Segmenteigenschaften nach deren Laden.

segment Strukturmember, das geändert werden soll

<EXIT <=> > Name des standardmäßig parametrisierten Exits, der bei allen Segmentzugriffen sowohl als Vor- als auch als Nachexit in Abhängigkeit von Zugriff und Status aufgerufen wird.

<EXIT NONE / NOEXIT>
eine bereits angegebene Exitverwendung wird unterbunden.

<NAME=> Segmentname für den Aufbau der SSA im IMS, wenn von ´segment´ verschieden.

<SORT> Default, DB ist aufsteigend sortiert

<NOSORT> DB ist nicht nach dem Primärschlüssel aufsteigend sortiert (nur bei Rootsegmenten relevant)

II DIE TESTHILFEN

Zum Testen der Steuerkommandos stellt das Programm einen TRACE-Befehl in vier Versionen zur Verfügung:

TRACE (ohne Zusatz) Listen der Datenbanken im PSB und den PSB-Namen
Listen der geladenen PO- Feldstrukturtabellen bzw der ge MAPten Tabellen.
Listen der definierten Variablen mit ihren Attributen und der Konstanten mit Attributen und nhalt.
Zur Laufzeit: keine Ausgabe.

TRACE FLOW Zusätzlich zu TRACE: Listen des mit den Konditionssprüngen und den Blockleveln ergänzten Kommandofiles, nach Level strukturiert.
Zur Laufzeit: Ausgabe der aktuellen Zeilennummer '<nnnn>'.

TRACE FULL Zusätzlich zu TRACE FLOW: Listen der funktionalen Auflösung der Kommandozeilen mit den jeweiligen Operanden. Die Laufzeitfunktionen heißen immer #XXXX, z.B #LET, od. #IF.
Zur Laufzeit: Ausgabe der Laufzeitfunkfunktionen nach der Zeilennummer aus TRACE:
<nnnn>.#XXXX<.#YYYY<.#..>>

TRACE HEX Zusätzlich zu TRACE FULL:
Zur Laufzeit: Namentliche Ausgabe aller Operanden 2 der Laufzeitfunktionen hexadzimal im DUMP - Format und als Character-String.

TRACE SCAN Wie TRACE FULL, jedoch keine Durchführung des Kompilats, daher ohne Laufzeit-funktion. Zur Syntax- und Abarbeitungsprüfung des Kommandofiles.

Wird der TRACE - Befehl im Programmvorlauf vor dem ersten ausführbaren Kommando gesetzt, dann ist er selbst ein nicht ausführbarer Befehl.

Wenn nur bestimmte Statements, von deren korrektem Arbeiten man sich überzeugen will, von TRACE bearbeitet werden sollen, so kann man an beliebiger Stelle im Programm den TRACE Befehl setzen und ihn nach den interessierenden Statements mit NOTRACE wieder deaktivieren.

Die Trace-Laufzeitfunktionen werden nur ausgeführt, wenn nach dem Kompilieren der TRACE aktiv ist.

Bei Originaldurchführung den TRACE - Befehl unbedingt herausnehmen oder noppen, weil die entsprechenden Laufzeitbefehle dann gar nicht generiert werden (jeweils ein TRACE - Befehl pro normalem Durchführungsbefehl = fast doppelt so langsam!) und die SYSPRINT-Aufbereitungen Fresser sind, die außerdem wirklich viel IO produzieren.

Der Original-Input sowie die Zähler und die Timestamps werden ohnehin immer gelistet.

NOTRACE

Deaktiviert den vorausgegangenen TRACE Befehl.
Damit kann der TRACE - Scope auf bestimmte Statements eingegrenzt werden, um problematische Programmteile zu untersuchen oder zur hexadezimalen Aufbereitung (nach TRACE HEX) bestimmter Variablen oder Segmentfelder.

SHOW segment / variable / (variable)

Ausgabe kompletter Segmentinhalt oder Variableninhalt auf SYSPRINT

segment - Segmentname lt. Include

variable - Variable lt. Definition oder Include

(variable**)** - dynamisch substituierte Variable/Segment

Der Inhalt von variable kann auch in der Form segment.feld vorliegen.
Der SHOW - Befehl kann sowohl auf Datenbanksegmente wie auch auf einzelne Variable angewandt werden. Die Ausgabe erfolgt unformatiert (keine Aufbereitung nach Feldattributen) als Character-String in der maximalen Segmentlänge oder der definierten Länge der Variablen auf SYSPRINT.

III UNTERPROGRAMME UND EXTERNE CALLS

PROC procname

interne Unterprogrammdeklaration:

Alle nachfolgenden Kommandos werden dem mit procname genannten Unterprogramm zugeordnet. Diese Befehle werden nach Unterprogrammaufruf mittels DO - Befehl abgearbeitet. Sie entsprechen in Syntax und Ergebnis den Kommandos aus dem Direktstream. Jedes Unterprogramm muss mit ENDPROC abgeschlossen werden und kann jederzeit mittels RETURN verlassen werden.

Lokale Variable existieren nicht, alle Variablen, auch wenn sie nur in Unterprogrammen Verwendung finden, müssen global und am Anfang des Programmes definiert werden, sonst Abbruch der Übersetzung mit Fehlermeldung.

Von Unterprogrammen können beliebige weitere Unterprogramme aufgerufen werden, solange die tiefste bei der Implementierung angegebene Schachtelungstiefe (= Summe aller gleichzeitig aktiven SELECT/IF/LOOP/DO - Gruppen) nicht überschritten wird (Default = 50). Auch rekursive Aufrufe sind gestattet, durch das Fehlen lokaler Variablen aber mit Bedacht anzuwenden. Eventuell kann man sich hierbei mit dem PUSH / POP – Mechanismus behelfen.

Die maximale Anzahl der definierbaren Unterprozeduren wird bei der Implementierung angegeben, Default = 20.

Der bedingte Rücksprung aus Unterprozeduren ist die effizienteste Methode zur Unterbrechung der Abarbeitungsreihenfolge bei komplexeren Abfragestrukturen.

LOAD callnam PLI/ASM/COB < <DSN=/DATASET=/DA=dataset.name> /
<DDN=/DD=DD-Statement aus JCL> >
<TAB>

Deklaration externes <Unter>-Programm / Tabelle:

<TAB> - zum Laden von Tabellen, an das Hauptprogramm wird die Adresse des Beginns des Modules retourniert und nicht der (eventuell an anderer Stelle befindliche) Entry.

Externe Prozeduren werden über die STEPLIB geladen, wenn keine andere Ladebibliothek via Dataset oder DD- Paramter angegeben ist. Wegen der unterschiedlichen Parametrisierung ist die Sprache des zu ladenden Programms zwingend anzugeben.

Externe Unterprogramme kommunizieren mit dem aufrufenden Hauptprogramm IMSQL über Parameter oder über die Handshake-Macros IMSQLE und IMSQLI.

IV DIE DATENZUGRIFFE

SELECT FILE filnr / <FILE> filnam /
segment<(n)> / (variable)
<MULT/MULTIPLE>
<HOLD>

Mit dem Select Kommando werden alle Lesezugriffe auf die Datenbanken, sowohl DB wie auch GSAM durchgeführt. Ist der Lesezugriff nicht erfolgreich, wird zum entsprechenden ENDSEL - Statement verzweigt.

FILE filnr Filenummer der GSAM-DB von der gelesen werden soll. Diese muss im Vorlauf deklariert worden sein. filnr ist nur mit dem Keywort FILE gültig.

<FILE> filnam
Datenbankname der GSAM-DB, DD-Name des OS- oder VSAM- Files. Das Keywort FILE ist wahlfrei.

segment Name des Segmentes, das gelesen werden soll, wie via USES oder MAP generiert.

Ist MULT od. MULTIPLE spezifiziert so wird unqualifiziert zugegriffen, wobei mit der WHERE - Anweisung (s. dort) der Zugriffsbereich definiert werden kann, was beim Lesen zu einem Suchzugriff führt. Segmente werden einfach sequentiell gelesen, bis der Reasoncode des DB-Zugriffes ungleich ' ' wird (entsprechender Rücksprung vom korrespondierenden ENDSEL).

Ist MULT od. MULTIPLE nicht angegeben, so wird ein Direktzugriff eingeleitet, der ein entsprechendes Setzen der Segmentkeyfelder mittels LET - Kommando vor Aufruf des SELECT-Statements voraussetzt. Ist der Segment-Key nur zum Teil bekannt haben dem SELECT-Statement entsprechende WHERE-Anweisungen zu folgen.

<(n)> **(i)** Occurence der namensgleichen Datenbank im PSB. Mehrfache PCBs auf eine Datenbank werden durch unterschiedliche Occcurencies unterschieden

z.B.: PSB TESTPSB mit 2 PCBS auf EINEDB

```
SELECT ROOT MULT
  LET ROOT(2).KEY = ROOT.KEY
  SELECT ROOT(2)
    ...
```

Besser und leichter lesbar, weil Verwechslungen mit Feld-Arays hintangehalten werden, ist es allerdings mittels MAP (siehe dort) eine Segmentbeschreibung unter gänzlich neuem Namen für die 2. (n-te) Occurence zu generieren:

```
USES ROOT
MAP ROOT2 NAME=ROOT DB=EINEDB KEY=KEY MEM=ROOT OCC=2
SELECT ROOT MULT
  LET ROOT2.KEY = ROOT.KEY
  SELECT ROOT2
    ...
```

Achtung: Occurency 1 (n = 1) ist nicht unterstützt.

(variable**)** Datenbanksegmente und Files können auch dynamisch gelesen werden, das heißt, der Segmentname wird aus einer Variablen genommen (muss eine Charactervariable mit mindestens 8 Byte Länge sein), die zur Laufzeit einen gültigen Segmentnamen beinhaltet. Da diese Segmente im Vorhinein geladen werden müssen, sind sie am Programmbeginn mittels der USES - Anweisung bekanntzugeben. Die Klammern um den Variablennamen sind unbedingt nötig, damit das Programm die Variable durch ihren Inhalt ersetzt. Sinnvoll zum Laden / Entladen von Konten oder Kontenteilen, bzw .verarbeiten mehrerer gleicher Files.

MULT/MULTIPLE

Lesen bei GSAM-DB/OS-File bis EOF
Lesen DB mit GN bis Statuscode ungleich ' ' d.h: LOOP bis EOF od SC nicht ' ' über alle Statements zwischen SELECT und dem ENDSEL auf gleichem Level lt. Randbedingung.

HOLD (i)

Get Hold Unique/Next auf Segment verhindert zusaetzliches internes GHU vor DELETE und REPLACE - Zugriffen. Oft die einzige Möglichkeit, um bei KEYlosen Segmenten die Positionierung zu halten.

Hinweis:
Dependentsegmente von vollqualifizierten Roots können direkt, ohne vorausgehendes Lesen des Parent bearbeitet werden, wenn nur der Parent Key angegeben wird. Der Inhalt des Parents ist dann jedoch nicht verfügbar. Beispiel:

```
LET ROOT.KEY = ROOTKEY
LET DEPENDENT.KEY = DEPKEY
SELECT DEPENDENT
```

REPLACE segment / (variable)

DELETE segment / (variable)

Name des Segmentes, das Replaced/Deleted werden soll. (i) Beim Kodieren eines REPLACE od. DELETE - Befehls wird das Segment vor dem Zugriff noch einmal mit GHU (Get Hold Unique) gelesen (redundant), wenn nicht HOLD beim Lesezugriff kodiert wurde.

(variable**)**

Analog zum dynamischen SELECT - Statement können alle Datenbankzugriffe auf den Inhalt einer Variablen bezogen werden.

INSERT segment / (variable)

segment

Name des Segmentes, das Inserted werden soll. Da vollqualifiziert positioniert wird, ist zuvor kein Lesezugriff notwendig, da das Programm jedoch bei Duplicate Insert abbricht, empfehlenswert. Auf richtige Wahl des neuen Keys achten!

(variable**)**

Insert eines dynamisch zu substituierenden Segmentes.

Bei REPLACE, DELETE, INSERT ist das Programm im Falle eines Status Codes nicht ' ' im ERROR Status. Um das zu verhindern, muss die Verträglichkeit des jeweiligen Befehles mit der Datenbank im Voraus sichergestellt werden.

LIST filnr/gsambdb/OS-File

entspricht INSERT .....

filnr Filenummer der GSAM-DB auf die geschrieben werden soll. Diese muss im Voraus deklariert worden sein, Proc-Option ist 'LS'.

gsamdbd
GSAM - Datenbankname /OS-FIle auf die geschrieben werden soll. Diese müssen im Voraus deklariert worden sein. Der Outputbereich wird in der Länge der im entsprechenden DCB angegebenen Satzlänge ausgegeben. Alle Feldinhalte der nach der Filedefinition kodierten Output-Variablen müssen zu diesem Zeitpunkt mittels LET zugewiesen worden sein, andernfalls wird das Ergebnis wohl nicht ganz befriedigen.

SEARCH segment <MULT/MULTIPLE>

Mit dem Search Kommando wird ein Zugriff auf die intern via STORE abgelegten Segmentlisten erreicht, analog dem SELECT - Kommando. Es können nur ganze Segmente (= beliebige Strukturen aus MAP, USES, FILE, DBD oder impliziter Deklaration bearbeitet werden (siehe STORE).

Vorteil: keine zusätzliche physischen IOs, parallele Datenverfügbarkeit, keine Positionsänderung der DB, erheblich schneller als via IMS.

segment Name des Segmentes, das vorher geSTOREd, aus der Kette in den allgemeinen Segment-Buffer gelesen werden soll. Die Daten werden bei erfolgreichem SEARCH immer im Originalsegment abegelegt und werden in den gemeinsamen IO-Buffer gestellt.

Ist MULT oder MULTIPLE spezifiziert, so wird unqualifiziert zugegriffen, d.h. die nächste gleichnamige Kopie aus der STORE-Kette wird in den gemeinsamen IO-Buffer gestellt.

Bei fehlendem MULT wird ein Direktzugriff eingeleitet, der zum Zeitpunkt des SEARCH im IO-Bereich befindliche Segmentkey (falls gekeyed) dient zur Identifizierung des gewünschten Speicherelementes.

Blockung und Return-Code-Handling analog SELECT.

MULT/ MULTIPLE get next analog SELECT, Repositionierung siehe FIRST

Bsp.:

```
SELECT ROOT
       SELECT DEPENDENT MULT
       STORE DEPENDENT                  < einspeichern in RAM-Liste
       ...                              < erste Segmentverarbeitung
ENDSEL
SEARCH DPENDENT MULT                    <noch einmal alle verarbeiten
       ...                              < Positionierung nach STORE immer
ENDSEARCH                               auf den Listkopf
...
LET DEPENDENT.KEY = MYKEY
SEARCH DEPENDENT                        < get-unique lt. KEY
       ...
ENDSEARCH
FIRST DEPENDENT                         < repositionieren auf Listkopf
SEARCH DEPENDENT MULT                   < und noch einmal alle durch
       ...
ENDSRCH
RELEASE DEPENDENT                       < freigeben und löschen d. Speichers
...
ENDSEL
```

Die Funktionen SEARCH, STORE, RESTORE und FIRST sind auch als PLI-Makros für gefetchte PLI-Subroutinen vefügbar, siehe Handshake im Anhang

V DIE SYSTEM ZUGRIFFE

CHECK <ALWAYS/n> (i)

ALWAYS
unbedingter Checkpoint an dieser Stelle

n nach n-Durchläufen wird ein Checkpoint gezogen. Bei Weglassen dieses Parameters wird der Checkpoint nur nach Maßgabe der Ressourcenauslastung an dieser Stelle gezogen.

Erzwungener Checkpoint an speziellen Stellen im Programm. Bei mehrfachen Verarbeitungsblöcken kann die Checkpointschreibung an speziellen SYNC-Points wünschenswert sein. Bei Codierung eines (oder mehrerer) CHECK - Statements werden Checkpoints nur an diesen Stellen gezogen, in Abwesenheit des CHECK - Statements erledigt das Programm die Checkpointschreibung selbsttätig.

NOCHECK
Checkpoints werden generell unterbunden.

SYNC <ALWAYS/n> (i)

ALWAYS
unbedingter Syncpoint an dieser Stelle

n nach n-Durchläufen wird ein Syncpoint gezogen. Bei Weglassen dieses Parameters wird der Syncpoint nur nach Maßgabe der Ressourcenauslastung an dieser Stelle gezogen.

Bewußter Syncpoint an speziellen Stellen im Programm. Bei mehrfachen Verarbeitungsblöcken kann die Syncpointschreibung an speziellen - SYNC-Points wünschenswert sein. Bei Codierung eines (oder mehrerer) SYNC - Statements werden Syncpoints nur an diesen Stellen gezogen. Zum Umschalten auf Syncpointschreibung ist die Default-mäßige Checkpointschreibung unbedingt via NOCHECK (i) abzudrehen!

Achtung:
Bei Verwendung von FAST-PATH-DBs Ist das Programm nach Durchführung mit SYNC ohne Checkpoint im Fehlerstatus!

ROLB (i)
erzwungenes Rollback (Backout aller IMS-DB-Veränderungen) seit dem letzten Checkpoint. Generelles Backout ist via NOUPDATE viel bequemer zu erreichen.

OPEN filnam (m)

filnam Name einer OS-Datei die vorher mit Option NOOPEN deklariert wurde

CLOSE filnam (m)

filnam Name einer OS-Datei ohne NOOPEN Attribut oder mit NOOPEN und erfolgtem OPEN.

Gezieltes Schließen einer OS-/VSAM-Datei. Am Ende des Programms werden sämtliche offenen Dateien geschlossen, unabhängig davon, ob das OPEN automatisch, oder nach NOOPEN-Deklaration und OPEN-Anweisung manuell erfolgte.

Bei Verwendung von Lademodulen mit Inputsteuerung (z.B. IDCAMS), die vom Programm IMSQL befüllt werden, oder von solchen, deren Output vom IMSQL interpretiert werden soll, ist eine explizite OPEN-CLOSE - Logik implementiert. Diese Dateien müssen jedoch im Vorhinein das Attribut NOOPEN erhalten, um die automatische Unterstützung zu verhindern.

VI BEREICHSSTEUERUNG

Die Bereichssteuerung funktioniert bei allen Segmenten, mit sensitiven KEY - Feldern, wobei auch eine Direktzuweisung der Segmentkeys erfolgen kann.

WHERE segment.feld operator name2

segment.feld
Keyfeld, oder bei zusammengesetzten Keys einzeln deren Komponenten.

operator Vergleichsoperator:
GE / >= : operator 2 wird in den Segmentbereich übertragen.
LE / <= : operator 2 wird in den Endkey übertragen
EQ / = : operator 2 wird in den Segmentbereich und in den Endkey übertr.
LT,GT,NE nicht unterstützt.

name2 Bezeichner, der bereits als Variable deklariert worden ist, Segmentfeld oder Konstante.

BSP:
```
*Suchen Kontenbereiche
SELECT KONTEN MULTIPLE
      WHERE KONTEN.BLZ = 12345                  (Key Teil 1)
      WHERE KONTEN.KONTO >= 50000000000         (Key Teil 2)
      WHERE KONTEN.KONTO <= 600000000
            ...
ENDSEL dies ist auch ein kommentar
```

Das SELECT Statement muss bei Bereichsangaben nicht die MULT Option besitzen, ist der Key nur zum Teil bekannt, kann hiermit der Zugriffsbereich eingegrenzt werden, das Programm setzt dann einen GET-UNIQUE Zugriff ab.

Die WHERE - Anweisung ist selbst keine ausführbare Anweisung. Da sie hinter dem entsprechenden SELECT segment <MULT> kodiert werden muss, würde zwar die Endebedingung als IF - Zweig funktionieren, nicht aber das Setzen des Suchkeys, wenn dieser erst zur Laufzeit bekannt ist.

Aus diesem Grund inserted diese Anweisung die entsprechenden Laufzeitbefehle in den Kommandofluss der vorgelagerten SELECT - Anweisung, mit dem Zusatzbefehl, die ursprüngliche Verkettung nach erfolgter Durchführung wieder herzustellen. Das Setzen der Suchzugriffe eliminiert sich selbst nach der Abarbeitung. Es kommt daher zur Laufzeit zu keinen Verzögerungen wegen der Statusprüfung.

- **nur IMS**:
  Der Aufbau der SSA(s) erfolgt dynamisch bei jedem Datenbankzugriff wobei das Setzen der Bereichsgrenzen nur einmal (beim ersten Durchlauf der SELECT - Schleife) geschieht. Wird der Segmentkey innerhalb der Schleife überschrieben, so ist das Ergebnis nicht vorhersehbar.

- **nur VSAM**:
  Der Zugriff erfolgt >= KEY aus Segment.

- Achtung:
  Zwischen SELECT – und zugehörigen WHERE - Statements dürfen keine anderen Anweisungen kodiert werden! Arithmetische Ausdrücke in der WHERE - Anweisung sind nicht unterstützt.

VII DATENMANIPULATION

LET name1 = oper2 <oprat1 oper3 <oprat2 oper4 <...>>>
LET name1 = oper2 BY NAME/BYNAME
LET (name1) = oper2 <oprat1 oper3 <oprat2 oper4 <...>>>

name1 - Zielfeld

(name1**)** name1 beinhaltet den Feldnamen d. Zielfeldes kann auch in der Form segment.name vorliegen

oprat1,oprat2,..
- Operatoren die auf den Datentyp in name1 angewandt werden können. möglich sind:
'+' - addieren
'-' - subtrahieren
'*' - multiplizieren
'/' - dividieren

oper3,oper4,..
Sourcefelder der Operation

BY NAME/BYNAME
nur auf Segmente (MAP oder USES oder implizit) anwendbar. Für jedes gleichnamige Feld der beiden Strukturen (sowohl in Bezeichnung wie im Index) wird eine Wertzuweisung generiert. siehe auch Beispiel

Alle Operanden werden vor der Operation in den Datentyp des Zielfeldes umgewandelt. Das kann zu Fehlern führen.

BSP.:
```
VAR ZIEL FIXED(5)
 ...
LET ZIEL = 3.5 * 4
 ...
ZIEL
```
ist in diesem Fall 12, nicht 14 !

Da das Problem im Vorhinein bekannt ist, kann durch entsprechende Wahl eines Zwischenfeldes das richtige Ergebnis erreicht werden.

BSP.:
```
VAR ZIEL FIXED(5)
VAR ZIEL1 FIXED(7,1)
...
LET ZIEL1 = 3.5 * 4
LET ZIEL = ZIEL1
...
```

Die Instruktion muss auf einer Zeile abgeschlossen sein. Wie eingangs erwähnt, arbeitet die Abarbeitung einer gestaffelten Zuweisung strikt von links nach rechts, ohne auf Operationsprioritäten Rücksicht zu nehmen !!

Anmerkung: Bei Bitfeldern wird immer ein Overlayfeld über das entsprechende Zielfeld generiert mit Namen BITOV#nn mit nn = laufende Nummer, in dem auch die Information über die Bitposition gespeichert wird.

PUSH name1

name1 - SOURCEfeld
Der Inhalt der Variablen name1 wird gestackt, dieser selbst jedoch nicht verändert.
Wiederhergestellt werden diese Werte mittels

POP name1

name1 - ZIELfeld
Der Inhalt der Variablen name1 wird wieder von Stacker geholt.

Achtung: Es ist unbedingt auf die richtige Reihenfolge von PUSH und POP zu achten, das System folgt streng der FILO-Regel (First In Last Out).

```
BSP:  VAR A CHAR(8)
VAR B FIXED(3)
VAR C CHAR(1)
LET A = 'TEXT'
LET B = 4
LET C = 'B'
PUSH A
LET A = 'SONNE'
PUSH A
PUSH (C)      Inhalt von C = 'B': entspricht PUSH B
POP A         (gepackt 4 + ' ', ursprünglicher Inhalt von B)
POP B         'SON' - beim nächsten Zugriff auf diese Feld kommt es unweigerlich zum 0C7.
```

STORE segment

segment Struktur/Segment aus MAP,USES,...

Der aktuelle Segmentinhalt wird in eine zur Laufzeit allokatierte Speicherliste übertragen und steht bis zum entspr. RELEASE allen Prozessen innerhalb des REP - Scopes gleichzeitig zur Verfügung. Mittels SEARCH können die einzelnen abgelegten Segmentversionen jederzeit über deren Key oder sequentiell erneut in den Segment-IO-Bereich gestellt werden, ohne noch einmal von der Datenbank, vom File oder sonst irgendwie eingelesen oder erzeugt werden zu müssen.

Die zuletzt geSTOREte Version wird immer als erste in die Liste eingefügt, was einerseits zu einer Umkehrung der ursprünglichen Sortierreihenfolge führt (Last In, First Element), andererseits jedoch dem nachfolgenden seq. SEARCH-zugriff die sofortige Abarbeitung ohne vorausgehendes FIRST ermöglicht. Manchmal ist auch nur die automatische Sortierumkehrung erwünscht.

Bei gekeytem SEARCH-zugriff (vorausbefülllen des Keys im Original-Sgmentbuffer) und Zugriff ohne MULT ist die Reihenfolge ohnehin irrelevant.
STORE verändert nicht den originalen Segmentinhalt.
Beispiel siehe SEARCH

RESTORE segment

segment Struktur/Segment aus MAP, USES,...

Das mittels STORE gespeicherte Segment mit dem aktuelle Key lt. Allgem. IO-Bereich wird gesucht und mit dem Inhalt des IO-Bereiches überschrieben. Wird das Segment nicht in der Speicherliste gefunden, so wird neu geSTOREd. Bei nicht-gekeyten Segmenten gilt das aktuelle gespeicherte Segment lt. letztem SEARCH. Der Konditioncode (via IF No abfragbar) wird anhand des vorausgehenden Zugriffserfolges gesetzt.
RESTORE verändert nicht den originalen Segmentinhalt.

FIRST segment

segment Struktur/Segment aus MAP,USES,...

Bei bestehender Segment-Clone-Kette nach STORE wird der segmentspezifische Positiosnzeiger auf das erste Element der Liste gesetzt. Das Originalsegment bleibt unverändert. Ein nachfolgender unqualifizierter Lesezugriff liefert das erste Element der Kette.
Beispiel siehe SEARCH

RELEASE segment

segment Struktur/Segment aus MAP,USES,...

Sämtliche existierende Versionen der Struktur segment werden aus dem Speicher gelöscht, der originale Segmentinhalt bleibt unverändert.

Unbedingt vor dem neuerlichen Einspeichern einer unabhängigen Datengruppe gleichen Namens anwenden, sonst werden die alten Inhalte noch einmal verarbeitet. Außerdem führen die wiederholten Speicheranforderungen irgendwann unweigerlich zum Absturz des Programms wegen Speiecherüberlauf.

VIII KONDITIONALE

IF / AND / OR name1 vgop oper2 <oprat1 oper3 <oprat2 oper4 <...>>>
oder < , oper3 < , oper4 < , ...>>>

name1 - Zielvergleichsfeld oder (name1) dynamisch

vgop - Vergleichsoperator möglich sind:
LT / < - kleiner
LE / <= - kleiner od. gleich
NE / ^= - ungleich
EQ / = - gleich
GE / >= - grösser od. gleich
GT / > - grösser

oprat1,oprat2,.. - Operatoren die auf den Datentyp in name1 angewandt werden können, möglich sind:
'+' - addieren
'-' - subtrahieren
'*' - multiplizieren
'/' - dividieren
',' - Mehrfachvergleich (SET)

oper3,oper4,.. - Sourcefelder der Operation oder (oper3..) bei dynamischer Substitution

Trifft der Vergleich zu, so wird mit der Zeile fortgefahren, die direkt hinter der letzten Konditionalzeile steht, ansonsten bei den Zeilen OR, ELSE oder ENDIF, je nachdem was als nächstes kommt.

Zu AND und OR: Da es auch bei den Vergleichsoperationen keine Klammerung gibt, muss bei verschachtelten Abfragen auf die richtige Kombination geachtet werden:

BSP

```
                                    bei JA   bei NEIN (zu Zeile)
      000006      IF     ...        7        8
      000007      AND    ...       11        8
      000008      OR     ...        9       16
      000009      AND    ...       10       16
      000010      AND    ...       11       16
      000011      LET    ...
                         ...
      000016      ENDIF
      ...
```

ELSE

Ist ein ELSE - Zweig kodiert, so ist diese Zeile die Zieladresse bei missglückten Vergleichen der Zeilen IF, AND und/oder OR,.

```
                                  bei JA   bei NEIN (zu Zeile)
000006      IF     ...             7        8
000007      AND    ...            11        8
000008      OR     ...             9       16
000009      AND    ...            10       16
000010      AND    ...            11       16
000011      LET    ...
                   ...
000016      ELSE   ...            20       17
000017             SELECT ..
                   ...
000020      ENDIF                 ...
```

IF / AND / OR NO / YES

NO - Die Kommandos innerhalb der IF - Bedingung werden durchgeführt, wenn die dem IF - Statement vorausgehende Konditionsprüfung den Wert FALSE lieferte.

YES - Die Kommandos innerhalb der IF - Bedingung werden durchgeführt, wenn die dem IF - Statement vorausgehende Konditionsprüfung den Wert TRUE lieferte.

Die beiden Keywörter YES und NO beziehen sich auf den Condition-Code, der beim Eintritt in die IF - Bedingung herrscht. Dieser CC ist somit das einzig existierende System - Konditional, und kann auch nur gelesen werden. Gesetzt wird sie immer durch die Konditionalsaktionen: IF, SELECT, LOOP, SEARCH. Sinnvoll ist die Abfrage des Konditionals sicher nur nach einem Select – oder Search- Statement, da bei IF - Konstruktionen ohnehin ein ELSE - Zweig existiert, und nach einem LOOP der Konditional immer auf FALSE steht, bzw. innerhalb der LOOP - Schleife auf TRUE.

BSP.:
```
...
LET SEGMENT.KEY = 300
SELECT SEGMENT
      LET SEGMENT....
      ...
      REPLACE SEGMENT
ENDSEL
IF NO
      LET SEGMENT.....
      ...
      INSERT SEGMENT
ENDIF
...
```

SWITCH **WHEN name1 vgop oper2 <oprat1 oper3 <oprat2 oper4 <...>>>**
oder < **, oper3 < , oper4 < , ...>>>**
< **AND name1 vgop oper2 <oprat1 oper3 <oprat2 oper4 <...>>>**
oder < **, oper3 < , oper4 < , ...>>> >**
<<AND ... >>
Command 1
...
WHEN
...
< **WHEN NONE** > bei Nicht-Zutreffen einer der vorausgegangenen Bedingungen
ENDSWITCH / ENDSW

- Parametrisierung wie bei IF

Trifft ein WHEN-Vergleich zu, so wird mit der Zeile fortgefahren, die direkt hinter der letzten Konditionalzeile steht und beim nächsten WHEN wird zum ENDSWITCH - Statement verzweigt, ansonsten wird das nächste WHEN (wenn vorhanden) direkt angesprungen.
Hinweis: Bei Kodierung einer Switch-Gruppe wird der interne Programmlevelzähler aus programmtechnischen Gruenden um 2 erhöht.

BSP.:		zu Zeile:	bei JA	bei NEIN
000005	SWITCH		6	
000006	WHEN ...		7	9
000007	AND ...		8	9
000008		LET ...	16	
000009	WHEN ...		10	11
000010		LET ...	16	
000011	WHEN NONE			
		...		
000016	ENDSWITCH			

Spezialanwendung END

Eine SWITCH - Gruppe kann jederzeit mit einer END - Anweisung verlassen werden, End bewirkt innerhalb der Switch-Gruppe keinen Programmabbruch!

BSP:
```
SWITCH
 WHEN ...
 AND  ...
 LET  ...
 LET  ...
 LET  ...
 IF   ...
      END
 ENDIF
 LET  ...
 LET  ...
 WHEN       ...
 LET  ...
 WHEN NONE
 ...
ENDSWITCH
```

IX PROGRAMMFLUSS - STEUERUNG

LOOP name1

name1 - Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert.

Der Loop Befehl entspricht einer BCT Schleife der Variablen name1, vergleichbar dem WHILE in PLI. Ist der Inhalt der Variablen name1 größer als 0, so werden die Kommandos, die zwischen dem LOOP - Befehl und dem entsprechenden ENDLOOP geschrieben wurden durchgeführt und die Variable um 1 erniedrigt.

Als Variablenformat sind Dezimal-, Binär- und Characterfelder zugelassen, sind die letzteren allerdings nicht numerisch, dann befindet sich das Programm in ERROR.

DO procname

procname - Am Anfang des Programmes definiertes Unterprogramm. Von jeder beliebigen Stelle kann in ein Unterprogram verzweigt werden. Alle Variablen behalten beim Eintritt ihre Werte und werden innerhalb der Subroutine per absoluter Adresse referenziert.

Rückkehr ins Hauptprogramm (oder in ein vorgelagertes Unterprogramm) erfolgt mittels RETURN oder beim ENDPROC - Statement automatisch.

Bei Unterprogrammen handelt es sich immer um Prozeduren und nicht um Funktionen, sie können daher keine Werte retournieren.

RETURN

Von jeder beliebigen Stelle kann aus dem Unterprogramm zur Anweisung die im Aufrufer direkt auf die DO-Anweisung folgt zurückgesprungen werden. Die Rückkehr ins Hauptprogramm (oder in ein vorgelagertes Unterprogramm) erfolgt beim ENDPROC - Statement automatisch.

CALL callname<(parm1<,parm2<,...<,parm8>>>>)>

callname - Am Anfang des Programmes via LOAD-Befehl geladenes externes ausführbares Programm

parm1..parm8 - gültige Variablennamen

Ein etwaiger Return-Code aus dem Unterprogramm (Register 15) wird nicht erkannt.

Mittels der Handshake-Makros IMSQLE und IMSQLI haben geladene PLI-Unterprogramme direkten Zugriff auf alle internen Variablen und Segmentbereiche des aufrufenden IMSQL.

Achtung PLI: Die Parameterübergabe innerhalb des aufgerufenen Programms funktioniert nur bei ENTRY PLICALLA gelinkten Modulen und bei Parameterdeklaration als CHAR(1) mit dem echten Parameter als Overlay:

```
SUBPGM: PROC(A,B,C,..) OPT...
DCL (A,B,C) CHAR(1);
DCL 1 PARMA BASED(ADDR(A)),
 ...
DCL PARMB PTR BASED(ADDR(b));
DCL PARMC ... BASED(ADDR(C));
 ...
```

GOTO zeilnr/name1

zeilnr - Absolute Inputzeilennummer incl. aller Kommentare und Deklarationen

name1 - Variable, die in eine binaerzahl convertiert werden kann. Sprung auf Zeilennummer lt name1

Die meisten Programmflussprobleme sollten eigentlich mit den IF, SELECT,DO und CALL-Anweisungen abdeckbar sein. Sollte jemandem jedoch wirklich nichts mehr einfallen, dann ist dieses Kommando zumindest implementiert. Problematisch ist die Anwendung vor allem wenn man als Zieladresse die absolute Zeilennummer des Kommandofiles setzt. Diese ändert sich mit jeder Änderung am Steuerfile. Besser ist das Setzen der Sprungadresse mittels der LABEL – Anweisung.

Achtung: Die vorausgegangenen Condition - Codes und blockspezifischen Randbedingungen bleiben gesetzt, was u.U. zu unbeabsichtigten (Re-)Aktionen des Programms führen kann.

LABEL name1

name1 - Variable, die in eine binaerzahl convertiert werden kann.

Die aktuelle Zeilennummer des Commands wird der Variablen name bereits während der Initialisierungsphase zugewiesen. Eine spätere Änderung des Inhaltes bewirkt somit eine Zieländerung der korrespondierenden GOTO - Befehle.

BSP:
```
VAR X FIXED(3)
...
GOTO X
SHOW 'NIE'   - wird nie angesprungen
LABEL X
SHOW 'IMMER' ...
```

X BLOCK - UND PROGRAMMABSCHLÜSSE

ENDLOOP	- Abschluss einer jeden LOOP - Schleife
ENDSEL	- Abschluss eines jeden SELECT-Blockes
ENDSRCH / ENDSEARCH	- Abschluss eines jeden SEARCH - Blockes
ENDIF	- Abschluss eines jeden IF-Blockes
ENDPROC	- Abschluss einer Unterprogrammdeklaration
ENDSWITCH / ENDSW	- Abschluss einer jeden SWITCH - Gruppe
ENDFILE	- optionaler Abschluss einer Filedeklaration
END	- Reguläres Beenden des Programms, muss zumindest als letztes Statement im Kommandofile kodiert werden.

Jeder SELECT, SEARCH, SWITCH oder IF Block muss mit einem korrespondierenden ENDSEL, ENDSRCH, ENDSWITCH oder ENDIF abgeschlossen werden. Diese Anweisungen markieren die Gültigkeitsbereiche der Condition-Codes und sind daher für den Programmfluss entscheidend.

Nämliches gilt für die LOOP - Schleife, deren Gültigkeitsbereich durch das folgende ENDLOOP beendet wird, unter der Voraussetzung, dass dazwischen keine Blöcke offen bleiben.

Das Ende jeder DO-Gruppe (Unterprogramme) muss mit ENDPROC markiert werden, damit das Programm den eigentlichen Programmstart erkennt. Unterprogramme müssen immer vor dem ersten ausführbaren Befehl des Hauptprogramms kodiert werden.

Die END Anweisung muss jedenfalls am Ende des Steuerfiles gesetzt werden, damit das Programm überhaupt aufhört. Daher übernimmt das Programm diese Aufgabe selbst, wenn die Anweisung fehlt.

Ganz praktisch ist die END Anweisung jedoch auch in einem IF - Scope, weil man dadurch ein zusätzliches konditionsgesteuertes reguläres Beenden des Programms an jeder beliebigen Stelle erzwingen kann.

BSP.:
```
IF KONTO.KAPITAL < 0
      <MESSAGE AUFBEREITEN>
      <LIST>
      END
ENDIF
 ...
```

Bei allen ENDxxx - Anweisungen, bei END alleine sowie bei GOTO nach der Zielzeilennummer, bei LIST nach der Filenummer und bei LOOP nach dem Variablennamen kann ein beliebiger Kommentar angeschrieben werden. Diese Zeilenbereiche werden vom Programm ignoriert.

XI BUILTIN - FUNKTIONEN

LENGTH(name1)

name1 - Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert.

LENGTH(..) ist auch als Pseudovariable zugelassen, z.B.: LENGTH(BEREICH) = 17 + 4

Achtung: Wird bei der Längenzuweisung die ursprüngliche maximale Datenlänge überschritten, kommt es zu Speicherüberschreibungen!

Achtung: Bei der Anwendung auf ein Segment wird dessen Maximallänge retourniert. Zur Abfrage der aktuellen Länge ist das entsprechende Satzängenfeld zu verwenden. Die Änderung einer solchen Länge bewirkt auch die Änderung der Maximallänge des Segmentes, bzw der Schreiblänge eines OS- oder VSAM- Outputbestandes.

INDEX(name1,name2)

name1,name2 - Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert, oder eine Zeichenkonstante.

Alle Daten werden in das Zeichenformat (Char) umgewandelt. Die Funktion liefert den Beginn des Strings name2 im String name1, bzw. 0 bei fehlender Übereinstimmung.

VERIFY(name1,name2)

name1,name2 - Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert, oder Zeichenkonstante.

Alle Daten werden in Zeichenformat (Char) umgewandelt. Die Funktion liefert die Position des ersten Bytes in name1 das nicht in name2 vorkommt.

COUNT(segment/filnr,zugriff)

segment - Gültiger Segmentname oder GSAM/OS-File

filnr - Nummer der GSAM-Datenbank im PSB

zugriff - Zähler der interessierenden Zugriffsart, gültig sind SELECT, REPLACE, DELETE, INSERT und LIST.

Retourniert wird immer im Format DEC FIXED(11);

BSP: `SHOW COUNTER(GNSRTI,REPLACE)`

SCALE(name)

name - Gültiger Variablenname
Retourniert wird im Format BIN FIXED(15) Die Verwendung als Pseudovariabel ist zugelassen.

Wichtig bei MAP, da aus Assemblerstrukturen keine Kommastellen generiert werden können.

```
KOMMA(BETRAG) = 2
```

ABS(name)

name - Gültige Variable, muss Fixed sein!

Liefert den Absolutwert der Variablen name in deren Länge lt. Deklaration.

BSP: `IF ABS(GNSRTI.KAPI) > GRENZWERT ...`

PCB(name)

name - Gültiger Name eines IMS-Segmentes

Liefert die Adresse des PCB der diesem Segment zugeordneten Datenbank.

BSP: CALL PLIPGM (PCB(DBSEG)) - ermöglicht Subroutinen den Zugriff auf IMS-Datenbanken - diese sind allerdings dem Hauptprogramm dann nicht bekannt und werden weder gezählt noch in die Positionierung übernommen.

ADDR(name)

name - gültiger Variablen/Segmentname

Retourniert wird immer im Format BIN FIXED(31) Mit Adressen kann uneingeschränkt gerechnet werden, z.B.: ADDR(a) = ADDR(b(4 + i)) + 5. Die Verwendung als Pseudovariabel ist zugelassen. Ist das Empfangsfeld einer Adressfunktion ein Segment, so werden sämtliche Felder des Segmentes/der Struktur auf die neuen Offsets gestellt. Dies gilt auch für BASED- Segmente.

BSP.:
```
MAP STRUKTUR1 ORG *
MAP STRUKTUR2 ORG STRUKTUR1.VARBER1
LET ADDR(STRUKTUR1) = ADDR(IBER)
```

Hier werden sowohl die Struktur STRUKTUR1 wie auch STRUKTUR2 neu positioniert.

DECNUM(name)

name - gültige (gepackte) Variable

Retourniert die Position des ersten Halbbytes in name, das für gepackte Werte nicht zugelassen ist. Für die Digits 1 bis n-1 sind nur die Werte 0 bis 9, für das Digit n nur die Werte C bis F (hexadezimal) zugelassen.

Ist die Dezimalzahl name korrekt, wird 0, sonst der Wert 1 bis 16 retourniert.

BSP. : `VAR A FIXED(5)`
`SHOW DECNUM(A)` retourniert 0 für z.B. x‘01234C‘ und
4 für z.B. x‘012AB0‘.

SUBSTR(name1,OFFSET/name2<,LÄNGE/name3>)

name1,name2,name3 - Bezeichner, der bereits als Variable oder File-Feld definiert wurde, oder der in der Form segmentname.feld als Teil einer der Datenbanksegmentbeschreibungen existiert, oder In-Stream-Konstante.

OFFSET,name2 - Beginn der Teilzeichenkette in name1 - muss in einen numerischen Wert konvertierbar sein. Ist der Wert <= 0 oder > als die Länge des Originalstrings wird " (nichts) retourniert.

LÄNGE,name3 - wahlfrei, Länge der zu extrahierenden Zeichenkette

SUBSTR kann auch als Pseudovariable verwendet werden, Zuweisungen wie

LET SUBSTR(var1,x,y) = 'TEXT' / var sind unterstützt.

Wichtig !!: Funktionen können nicht geschachtelt werden, Konstrukte wie SUBSTR(var1,LENGTH(INDEX(var2,var3))) sind nicht möglich.

XII SYSTEMVARIABLE

System-Variable sind, wenn nicht anders angegeben, nicht überschreibbar.

System-Variable

_SYSTEM	Inhalt IMS oder z/OS, retourniert den Systemstatus
_SEGINT	Adresse interne Segmentspeichertabelle. Ermöglicht Unterprogrammen den direkten Zugriff auf alle im Hauptprogramm via USES, MAP oder implizitem Gebrauch geladenen Segmentbeschreibungen. Beispiele siehe Anhang
_SEGMAX	Anzahl der generierten Segmente in Tabelle über _SEGINT
_CURSEG	Name des zuletzt via IMS oder IO bearbeiteten Segmentes
_CURSLVL	hierarchischer Level des _CURSEG (nur IMS)
_CURKFB	aktuelle Key-Feedback-Area (nur IMS)
_CURSTC	Aktueller IMS - Status-Code
_CURLINE	Aktuelle Verarbeitungszeile (überschreibbar)
_ISCHKP	‚J' bei erstem Statement nach Check- / Syncpoint, sonst ‚N'
_DATE	heutiges Datum in der Form CHAR(8) JJJJMMTT

XIII Programmeldungen und TRACE - Formate:

Am Kopf der Sysprintmeldungen steht die Programmkennung mit dem Timestamp bei Beginn der Kompilierung in der Form *hhmmssttt*.

Dann folgt das Listing der Input - Statements, so wie sie kodiert wurden. Sollten während des Kompilierens Fehler erkannt worden sein, so werden die entsprechenden Fehlermeldungen in das Listing, der entsprechenden Zeile folgend, eingeschoben, sie enthalten die Zeilennummer und die Art des aufgetretenen Fehlers.

Hierauf folgt bei aktivem TRACE die Tabelle der definierten Variablen, der Konstanten, und der automatisch generierten Bitfeldoverlays, wenn solche angesprochen wurden. Es werden die Namen, die Formate (*C* = CHAR,*B* = BIT,*X* = BIN,*P* = DEC,*L* = Loadmodul,*A* = Adresskonstante), die Länge in Bytes und bei den Konstanten deren Werte gelistet.

Es folgt ab TRACE FLOW die Liste der geladenen DB-Segmentbeschreibungen und eine formatierte Ausgabe der Kommandozeilen, mit dem Inhalt:

Zeilennummer, Block-Level, Sprungziel bei Condition TRUE, Sprungziel bei Condition FALSE, Command

Die Sprungziele sind als Inputzeilennummer zu interpretieren.

Bei TRACE FULL, HEX oder SCAN folgt nun die funktionale Auflösung der Kommandos mit den jeweiligen Operanden, das sind die zur Laufzeit aufgerufenen Funktionen. Diese Tabelle hat die Form:

Zeilennummer, Funktion, Operand_1, Operand_2, Operator

Da die DB - Funktionen keine Operanden nach Art der Variablen brauchen, wird bei diesen das Segment und der Zugriff gelistet, bei FILE - Zugriffen der File - Name und der Zugriff.

Der Operator ist nur bei Vergleichsoperatoren angegeben, bei Indexberechnungen wird die Basisvariable und die Indexvariable angezeigt.

Im Anschluss an dieses Kapitel finden Sie eine Aufstellung der Laufzeitfunktionen mit Funktionsbeschreibung.

Die Kompilierung ist damit abgeschlossen. Es folgt der Timestamp des Laufzeit - Beginns.

War TRACE am Ende des Kommando - Inputs aktiv, so folgt nun bei:

TRACE --- - nichts

TRACE FLOW - alle Nummern der abgearbeiteten Kommandozeilen in der Form
<nnnn><nnnn><....

TRACE FULL - zusätzlich die Namen der aufgerufenen Funktionen <nnnn>.#FU1
.#FU23.#FU12<nnnn>.# .., sowie Art und Ergebnis von DB/OS - Zugriffen,

TRACE HEX - ausserdem die Anzeige des Operanden_2 (Source) hexadezimal und gezont.

```
<nnnn>.#FU1
HEX(BETRAG ) 40404040 40F1F2F3 ' 123'
.#FU23
HEX(ACCU ) 00007971 04304C ' '
```

TRACE SCAN - gar nichts, weil keine Durchführung erfolgt

Am Ende der Abarbeitung folgt wieder ein Timestamp und die Ausgabe der Zähler der Datenbank- und Filezugriffe, die nicht 0 sind.

Wurden also Segmente unqualifiziert angesprochen, für die kein Zähler gelistet wird, dann war kein Zugriff erfolgreich.

Anhang 1 - Beispiele

Anwendung der Kommandos und Programmbeispiele:

Es empfiehlt sich ein PO-Dataset für die Vorlaufkarten anzulegen (LRECL=80,BLKSIZE=3040,FB), da ähnlich gelagerte Probleme häufig auftreten und bei Originaldurchführung nur der entsprechende Member-Name für file SYSCMD angegeben werden muss.

- Beispiels - JCL:

für IMS - Anwendungen:

siehe ROSCOE - Member PAR.§AID-IMS mit PSB AID-IMS
(Testdatenbanken, Originalprocedure z.B.: AID-IMS01)

Beispiel 1:

Lesen alle Rootsegmente GNSRTI auf DEGNKTIX

```
SELECT GNSRTI MULT
ENDSEL
```

Das Programm liefert auf der Testdatenbank folgenden Output:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*              IMSQL - VERSION  4.1                        *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   **** IMSQL :  BEGINN AUFBAU / KOMPILIERUNG 090430  102355084

                  DATUMSKARTE PER               300409

        1               SELECT GNSRTI MULT
        2               ENDSEL
        3               END

                 KOMPILIERUNG BEENDET

   **** IMSQL :  BEGINN VERARBEITUNG 090430  102355241


CHECKPOINT NACH:   GNSRTI      UM   102412913
   SEGMENT: GNSRTI   HEX(RTIKEY  ): '00101C0013016215200CE4E2C4'

                 VERARBEITUNG ABGESCHLOSSEN 090430  102426071

   **** IMSQL :  ARBEITSSTATISTIK


          9.068      "GET     " AUF DEGNKTIX-DB, SEGMENT GNSRTI

*********  PGM  I M S Q L  ENDE ***************
```

Beispiel 2

Lesen GNSRTI/DEGNKTIX vollqualifiziert und Ausgabe Kontobezeichnung + Kontonummer

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*              IMSQL - VERSION  4.1                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   **** IMSQL :  BEGINN AUFBAU / KOMPILIERUNG 090430  104524442

                  DATUMSKARTE PER                300409

         1              LET GNSRTI.MANDNR    = 101
         2              LET GNSRTI.KTONR     = 976504506
         3              LET GNSRTI.WRG       = 'ATS'
         4              SELECT GNSRTI
         5                 SHOW GNSRTI.NAMEK1 GNSRTI.KTONR
         6              ENDSEL
         7              END

                KOMPILIERUNG BEENDET

   **** IMSQL :  BEGINN VERARBEITUNG 090430  104524699


NAMEK1                  MARIO PARRAGH
KTONR                     pµÜ&%

                VERARBEITUNG ABGESCHLOSSEN 090430  104524980

   **** IMSQL :  ARBEITSSTATISTIK


             1      "GET     " AUF DEGNKTIX-DB, SEGMENT GNSRTI

*********  PGM  I M S Q L  ENDE ***************
```

Die Ausgabe mittels SHOW erfolgt ohne Konvertierung im direkten Hexadezimalen Format der Speicherung. Bei Textstellen (NAMEK1) ist das Ergebnis direkt lesbar, bei gepackten Werten (KTONR) muss eine Möglichkeit zur Hexadezimalen Darstellung gewählt werden:

```
_____________________________________________________________________________
NAMEK1                  MARIO PARRAGH
DCDCDF4444444444444444DCDCD4DCDDCCC444444444444444444444444444444444444
514521000000000000000000419960719917800000000000000000000000000000000000
_____________________________________________________________________________
KTONR                     p  &%
DEDDD444444444444444444400960564444444444444444444444444444444444444444
236590000000000000000000007540C000000000000000000000000000000000000000000
_____________________________________________________________________________

444444444444444444444444444444444444444444444444444444444444444444444444
000000000000000000000000000000000000000000000000000000000000000000000000000
_____________________________________________________________________________
                VERARBEITUNG ABGESCHLOSSEN 090430  104524980
444444444444444ECDCDCCCEEDC4CCCCECCDDEECD4FFFFFF44FFFFFFFFF444444444444
000000000000000005591925934570127523836225500904300010452498000000000000
```

Beipiel 3

Lesen Rootsegment (GNSRTVA/DEGNKTVA) vollqualifiziert und alle Zinsensegmente der Randschicht(GNSZIR/DEGNKTVA):

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*             IMSQL - VERSION  4.0                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   **** IMSQL :  BEGINN AUFBAU / KOMPILIERUNG 090430  105422240

                  DATUMSKARTE PER               300409

        1              LET GNSRTV.MANDNR    = 101
        2              LET GNSRTV.KTONR     = 976504506
        3              LET GNSRTV.WRG       = 'ATS'
        4              SELECT GNSRTV
        5                 SELECT GNSZIR MULT
        6                    SHOW GNSZIR
        7                 ENDSEL
        8              ENDSEL
        9              END

                  KOMPILIERUNG BEENDET

   **** IMSQL :  BEGINN VERARBEITUNG 090430  105422522

GNSZIR                   º”1J±1       Ÿ& _q_¿
GNSZIR                   ºÍ1J±3       _ _r_Ú

                  VERARBEITUNG ABGESCHLOSSEN 090430  105424115

   **** IMSQL :  ARBEITSSTATISTIK


              1      "GET     " AUF DEGNKTVA-DB, SEGMENT GNSRTV
              2      "GET     " AUF DEGNKTVA-DB, SEGMENT GNSZIR

*********  PGM  I M S Q L  ENDE ***************
```

Alle Parentage-Segmente einer Segmentkette werden immer vollqualifiziert identifiziert, eine korrekte Zuordnung der Dependents zu ihren Roots ist daher immer gegeben.

Beipiel 4

Join über 2 Roots und Primärkey

```
SELECT GNSRTI MULT
   LET GNSRTV.RTVKEY = GNSRTI.RTIKEY
   SELECT GNSRTV
      SHOW GNSRTV
   ENDSEL
ENDSEL
```

Ist keine Eintragung in der DEGNKTVA Datenbank mit dem Primärkey aus GNSRTI/DEGNKTIX vorhanden, so würde kein Segment beim SHOW angezeigt.

Nun das gleiche Beispiel mit Fileausgabe:

Beispiel 5

```
FILE 2 OUT
   RTIKEY     CHAR(13)
   TEXT       CHAR(20)
*
LET TEXT = ´VORHANDEN´
*
SELECT GNSRTI MULT
   LET GNSRTV.RTVKEY = GNSRTI.RTIKEY
   SELECT GNSRTV
      LET RTIKEY = GNSRTI.RTIKEY
      LIST 2
   ENDSEL
ENDSEL
```

Eine numerische (= logische) Filedeklaration bezieht sich auf die entsprechende n-te GSAM - Datenbank (= IMS - OSFILE) im verwendeten PSB. Nehmen wir an, im gegenständlichen PSB IMSQL wäre der physische Name DPZZREP2. Man hätte dann auch schreiben können:

```
FILE DPZZREP2 OUT
   ...
      LIST DPZZREP2
   ...
```

Auch ein beliebiges Mischen der logischen und der physischen Filename ist möglich. Variable ohne Node (Punkt) werden immer im lokalen Variablenspeicher gesucht. Daher sind gleiche Namen, wie RTIKEY im File und im Segment GNSRTI, verwendbar. Sucht man im obigen Beispiel genau die Segmente, die nicht passen so wäre zu schreiben:

Beispiel 6

```
FILE 2 OUT
   RTIKEY     CHAR(13)
   TEXT       CHAR(20)
*
LET TEXT = ´NICHT VORHANDEN´
*
SELECT GNSRTI MULT
   LET GNSRTV.RTVKEY = GNSRTI.RTIKEY
   SELECT GNSRTV
   ENDSEL
   IF NO
      LET RTIKEY = GNSRTI.RTIKEY
      LIST 2
   ENDIF
ENDSEL
```

Beispiel 7

Join über Segmente und Datenbanken mit unterschiedlichen Keystrukturen:

Ausgabe aller GN-Beteiligungen mit Kundennummer

```
LOAD KDCKDKB ASM
VAR KDFU CHAR(1) INIT(´K´)
VAR KDRC FIXED(3)
VAR GBTDAT FIXED(7)
FILE 2 OUT
   RTIKEY    CHAR(13)
   BM        CHAR(4)
   MAKUNR    CHAR(10)
   ADRNR     FIXED(3)
*
SELECT GNSRTI MULT
   CALL KDCKDKB(KDFU,GNSRTI.WRG,GBTDAT,KDRC)
   SELECT KDSKONT MULT
      WHERE KDSKONT.BLZ    = GNSRTI.BLZ
      WHERE KDKSONT.KTONR  = GNSRTI.KTONR
      WHERE KDSKONT.GBTDAT = GBTDAT
      WHERE KDKSONT.BM     >= ´0001´
         SELECT KDSKOKU
            LET RTIKEY   = GNSRTI.RTIKEY
            LET DPZZREP2 = KDSKOKU BY NAME
            LET BM       = KDSKONT.BM
            LIST 2
         ENDSEL
   ENDSEL
ENDSEL
```

Der KDS-Zugriff (Kundendatensystem) hat mit teilspezifiziertem Key zu erfolgen, da der letzte Teil des Kontenkeys (BM = Beteiligungsmerkmal) n-mal an einem Konto vorkommen kann. Der Zugriff erfolgt daher mit MULT und einer Bereichsangabe beim Beteiligungsmerkmal (>=).

Da das Segment KDSKOKU nur einmal als Dependent am KDSKONT vorkommen kann und in der SYSSEG (= Segmentbeschreibungstabelle) ohne Key definiert wurde, kann es via SELECT ohne MULT vollqualifiziert gelesen werden (= *get unique*) und ist dennoch eindeutig indentifiziert. Dieses Segment enthält unter anderem die Felder MAKUNR und ADRNR, die bei der BY NAME - Zuweisung in den IO-Bereich des Files übertragen werden.

Achtung:

Wäre der Steuerinput nicht aufsteigend sortiert, so würde der innere Zugriff aus das KDS (= *get next*) nur bei Steuer-Keys funktionieren, die absolut größer sind als alle bisher im Steuerfile angegebenen, Im Extremfall, bei dem bereits der erste Key der höchste ist, wäre nur dieser Zugriff erfolgreich. Um dies zu verhindern, ist ein 2-stufiger Zugriff nötig wie beschrieben in Beispiel:

Beispiel 8

DB-Zugriff über unsortierten Input bei nur teilweise bekanntem Primärkey:

```
LOAD KDCKDKB ASM
VAR KDFU CHAR(1) INIT(´K´)
VAR KDRC FIXED(3)
VAR GBTDAT FIXED(7)
FILE 1 INP
   KTONR     FIXED(13)
   WRG       CHAR(3)
   BLZ       FIXED(5)
FILE 2 OUT
   KOIXK     CHAR(18)
   MAKUNR    CHAR(10)
   ADRNR     FIXED(3)
*
PROC AUSSI
   LET KOIXK    = KDSKONT.KOIXK
   LET DPZZREP2 = KDSKOKU BY NAME
   LIST 2
ENDPROC
*
PROC KDS
   SELECT KDSKONT MULT
      WHERE KDSKONT.BLZ    = BLZ
      WHERE KDKSONT.KTONR  = KTONR
      WHERE KDSKONT.GBTDAT = GBTDAT
      WHERE KDKSONT.BM     >= ´0001´
         SELECT KDSKOKU
            DO AUSSI
         ENDSEL
   ENDSEL
ENDPROC
*
SELECT FILE 1 MULT
   CALL KDCKDKB(KDFU,WRG,GBTDAT,KDRC)
   SELECT KDSKONT
      WHERE KDSKONT.BLZ    = GNSRTI.BLZ
      WHERE KDKSONT.KTONR  = GNSRTI.KTONR
      WHERE KDSKONT.GBTDAT = GBTDAT
      WHERE KDKSONT.BM     >= ´0001´
         SELECT KDSKOKU
            DO AUSSI
            DO KDS
         ENDSEL
   ENDSEL
ENDSEL
```

Die Ausgabe hat an 2 Stellen ident zu erfolgen und wurde daher in die Unterprozedur AUSSI ausgelagert. Der erste Zugriff aus das KDS je Inputsatz erfolgt ohne MULT d.h. mit GU (*ge unique*) und Bereichsteuerung mittels WHERE-Anweisung. Dies ermöglicht freie Positionierung in der Datenbank. In der Procedure KDS wird wieder mit MULT und *get next* zugegriffen, um alle Beteiligungen zu diesem Konto sequentiell zu erhalten.

Das läuft natürlich nur bei sortierten Segmenten. Sonst ist mit IF ... ENDIF zu arbeiten.

Beispiel 9

DB-Unload und Reload:

```
* ENTLADEN BELIEBIGES KONTO
*
FILE 1 INP LENGTH 80
   MANDNR CHAR(5)
   KTONR  CHAR(13)
   WRG    CHAR(3)
*
FILE 4 OUT
   SL BIN  FIXED(15)
   FELD    CHAR(8)
   SEG     CHAR(4000)
*
VAR SLG BIN FIXED(15) ORG SEG
*
USES GNSRTI
USES GNSROOT GNSUML GNSSPER GNSHINW GNSSOVA GNSABS GNSKART GNSMAHN
USES GNSKREI GNSBWG
USES ROOT DEPENDENT GNSORD GNSSODI GNSHIAE GNSUMS GNSZIBI GNSAUT
USES GNSRTV GNSDEK GNSZIR GNSKOND GNSKAT GNSSH GNSSCHI GNSKRED
USES GNSZI GNSVAL GNSZESS
USES GNSRTSO GNSNYM GNSWWS GNSFEST GNSFWKT
USES GNSRHI  GNSHIAEH
DBD DEGNKTST
DBD DEGNKTUM
DBD DEGNKTVA
DBD DEGNKTSO
DBD DEGNHIAE
*
PROC AUSSI
   SWITCH
      WHEN _CURSEG = 'GNSRHI'
      WHEN _CURSEG = 'GNSUMS'
      WHEN _CURSEG = 'GNSAUT'
      WHEN 'GNSHIAE' = _CURSEG
      WHEN NONE
        LET FELD = _CURSEG
        LET SEG = (FELD)
        LET SL = SLG + 10
        LIST 4
   ENDSW
ENDPROC
*
PROC KTST
   SELECT DEGNKTST MULT
      IF _CURSLVL = 1
         RETURN
      ENDIF
      DO AUSSI
   ENDSEL
ENDPROC
*
PROC KTUM
   SELECT DEGNKTUM MULT
      IF _CURSLVL = 1
         RETURN
      ENDIF
      DO AUSSI
   ENDSEL
ENDPROC
```

```
*
PROC KTVA
   SELECT DEGNKTVA MULT
      IF _CURSLVL = 1
         RETURN
      ENDIF
      DO AUSSI
   ENDSEL
ENDPROC
*
PROC KTSO
   SELECT DEGNKTSO MULT
      IF _CURSLVL = 1
         RETURN
      ENDIF
      DO AUSSI
   ENDSEL
ENDPROC
*
SELECT FILE 1 MULT
   SHOW DPZZREP1
   LET GNSRTI = DPZZREP1 BY NAME
   LET GNSROOT.KTOKEY  = GNSRTI.RTIKEY
   LET ROOT.RTUKEY   = GNSRTI.RTIKEY
   LET GNSRTV.RTVKEY   = GNSRTI.RTIKEY
   LET GNSRTSO.RTSOKEY = GNSRTI.RTIKEY
   LET GNSRHI.RHIKEY   = GNSRTI.RTIKEY
   SELECT GNSRTI
      DO AUSSI
      SELECT GNSROOT
         DO AUSSI
         DO KTST
      ENDSEL
      SELECT ROOT
         DO AUSSI
         DO KTUM
      ENDSEL
      SELECT GNSRTV
         DO AUSSI
         DO KTVA
      ENDSEL
      SELECT GNSRTSO
         DO AUSSI
         DO KTSO
      ENDSEL
      SELECT GNSRHI
         DO AUSSI
         LET FELD = 'GNSHIAEH'
         SELECT GNSHIAEH MULT
            LET SEG = (FELD)
            LET SL = SLG + 10
            LIST 4
         ENDSEL
      ENDSEL
   ENDSEL
   CHECK
ENDSEL
END
```

Und das entsprechende Reload:

```
* LADEN BELIEBIGES KONTO
*
FILE 3 INP
   SL BIN  FIXED(15)
   FELD    CHAR(8)
   SEG     CHAR(4000)
ENDFILE
*
VAR HISROOT  CHAR(1)
*
USES GNSRTI
USES GNSROOT GNSUML GNSSPER GNSHINW GNSSOVA GNSABS GNSKART GNSMAHN
USES GNSKREI GNSBWG
USES ROOT DEPENDENT GNSORD GNSSODI GNSHIAE GNSUMS GNSZIBI GNSAUT
USES GNSRTV GNSDEK GNSZIR GNSKOND GNSKAT GNSSH GNSSCHI GNSKRED
USES GNSZI GNSVAL GNSZESS
USES GNSRTSO GNSNYM GNSWWS GNSFEST GNSFWKT
USES GNSRHI GNSHIAEH
*
SELECT FILE 3 MULT
   LET (FELD) = SEG
   IF FELD = 'GNSRTI'
      LET GNSROOT.KTOKEY  = GNSRTI.RTIKEY
      LET ROOT.RTUKEY   = GNSRTI.RTIKEY
      LET GNSRTV.RTVKEY   = GNSRTI.RTIKEY
      LET GNSRTSO.RTSOKEY = GNSRTI.RTIKEY
      LET GNSRHI.RHIKEY   = GNSRTI.RTIKEY
      PUSH GNSRTI
      SELECT GNSRTI
         DELETE GNSRTI
      ENDSEL
      CHECK
      POP GNSRTI
      SELECT GNSROOT
         DELETE GNSROOT
      ENDSEL
      CHECK
      SELECT ROOT
         SELECT GNSUMS MULT
            DELETE GNSUMS
            CHECK
         ENDSEL
         DELETE ROOT
      ENDSEL
      CHECK
      SELECT GNSRTV
         SELECT GNSVAL MULT
            DELETE GNSVAL
         ENDSEL
         DELETE GNSRTV
      ENDSEL
      CHECK
      SELECT GNSRTSO
```

```
         DELETE GNSRTSO
      ENDSEL
      CHECK
      SELECT GNSRHI
         DELETE GNSRHI
      ENDSEL
      LET HISROOT = 'N'
   ENDIF
   IF FELD = 'GNSRHI '
      LET HISROOT = 'J'
   ENDIF
   IF FELD = 'GNSHIAE'
   UND HISROOT = 'J'
      LET FELD = 'GNSHIAEH'
   ENDIF
*  IF FELD ^= 'GNSHIAEH'
      LET (FELD) = SEG
      IF FELD ^= 'GNSAUT'
         SELECT (FELD)
            DELETE (FELD)
            LET (FELD) = SEG
         ENDSEL
      ENDIF
      IF FELD ^= 'GNSUMK'
      OR COUNTER(GNSUMS,INSERT) < 100
         INSERT (FELD)
      ENDIF
*  ENDIF
   CHECK
ENDSEL
```

Beispiel 10

Migration Kunde BA-CA (Heureka):

```
*************************************************************************
*         ALLGEMEINE VARIABLE FUER UNLOAD-KUNDE                         *
*************************************************************************
*
MAP  YHMKVKU  PLI DSN=SHRL.HEUK.PL1MCLIB
*
*************************************************************************
*                              BEREICHSTEUERUNG                         *
*************************************************************************
*
OSFILE IVLK INP
  USES YHMTVLK
*
*************************************************************************
*           GEMEINSAMER OUTPUT ALLER FORMATIERUNGSROUTINEN              *
*************************************************************************
*
FILE DPZZREP4 OUT LENGTH 1200
 USES HEADER
 MAP  YHMKT10 PLI DSN=SHRL.HEUK.PL1MCLIB
 MAP  YHMKT10Q PLI DSN=SHRL.HEUK.PL1MCLIB
 MAP  YHMKT31 PLI DSN=SHRL.HEUK.PL1MCLIB ORG YHMKT10
 MAP  YHMKT31Q PLI DSN=SHRL.HEUK.PL1MCLIB ORG YHMKT10+LENGTH(YHMKT31)
 MAP  YHMKT14  PLI DSN=SHRL.HEUK.PL1MCLIB ORG YHMKT10
 MAP  YHMKT39  PLI DSN=SHRL.HEUK.PL1MCLIB ORG YHMKT10
ENDFILE
*
*************************************************************************
*         VARIABLENDEKLARATIONEN FUER POSTFENSTER                       *
*************************************************************************
*
USES YHMKADR
*
*************************************************************************
*          PROTOKOLLBEREICH IN RESTARTBEREICH SICHERN (HOLD)            *
*************************************************************************
*
VAR CPROT CHAR(400) HOLD
USES YHMTPROT ORG CPROT
*
*************************************************************************
*                    AUFZURUFENDE UNTERPROGRAMME                        *
*                                                                       *
*                       FIXE UNTERPROGRAMME:                            *
*                                                                       *
* UHMTUHD - HEADERFORMATIERUNG AM ANFANG DES PROGRAMMES                 *
* UHMTUPR - PROTOKOLLAUSGABE AM ENDE DES PROGRAMMES                     *
*************************************************************************
*
LOAD UHMTUHD PLI
LOAD UHMTUPR PLI
*
*************************************************************************
*                 ANWENDERSPEZIFISCHE UNTERPROGRAMME                    *
*************************************************************************
*
LOAD UHMKU01 PLI
LOAD UHMKU02 PLI
LOAD UHMKU03 PLI
```

```
LOAD UHMKU04 PLI
LOAD UHMKUVK PLI
LOAD UHMKUTI PLI
*
*********************************************************************
*                  LESEN UND AUSGEBEN VON SEGMENTEN:                *
*********************************************************************
*
PROC ADR
   SELECT KDSADR MULT
     STORE KDSADR
     IF KDSADR.ADRKZ ^= '3'
       CALL UHMKU02(_SEGINT)
       LIST 4
       CALL UHMKU04(_SEGINT)
       SEARCH DPZZREP4 MULT
         LIST 4
       ENDSEARCH
       RELEASE DPZZREP4
     ENDIF
     SELECT KDSNELI MULT
       CALL UHMKU03(_SEGINT)
       LIST 4
     ENDSEL
   ENDSEL
ENDPROC
*
*
PROC NEG
   SELECT KDSNEG MULT
*     CALL UHMKU0X(_SEGINT)
*     LIST 4
     STORE KDSNEG
   ENDSEL
ENDPROC
*
*
PROC KALK
   LET KDSKALK.KSOKEY = 1
   SELECT KDSKALK
   ENDSEL
   IF NO
     LET KDSKALK.KSOKEY = 9
   ENDIF
ENDPROC
*
*********************************************************************
*                         LESEN BEREICHSTEUERUNG                    *
*********************************************************************
*
SELECT FILE IVLK
   SHOW YHMTVLK
ENDSEL
*
*********************************************************************
*      INITIALISIEREN STANDARDHEADER UND PROTOKOLLSTRUKTUR (FIX)     *
*                         INIT DIVERSER VARIABLEN                   *
*********************************************************************
*
CALL UHMTUHD(_SEGINT)
LIST 4
*
*********************************************************************
```

```
*                                HAUPTVERARBEITUNG                              *
*****************************************************************************
*
LET YHMKVKU.A_KUVK = 'E'
LET YHMKVKU.VONDAT = '01.07.2000'
LET YHMKVKU.VONZEI = '06.00.00'
SELECT KDSKUND MULT
   WHERE KDSKUND.MANDNR >= YHMTVLK.VMAND
   WHERE KDSKUND.KUNDNR >= YHMTVLK.VKEY
   WHERE KDSKUND.MANDNR <= YHMTVLK.BMAND
   WHERE KDSKUND.KUNDNR <= YHMTVLK.BKEY
      LET YHMKVKU.VORNAME = ' '
      LET YHMKVKU.TITELR = ' '
      LET YHMKVKU.TITELA = ' '
      LET YHMKVKU.TITELB = ' '
      LET YHMKVKU.TITELZ = ' '
      LET YHMKVKU.NEGKZ = 'N'
      LET YHMKVKU.LEGIT1 = ' '
      LET YHMKVKU.LEGIT2 = ' '
      CALL UHMKUVK(_SEGINT)
      CALL UHMKUTI(_SEGINT)
      DO ADR
      DO NEG
      DO KALK
      CALL UHMKU01(_SEGINT)
      LIST 4
      RELEASE KDSADR
      RELEASE KDSNEG
      CHECK NOCOUNT
ENDSEL
*
*****************************************************************************
*                                PROTOKOLLAUSGABE                             *
*****************************************************************************
*
CALL UHMTUPR(_SEGINT)
CALL UHMTUHD(_SEGINT)
LIST 4
```

Anhang 2 - Synonymtabelle

Im Folgenden sind sämtliche Keywörter in alphabetischer Reihenfolge und deren Synonyme angeführt. Bei Variablendeklaration, vor allem bei impliziter Deklaration in einer Filedeklaration, ist die Verwendung der Keywörter und deren Synonyme dringend zu vermeiden. Solche Doppelbelegungsfehler bringen oft völlig unerwartete Resultate.

<	LT,LESS,LESSTHAN,KLEINER,KL
<=	=<,LE,LESSEQUAL,KLEINERGLEICH,KG,KLGL
+	ADD,PLUS,ZUZUEGLICH,ERHOEHTUM
!!	CAT,CONCAT,VERKNUEPFT,VERBINDE
*****	REM,REMARK,COMMENT,MAL,KOMMENTAR
^=	=^,^,NOT,NE,UNGLEICH,NICHT
-	MINUS,SUBTRACT,WENIGER,ABZUEGLICH,VERMINDERTUM
/	:,DIV,DIVIDE,DURCH,GETEILTDURCH
>	GT,GREATER,GREATERTHAN,GROESSER,GR
>=	=>,GE,GREATEREQUAL,GROESSERGLEICH,GG,GRGL
=	EQ,EQUAL,GLEICH,IST,AUF,ZU,MIT
ABS	ABSOLUT
ADDR	ADRESSE
ALTER	AENDERE,VERAENDERE,MODIFIZIERE
ALWAYS	IMMER
AND	UND,&
ASM	ASSEMBLER
BIN	BINARY,BINAER
BIT	
BYNAME	BY,NAMENTLICH
CALL	RUF,RUFE
CHAR	CHARACTER,ZEICHEN,ZEICHENKETTE
CHECK	CHECKPOINT,COMMIT
CLOSE	SCHLIESSE
CMD	COMMAND,BEFEHL
COB	COBOL,KOBOLD
COUNT	COUNTER,INT,INTERNAL,ZAEHLER
DB	DBD,DATABASE,DATENBANK
DD	DDN,DDNAME
DEC	DECIMAL,DEZIMAL
DECNUM	NUMDEC,PACKED
DELETE	LOESCH,LOESCHE,ENTFERNE
DISPLAY	DSPLY,DISP
DO	MACH,TU
DSN	DA,DATASET,DATASETNAME,DATEI
DUMP	SPEICHERAUZUG,MUELL
ELSE	ANDERNFALLS,SONST
END	ENDE,STOP,AUS,HALT
ENDFILE	DATEIENDE,EOF,ENDOFFILE,ENDEDATEI
ENDIF	AUSWENN
ENDLOOP	AUSSCHLEIF,AUSLAUF,SCHLEIFENENDE
ENDSRCH	ENDSEARCH,ENDSUCH,ENDSR,ENDFIND,ENDFINDE
ENDPROC	AUSPROGRAMM,PROGRAMMENDE
ENDSEL	AUSLESEN,GELESEN
ENDSW	ENDSWITCH,AUSSCHALT,ENTSCHIEDEN
EXIT	

FILE	DATEI
FIELD	FELD,FIELDNAME,FELDNAME
FIRST	ERSTES,PRIMARY,ANFANG
FIX	FEST,FIXLANG
FIXED	GEPACKT,FESTKOMMA
FLDCNT	FIELDCOUNT,FELDANZ,ANZFELD
FLOW	FLUSS,LAUFZEIT,STMT,STATEMENT
FORM	FORMAT,DATENFORMAT,FELDFORMAT
FULL	ALLES,KOMPLETT
GSAM	GSAMDATEI
GOTO	GO,GEHZU
HEX	HEXADECIMAL,HEXADEZIMAL
HOLD	RESTART,SICHER,GESICHERT
IF	WENN,ANGENOMMEN
INDEX	STELLE
INIT	INITIAL,VORAUS,VORAUSBELEGT,KONSTANT
INP	IN,INPUT,EINGABE
INSERT	FUEGEIN,SCHREIB,SCHREIBE,NEU,NEUES
KEY	SCHLUESSEL,SCHLUESSELBEGRIFF
KOMMA	KOMMASTELLEN,DIGIT,SCALE
LABEL	MARKE,SPRUNGMARKE
LAN	LANGUAGE,SPRACHE,LANG
LENGTH	LAENGE,LEN,LNGE,LG
LET	SETZ,SETZE,AENDERE,BELEGE
LIST	PUT,SCHREIB,SCHREIBE
LOAD	LADE
LOOP	SCHLEIF,LAUF
MAP	GENERIERE
MEM	MEMBER
MULT	MULTIPLE,OEFTER,MEHRFACH,ALLE
NAME	BEZEICHNUNG
NO	FALSE,NICHT,FALSCH
NOSEG	NURDB,KEINSEG
NOCHECK	CHECKNIX
NOCOUNT,	ZAEHLNICHT
NODISP	NODSPLY,NODISPLAY
NODUMP	KAMIST,KASCHAS
NOEXIT	KAAUSGANG,KEINEXIT,NATIVE
NOIMS	MVS,ZOS,NODLI
NONE	KEINE,KEINER,NICHTS,"
NOOPEN	LASSZU
NOSORT	NOSRT,UNSORT,UNSORTIERT
NOTIME	ZEITLOS
NOTRACE	
NOUPDATE	NOUP,WIEDERHOLBAR
NOW	NUN,JETZT
OCC	OCCURENCY
OPEN	OEFFNE
OR	ODER,!
ORG	BASED,UEBERLAGERT,UEBER
OS	OSFILE,EXT,EXTERNAL,RECORD
OUT	OUTPUT,AUSGABE
PARENT	UEBERGEORDNET,MUTTER,VATER,ELTERN
PCB	DBPCB,DBPOINTER,DB-POINTER,DBPTR
PLI	PL1

POP	HOL,HOLE
PROC	SUBROUTINE,PROGRAMM,UNTERPROGRAMM
PROCSEQ	SEQ,INDEX,REIHENFOLGE
PUSH	
RELEASE	CLEAR,VERGISS
RESTORE	UEBERSCHREIB
REPLACE	ERSETZE
RETURN	RAUS,ZURUECK
ROLB	ROLLBACK,RUECKNAHME,VERGISSES
ROOTS	NODEP
SCAN	SYNTAX,VORLAUF
SCREEN	SCR,MESSAGE,MSG,SCHIRM
SEARCH	SUCH,FIND,FINDE
SELECT	GET,LIES,LESEN
SHOW	ZEIG,ZEIGE
SORT	SRT,SORTIERT
STORE	MERK,HALTE
SUBSTR	SUBSTRING,TEIL,TEILKETTE,TEILBEREICH
SWITCH	SCHALT,SCHALTE,ENTSCHEIDE
SYNC	SYNCPOINT,SYNCHRONISIERE
TAB	TABELLE
TEST	DEBUG
TRACE	KONTROLLE,KONTROLLIERE,VERFOLGE
UACC	ACCESS,ZUGRIFF,DBZUGRIFF
UPDATE	UPD,AENDERUNG
USES	USE,VERWENDE
VAR	VARIABLE,DCL,DECLARE,SPEICHER,WERT
VERIFY	UEBEREINSTIMMUNG,PASST,ZEICHENUEBEREINSTIMMUNG
VSAM	KEYED,CLUSTER
WHEN	FALLS
WHERE	WO
YES	RIGHT,RICHTIG,JA,OK

Anhang 3 - Handshake IMSQL <> PLI-Subroutines

Architektur: Sämtliche Lese- und Schreibzugriffe (via IMS) sollten ausschließlich im IMSQL -Hauptprogramm durchgeführt werden. Dies erfordert die Möglichkeit eines direkten Datenzugriffes der Unterroutinen auf die Arbeitsbereiche des IMSQL. Mittels der Handshake-Funktion über PLI-Macros stehen sämtliche Variable und Segmente den aufgerufenen Modulen über den Strukturnamen zur Verfügung.

Probleme: a) Ein Programm (IMSQL -Hauptprogramm oder PLI-Unterroutine) benötigt gleichzeitigen Zugriff auf alle gleichartigen Segente einer Dependentschicht,

b) Subroutine muß innerhalb eines Aufrufes mehrere Outputsegmente formatieren und ausgeben z.B. Warnungen oder Fehlerhinweise.

Lösung: Segmente, oder besser alle namentlich bekannten Strukturen können beliebig oft im Hauptspeicher angelegt werden. Sie können direkt (gekeyed) oder sequentiell ver- oder bearbeitet sowie auch unabhängig ausgegeben werden. Der Mechanismus steht bei völliger gegenseitiger Kompatibilität in beiden Umgebungen, IMSQL und PLI auch syntaktisch angeglichen zur Verfügung.

Elemente: Die Commands STORE, SEARCH, RESTORE, RELEASE (nur IMSQL) und FIRST sind im IMSQL als übliche CMDS, im PLI als MAKROS realisiert, die mit IMSQLI includiert werden können.

Wird auch das Entry Makro IMSQLE verwendet, so kann der Zugriff auf die IMSQL - Bereiche mittels Call-Makro IMSQLC aus einem PLI-Unterprogramm auf weitere PLI-Routinen weitergegeben werden. Die Länge der Kette ist nicht beschränkt, sämtliche Datenmanipulationen an den internen Bereichen sind allen beteiligten Komponenten transparent.

Die Datenspeicherung erfolgt via Speicherallokation im Hauptspeicher, ist entsprechend schnell, allerdings muss auf korrekte Speicherfreigabe via RELEASE geachtet werden um Platzprobleme zu vermeiden.

Sprachelemente

STORE segment

Anlegen einer Kopie der Struktur *Segment* im Hauptspeicher. Der originale Segmentspeicher (= das Segment selbst) bleibt unverändert. Strukturen können beliebig oft gestored werden (so lange Speicherplatz verügbar ist), die Segmentliste ist jedoch immer umgekehrt sortiert: Das zuletzt geSTOREte Element bildet den Kopf der Liste. Zum Einen vereinfacht das den internen Listenaufbau, zum Anderen steht die Datenkette gleich für den nächsten Lesezugriff komplett zur Verfügung. Es können nur ganze Segmente, keine Einzelvariablen geSTOREd werden. Für diese ist weiterhin PUSH - POP mit den bekannten Eigenschaften zu verwenden.

SEARCH segment <MULT>

Datenrückführung aus der Segmentspeicherliste in den originalen Segment-IO-Bereich. Alle Programme sehen immer nur den originalen IO-Bereich und können diesen bearbeiten. Bei SEARCH wird die spezifizierte Version aus der Segmentliste in den echten Segment-speicher übertragen. Die Variantenidentifikation kann direkt (= gekeyed) oder sequentiell erfolgen. Ist MULT kodiert so erfolgt ein sequentielles Lesen bis EOD der Store-Kette, bei fehlendem MULT wird ein Segment im Speicher mit dem Key aus dem originalen Segment gesucht. Analog SELECT wird bei erfolgreichem Zugriff zum nächsten Statement, sonst zum entsprechenden ENSDSEARCH verzweigt.

RESTORE segment

Update einer Kopie der Struktur *Segment* im Hauptspeicher. Der originale Segmentspeicher (= das Segment selbst) bleibt unverändert. Die geSTOREte Version des *Segment*es wird lt. Key des IO-Bereiches gesucht und mit dessen Inhalt überschrieben. Wird keine Eintragung lt. Key gefunden, so wird ein neuer Clone geSTOREd. Bei nicht gekeyten Segmenten gilt die aktuelle Postion lt. letztem SEARCH .Im Konditioncode (via IF NO abfragbar) steht das Resultat des vorausgegangenen SEARCH - Zugriffes.

FIRST segment

Repositionierung auf den Listenkopf. Bei mehrfachem sequentiellen Lesen oder sequentieller Verarbeitung nach get-unique muß auf das erste geSTOREte Element repositioniert werden. FIRST verändert nicht den originalen IO-Bereich und erzeugt auch bei fehlender Kette keinen Fehler.

RELEASE segment

Die gesamte Kette eines gespeicherten Segmentes wird wieder freigegeben und gelöscht. Mehrfaches RELEASE desselben Segmentes erzeugt keinen Fehler, auch wird der originale IO-Bereich nicht verändert.

PLI-Syntax STORE (segment)
SEARCH (segment <,MULT>)
RESTORE (segment)
FIRST (segment)
RELEASE in der Subroutine nicht unterstützt!
IMSQLC (pliroutine) Weitergabe der Zugriffsfunktionen
IMSQLE (pliroutine<,pam1...,parmn>) Standard-Entry-Makro
IMSQLI beinhaltet alle Arbeitsmakros

PLI-Makros in IMSQLI STORE , SEARCH, RESTORE, FIRST

SETSYSSEG(`strukturname`) Adressierung Struktur aus IMSQL

SETSYSVAR(`variable`) Adressierung Variable aus IMSQL

SETSYSPCB(`dbsegnam`) retourniert die Adresse des PCB auf die Datenbank des betroffenen DB-Segmentes

Beispiele:

a) parallele Segmentspeicherung im IMSQL und Bearbeitung ebendort:

```
SELECT ROOT
            SELECT DEPENDENT MULT
                  STORE DEPENDENT
                     ... sonst Segmentverarbeitung ...
            ENDSEL
            LET DEPENDENT.KEY = ´0001´
            SEARCH DEPENDENT
                  ...   jetzt steht dieses Segment im Speicher,
                        wenn gefunden,ändern
                  RESTORE DEPENDENT ... Update gesp. Version
            ENDSEARCH
            FIRST DEPENDENT         ... an den Listenanfang
            SEARCH DEPENDENT MULT
                  ... und alle noch einmal bearbeiten ..
            ENDSEARCH
            RELEASE DEPENDENT       ... Liste abbauen
ENDSEL
```

b) parallele Segmentspeicherung im IMSQL und Bearbeitung im PLI

```
LOAD PLIMOD01 PLI
SELECT ROOT
            SELECT DEPENDENT MULT
                  STORE DEPENDENT
            ENDSEL DEPENDENT
            CALL PLIMOD01(_SEGINT)
            RELEASE DEPENDENT
ENDSEL
```

Und im PLI-Modul: (bitte beachten Sie das spezielle ENTRY-Macro)

```
IMSQLE(PLIMOD01)
%INCLUDE IMSQLI;
DCL $DEPENDENT PTR;
DCL 1 DEPENDENT based($DEPENDENT).
   %INCLUDE DEPENDENT;;
$DEPENDENT = SETSYSSEG(`DEPENDENT`);
            ...
SEARCH (DEPENDENT,MULT)
            ... Verarbeitung PLI ...
END;

DEPENDENT.KEY = `0001`;
SEARCH (DEPENDENT)
            ... Einzelverarbeitung ...
END;

END PLIMOD01;
```

c) parallele Segmentspeicherung und Bearbeitung in IMSQL und PLI, Beispiel für mehrere Fehlerkennungen bei einem Outputbestand.

```
FILE OUTDB GSAM OUT LENGTH 1000
            MAP   HEADER
            MAP   DATEN
*
LOAD PLIMOD01 PLI
VAR RC CHAR(1)
*
SELECT ROOT
            SELECT DEPENDENT MULT
                  STORE DEPENDENT
            ENDSEL DEPENDENT
            CALL PLIMOD01(_SEGINT)
            LIST OUTDB              ... Ausgabe aktuellen IO-Speicher ...
            SEARCH HEADER MULT
                  LIST OUTDB     ... Ausgabe etwaig gespeicherte Daten ...
            ENDSEARCH
            RELEASE DEPENDENT
            RELEASE HEADER
ENDSEL
```

Und im PLI-Modul:

```
IMSQLE(PLIMOD01)
%INCLUDE IMSQLI;
DCL ($HEADER,$DEPENDENT) PTR;
DCL 1 DEPENDENT based($DEPENDENT),
   %INCLUDE DEPENDENT;;
DCL 1 HEADER BASED($HEADER),
   %INCLUDE HEADER;;
$DEPENDENT = SETSYSSEG(`DEPENDENT`);
$HEADER    = SETSYSSEG(`HEADER`);
            ...
SEARCH (DEPENDENT,MULT)
            ... Verarbeitung PLI ...
            IF warning/error THEN
            DO;
                  STORE (HEADER)          /* sichere aktuellen Inhalt */
                  HEADER.FEHLCOD = `I`;
                  HEADER.FEHLER  = `DA STIMMT WAS NICHT`;
            END;
END;  /* SEARCH */
```

a) parallele Segmentspeicherung und Bearbeitung in IMSQL und PLI, Beispiel für mehrere Outputbestände. Der gesamte Outputbereich wird gepeichert. Achtung unbedingt LENGTH angeben, da die Defaultlänge für IO-Segmente 32767 Bytes sind und ansonsten in dieser Länge geSTOREd wird.

b) Beachte:

c) Das PLI-Modul ruft seinerseits via IMSQLC-Makro weitere PLI-Subroutinen im IMSQL -Enviroment auf.

IMSQLC lädt die Module resident für den kompletten Programmlauf und stellt ihnen die IMSQL - internen Seicherbereiche zur Verfügung. Die Module können beliebig aus dem Hauptprogramm oder aus PLI-Subroutinen aufgerufen werden.

```
FILE OUTDB OUT LENGTH 1000
            MAP   HEADER
            MAP   DATEN
*
LOAD PLIMOD01 PLI
*
SELECT ROOT
            SELECT DEPENDENT MULT
                  STORE DEPENDENT
            ENDSEL DEPENDENT
            CALL PLIMOD01(_SEGINT)
            LIST OUTDB        ... Ausgabe aktuellen IO-Speicher ...
            SEARCH OUTDB MULT
                  LIST OUTDB     ... Ausgabe etwaig gespeicherte
            Outputbereiche ...
            ENDSEARCH
            RELEASE DEPENDENT
            RELEASE OUTDB
ENDSEL
```

Und im PLI-Modul:

```
IMSQLE(PLIMOD01)
%INCLUDE IMSQLI;
DCL $DEPENDENT PTR;
DCL 1 DEPENDENT based($DEPENDENT).
   %INCLUDE DEPENDENT;;
DCL 1 OBER1 BASED($OUT),
            %INCLUDE FILFORM1;
DCL 1 OBER2 BASED($OUT),
            %INCLUDE FILFORM2;

$OUT        = SETSYSSEG(`OUTDB`);
$DEPENDENT  = SETSYSSEG(`DEPENDENT`);

            ...
SEARCH (DEPENDENT,MULT)
            ... Verarbeitung PLI ...
            OBER1. .... = ...;
            STORE (OUTDB)          /* sichere aktuellen Inhalt */
            DO;
                  WHILE(AUSGABESAETZE_VORHANDEN);
                  IMSQLC(PLIMOD02<,Parms,...>);
```

```
                IMSQLC(PLIMODnn);
                OBER2. .... = ...;
                STORE (OUTDB)
            END;
END;  /* SEARCH */
```

NOTIZEN