1 Einleitung

Dieses Kapitel beschreibt kurz das Produkt DRIVE, die Zielgruppe des Handbuchs sowie das Konzept der Handbuchreihe. Außerdem enthält es eine Liste mit Änderungen gegenüber der Vorgängerversion sowie eine Aufstellung der in den DRIVE-Handbüchern verwendeten Darstellungsmittel.

1.1 Kurzbeschreibung des Produkts

DRIVE ist eine Programmiersprache der 4. Generation (4GL) zur Entwicklung kommerzieller Client-Server-Anwendungen. Sie ist die 4GL für Zugriff auf das BS2000 Datenbanksystem SESAM/SQL V2 und auf Dateien.

Die einheitliche Sprache mit mächtigen und leicht erlernbaren Anweisungen ermöglicht die Erstellung komplexer Anwendungen für Datenbankzugriff, Report, Oberfläche, Kommunikation und Verarbeitung. DRIVE versorgt automatisch systemnahe Schnittstellen zu den jeweiligen Komponenten und nimmt dem Programmierer diese Arbeit ab.

Zum Test seiner DRIVE-Anwendung stellt DRIVE dem Programmierer einen integrierten Debugger zur Verfügung.

DRIVE-Anwendungen können unabhängig vom Einsatz eines Transaktionsmonitors erstellt und getestet werden. Sie sind ohne Änderungen mit oder ohne Transaktionsmonitor ablauffähig.

Zur Steigerung der Performance können die erstellten DRIVE-Anwendungen mit dem Compiler DRIVE-COMP übersetzt werden.

1.2 Zielgruppe

Das Handbuch wendet sich an Programmierer, die DRIVE-Anwendungen oder Teile von VTV-Anwendungen mit DRIVE auf BS2000-Rechnern entwickeln. Dafür benötigt der Programmierer allgemeine Kenntnisse über das Betriebssystem BS2000.

Abhängig vom konkreten Einsatzfall sind weitere Kenntnisse erforderlich über:

– das Datenbanksystem SESAM

– den Transaktionsmonitor *open*UTM

– das Format Handling System FHS zur Erstellung von Bildschirmen

1.3 Konzept dieses Handbuchs

Dieses Handbuch ergänzt die Handbücher zu DRIVE/WINDOWS V2.1. Es beschreibt die neuen und geänderten Funktionen von DRIVE V2.2. Jedem Kapitel ist ein Handbuch aus der Dokumentation zu DRIVE/WINDOWS V2.1 (BS2000) zugeordnet; zusätzlich gibt es ein Kapitel mit Handbuchkorrekturen zu DRIVE/WINDOWS V2.1.

Im Einzelnen beziehen sich die Kapitel auf folgende Handbücher:

- Kapitel 2 enthält Korrekturen zur „DRIVE-Programmiersprache“ [2], zum „DRIVE-Lexikon“ [3] und zum „DRIVE-SQL-Lexikon“ [4].
- Kapitel 3 bezieht sich auf „DRIVE-Programmiersystem“ [1].
- Kapitel 4 bezieht sich auf das „DRIVE-SQL-Lexikon“ [4].
- Kapitel 5 bezieht sich auf das „DRIVE-Lexikon“ [3].
- Kapitel 6 und 7 beziehen sich auf „DRIVE-Programmiersprache“ [2].

1.4 Readme-Datei

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei. Sie finden die Readme-Datei auf Ihrem BS2000-Rechner unter dem Dateinamen SYSRME.*produkt.version.sprache*. Die Benutzerkennung, unter der sich die Readme-Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Die Readme-Datei können Sie mit dem Kommando /SHOW-FILE oder mit einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrucken:

```
/PRINT-DOCUMENT dateiname, LINE-SPACING=*BY-EBCDIC-CONTROL
```

bei SPOOL -Versionen kleiner 3.0A:

```
/PRINT-FILE FILE-NAME=dateiname,LAYOUT-CONTROL=
    PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
```

1.5 Änderungen gegenüber DRIVE V2.1

1.5.1 Komponenten

- DRIVE V2.2 arbeitet nur noch mit SESAM ab V2.x zusammen. Daher hat sich die Beschreibung zum Einsatz von DRIVE (siehe Kapitel „Einsatz von DRIVE“ auf Seite 13) und zum Anschluss von Datenbanken (siehe Kapitel „Datenbanken“ auf Seite 187) geändert.
- Ab SESAM/SQL V3.0 werden temporäre Views nicht mehr unterstützt.
- FHS-DE wird nicht unterstützt. Daher können die DRIVE-Anweisungen ADD BOX, REMOVE BOX und REPLACE BOX nicht verwendet werden. Entsprechendes gilt für die FHS-DE-spezifischen Parameter der Anweisung DISPLAY sreeenformat.

1.5.2 Neue DRIVE-SQL-Anweisungen

Alle im Folgenden aufgeführten Anweisungen sind auch dynamisch ausführbar.

- Neue SQL-Anweisungen zum Verwalten von Benutzereinträgen

Anweisung	**Seite**
CREATE SYSTEM_USER	113
CREATE USER	120
DROP SYSTEM_USER	133
DROP USER	137

- Neue SQL-Anweisungen zur Verwaltung der Speicherstruktur

Anweisung	**Seite**
ALTER SPACE	85
ALTER STOGROUP	87
CREATE INDEX	103
CREATE SPACE	108
CREATE STOGROUP	111
DROP INDEX	129
DROP SPACE	131

Anweisung	Seite
DROP STOGROUP	132
REORG STATISTICS	155

- Neue UTILITY-Anweisungen zur Datenbankverwaltung, siehe Seite 161:

  ALTER MEDIA DESCRIPTION
  CHECK CONSTRAINTS
  CHECK FORMAL
  COPY
  CREATE CATALOG
  CREATE MEDIA DESCRIPTION
  CREATE REPLICATION
  DROP MEDIA DESCRIPTION
  LOAD
  MIGRATE
  MODIFY
  RECOVER
  REFRESH REPLICATION
  REORG
  UNLOAD

- Neue Pragma-Klauseln, siehe Seite 144f:

  JOIN
  LOCK MODE
  UTILITY MODE

1.5.3 Erweiterte DRIVE-SQL-Anweisungen

- SQL-Anweisungen zur Datenbankverwaltung

Anweisung	Erweiterung	Seite
ALTER TABLE	column-definitions	89
CREATE SCHEMA	CREATE INDEX	106
CREATE TABLE	CALL DML	116
DROP SCHEMA	CASCADE	130
DROP TABLE	CASCADE	136
DROP VIEW	CASCADE	138

Anweisung	Erweiterung	Seite
GRANT	Spezialprivilegien	139
REVOKE	Spezialprivilegien	156

- SQL-Anweisungen zum Abfragen und Ändern von Daten

Anweisung	Erweiterung	Seite
DECLARE	FOR READ ONLY	121

1.5.4 Behandlung von SESAM-Warnings und -Meldungen

- Ausnahmebehandlung von SESAM-Warnungen:

  Die DRIVE-Anweisung WHENEVER wurde um die Systemvariable &WARNING erweitert, siehe Seite 164f.

- SQL-Meldungstexte ermitteln:
  Mit der neuen Funktion SQLMSGSTRING können SQL-Meldungstexte ermittelt werden, siehe Seite 169.

- Erweiterung der Systemvariablen &ERROR_STATE um &WARNING und &DISTRIBUTION_STATE um &DIS_WARNING.

1.5.5 Sonstige Änderungen

- Bei der Report-Anweisung PAGE PRINT muss bei Positionsangaben das Schlüsselwort POSITION angegeben werden, siehe Seite 168.
- Die Einsatzmöglichkeiten der Abkürzung „.*“ haben sich geändert, siehe Seite 182.

1.6 Darstellungsmittel

Die in diesem Handbuch verwendeten Zeichen und Schriftarten haben folgende Bedeutung:

`Schreibmaschinenschrift`

wird für feststehende Namen (z. B. Kommandos auf Betriebssytemebene, Dateinamen) und Fehlermeldungen im Fließtext verwendet. Außerdem wird sie in den Beispielen benutzt.

Kursive Schrift

kennzeichnet als Zwischenüberschrift Beispiele und im Fließtext frei wählbare Namen sowie Metavariablen.

Dieses Zeichen weist Sie auf eine sehr wichtige Information hin, die Sie unbedingt beachten müssen.

Bei Literaturverweisen, z.B. auf die oben erwähnten Handbücher, wird der Kurztitel zusammen mit einer Zahl in eckigen Klammern angegeben. Im Anhang befindet sich eine Literaturliste, die nach diesen Zahlen aufsteigend angelegt ist.

Syntax der DRIVE- und DRIVE-SQL-Anweisungen

Für die formale Darstellung der Anweisungen und Metavariablen werden folgende Metazeichen verwendet.

Formale Darstellung	**Erläuterung**	**Beispiel**
GROSSBUCHSTABEN	Großbuchstaben bezeichnen ein Schlüsselwort, das Sie in dieser Form eingeben müssen.	`COLUMNS`
Fettdruck	Die fettgedruckten Buchstaben bezeichnen die Abkürzung eines Schlüsselworts.	**`PERM`**`ANENT`
Kursive Kleinbuchstaben	Kursive Kleinbuchstaben bezeichnen eine Variable, für die Sie einen aktuellen Wert einsetzen müssen.	`LIBRARY=`*`bibliothek`*

Formale Darstellung	**Erläuterung**	**Beispiel**
( )	Runde Klammern sind Bestandteil der Anweisung. Runde Klammern müssen angegeben werden, sobald eine Angabe in runden Klammern steht.	*bibliothek*(*elemname*) oder CONCAT (*charausdruck1*, *charausdruck2*) oder ATTRIBUTE (*attribute*, ...)
{ }	Geschweifte Klammern fassen Einheiten zusammen. Geklammert wird von innen nach außen. Geschweifte Klammern dürfen nicht angegeben werden.	STATUS={ OFF | ADD | REMOVE } oder USING { [RETURN] [*level*] *varname datendef* }, ...
[ ]	Eckige Klammern schließen Angaben ein, die weggelassen werden können. Geklammert wird von innen nach außen. Eckige Klammern dürfen nicht angegeben werden.	[*set transaction*] oder [COBOL | C] TAC *tacname*
< >	Spitze Klammern sind Bestandteil der Anweisung. Spitze Klammern müssen angegeben werden, sobald eine Angabe in spitzen Klammern steht.	*aggregat*=< { *wert* | NULL }, ... >
|	Der senkrechte Strich trennt alternative Operandenwerte. Eine der Alternativen in geschweiften Klammern muss angegeben werden.	LETTERS={ CAPITAL | BOTH | UNCHANGED }

Formale Darstellung	Erläuterung	Beispiel
...	Punkte zeigen an, dass die unmittelbar davor stehende Variable mehrmals wiederholt werden kann.	AT *zeile* ...
	Steht vor den Punkten eine mit Klammern zusammengefasste Einheit, muss sie insgesamt angegeben werden.	USING { [RETURN] [*level*] *varname datendef* }, ...
	Steht ein Komma oder ein Semikolon vor den Punkten, muss es bei Wiederholungen angegeben werden, um mehrere Angaben der gleichen Strukturstufe zu trennen.	(*attribute*, ...)

2 Fehlerbehebungen

In den Handbüchern zu DRIVE/WINDOWS V2.1 sind leider einige Fehler aufgetreten, die im folgenden berichtigt werden.

2.1 Korrekturen zur DRIVE-Programmiersprache

Das Handbuch „DRIVE/WINDOWS - Programmiersprache“ [2] muss wie folgt korrigiert werden:

Zuweisungsverträglichkeit bei der Datenkonvertierung

(Zu Abschnitt 3.6 auf Seite 79 (Erläuterung von Punkt 19 der Tabelle auf Seite 75))

19 Die Konvertierung ist zuweisungsverträglich und vergleichbar. Ein Vergleich zwischen verschiedenen Datentypen liefert immer den Wert „ungleich“.

DATE → TIMESTAMP(3)
Bei der Konvertierung wird die Uhrzeit *stunde*:*minute*:*sekunde*.*bruchteil* mit der aktuellen Zeit aufgefüllt.

2.2 Korrekturen zum DRIVE-Lexikon

Im Handbuch „DRIVE/WINDOWS V2.1- Lexikon der DRIVE-Anweisungen“ [3] sind folgende Korrekturen vorzunehmen:

DECLARE VARIABLE - Variable definieren

(zu Kapitel 3, Seite 75f)

Die LIKE-Klausel zum Kopieren eines Cursors oder Tabelle in eine Variable ist nur auf oberster Stufe (Level = 1) erlaubt.

OPTION - Übersetzung eines DRIVE-Programms steuern

(zu Kapitel 3, Seite 150)

Bei Übersetzung eines DRIVE-Programms mit Option OBJECT=ON zur Erzeugung eines Objektcodes sind folgende Parameter-Anweisungen im Programm verboten:

– alle statischen Parameter

– dynamische Parameter LOG, LOGFILE, LOGPASSWORD, NORMSQL, SCHEMA, CATALOG, AUTHORIZATION und TEST

PARAMETER LOCK - DRIVE-Anweisung sperren

(zu Kapitel 3, Seite 175)

Die Anweisungen DELETE und UPDATE können auf folgende Weise gesperrt werden:

– DELETE { SEARCHED | POSTIONED }

– UPDATE { SEARCHED | POSTIONED }

READ FILE - Datei lesen

(zu Kapitel 3, Seite 183)

Die DRIVE-Systemvariablen &PHYS_REC_LENGTH und &DRIVE_REC_LENGTH gibt es nicht.

nullwert - Darstellung des NULL-Werts definieren

(zu Kapitel 5, Seite 346)

Bei Bildschirmein-/ausgaben ist die NULL-Wertdarstellung mit dem Zeichen ‘@’ vorbelegt.

2.3 Korrekturen zum DRIVE-SQL-Lexikon

Im Handbuch „DRIVE/WINDOWS V2.1- Lexikon der DRIVE-SQL-Anweisungen für SESAM/SQL 2“ [4] sind folgende Korrekturen vorzunehmen:

DECLARE - Cursor vereinbaren

(zu Kapitel 3, Seite 88)

```
DECLARE CURSOR [ { PERMANENT | TEMPORARY } ]
          [ SCROLL ]  CURSOR  [ PREFETCH n ]
          [ FOR cursorbeschreibung ]
```

SET TRANSACTION – Transaktionseigenschaften festlegen

(zu Kapitel 3, Seite 143)

Im Dialogbetrieb einer DRIVE-UTM-Anwendung hat die SET TRANSACTION-Anweisung keine Auswirkung auf die folgende SQL-Transaktion.

Das liegt daran, dass durch die SET TRANSACTION-Anweisung die SQL-Transaktion nicht geöffnet wird und DRIVE die Durchführung der Anweisung mit der Meldung DRI0009 quittiert, wobei die UTM-Transaktion beendet wird. Durch die Transaktions-Synchronisierung mit SESAM wird dabei auch die SQL-Transaktion “beendet”.

Dadurch wird wieder die Defaulteinstellung des Transaktionslevels aktiv.

3 Einsatz von DRIVE

Dieses Kapitel enthält die Änderungen und Ergänzungen zum „DRIVE-Programmiersystem“ [1].

3.1 Dialog mit DRIVE eröffnen und beenden

(Ergänzung zu Kapitel 3 von „DRIVE-Programmiersystem“ [1])

Dieses Kapitel beschreibt

- den Dialog-Ablauf im TIAM-Betrieb (ab Seite 14)
- den Dialog-Ablauf im UTM-Betrieb (ab Seite 20)
- wie Sie DRIVE mit einer BS2000-Prozedur aufrufen können (ab Seite 26)

3.1.1 Dialogablauf im TIAM-Betrieb

Ablaufschema für den Dialog mit DRIVE im TIAM-Betrieb

Namen in Kleinbuchstaben müssen durch die gültigen Namen in der jeweiligen Umgebung ersetzt werden.

`/LOGON userid,abrechnungsnr,'password'`	Einleiten eines BS2000-Prozesses

`/SET-FILE-LINK LINK-NAME=DRIVEOML,-`
`/  FILE-NAME=SYSLNK.DRIVE.022`
`/SET-FILE-LINK LINK-NAME=LIBOML,-`
`/  FILE-NAME=SYSPRG.DRIVE.022`
	DRIVE-Modulbibliotheken zuweisen

`/SET-FILE-LINK LINK-NAME=BLSLIB01,-` `/  FILE-NAME=crtelib`	CRTE-Laufzeitbibliothek zuweisen
`/SET-FILE-LINK LINK-NAME=BLSLIB02,-` `/  FILE-NAME=lmslib`	LMS-Laufzeitbibliothek zuweisen
`/SET-FILE-LINK LINK-NAME=BLSLIB03,-` `/  FILE-NAME=fhsmacrolib`	FHS-Laufzeitbibliothek zuweisen
`/SET-FILE-LINK LINK-NAME=BLSLIB04,-` `/  FILE-NAME=systemmacrolib`	Systembibliothek zuweisen (Bibliothek, die den Modul DCCOBRTS enthält)

`/SET-FILE-LINK LINK-NAME=SESAMOML,-`
`/  FILE-NAME=sesamlib`
`/SET-FILE-LINK LINK-NAME=SESCONF,-`
`/   FILE-NAME=sesamkonf`
	SESAM-Modulbibliothek und Konfiguration zuweisen
	(nur falls Sie mit einer SESAM-Datenbank arbeiten)

`/SET-FILE-LINK LINK-NAME=FORMOML,-` `/  FILE-NAME=formatlib`	Formatbibliothek zuweisen (nur falls Sie mit FHS-Formaten arbeiten)
`/SET-FILE-LINK LINK-NAME=MROUTLIB,-` `/  FILE-NAME=fhsrtslib`	FHS-Laufzeitbibliothek zuweisen (nur falls Sie mit FHS-Formaten arbeiten)
`/SET-FILE-LINK LINK-NAME=RSOML,-` `/  FILE-NAME=SYSLIB.DRIVE.022`	Bibliothek für den Reportgenerator zuweisen (nur falls Sie mit Reports arbeiten)
`/SET-FILE-LINK LINK-NAME=USEROML,-` `/  FILE-NAME=usrlib`	DRIVE-Bibliothek zuweisen (nicht notwendig, falls Sie die DRIVE-Bibliothek mit der DRIVE-Anweisung PARAMETER DYNAMIC LIBRARY zuweisen werden, siehe Abschnitt „Dialog parametrisieren“ auf Seite 18)

`/START-PROGRAM FROM-FILE=*MOD(-`
`/  LIB=objlib,ELEM=modulname,PROG-MO=ANY,-`
`/  RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,-`
`/              UN-EXTRNS=DELAY,LO-IN=REF))`
	generierte DRIVE-Fassung (LLM) aufrufen (siehe Seite 37).

`PARAMETER`	DRIVE-Parameter eingeben
im Dialog-Modus arbeiten:	

`DECLARE ... CURSOR`
`FETCH ...`
`INSERT ...`
`UPDATE ...`
	DRIVE-Anweisungen im Dialog-Modus

oder im Programm-Modus arbeiten:	
`DO programmname`	Datenbearbeitung mit DRIVE-Programmen
oder in den EDT verzweigen:	

`EDT`
`.`
`.`
`.`
`HALT`
	Erstellen von DRIVE-Programmen

oder im Debug-Modus arbeiten:	

`DEBUG programmname`
`.`
`.`
`.`
`BREAK DEBUG`
	DRIVE-Programme debuggen

`STOP`	Dialog mit DRIVE beenden
`/LOGOFF`	BS2000-Prozess beenden

3.1.1.1 Dialog eröffnen

- Leiten Sie zunächst mit dem BS2000-Kommando /LOGON ... einen BS2000-Prozess ein.

  ```
  /LOGON userid,abrechnungsnr,'password'
  ```

Das Betriebssystem BS2000 gibt eine Meldung aus über die Eröffnung des BS2000-Prozesses. Es erwartet weitere BS2000-Kommandos.

- Weisen Sie nun die Modulbibliothek, die die DRIVE-Bindemodule enthält, als DRIVE-Bindemodul-Bibliothek zu. Verwenden Sie dazu das BS2000-Kommando:

  ```
  /SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
  ```

DRIVE benötigt für seinen Ablauf interne DRIVE-Programme, deren Zwischencodes sich in der unter dem Namen SYSPRG.DRIVE.022 ausgelieferten Bibliothek befinden.

- Weisen Sie diese Bibliothek mit folgendem BS2000-Kommando zu:

  ```
  /SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
  ```

Damit offene Externverweise vom dynamischen Bindelader aufgelöst werden können, müssen Sie die Laufzeitbibliotheken von CRTE, LMS und FHS sowie die Systemmacrobibliothek zuweisen.

► Weisen Sie diese Bibliotheken mit folgenden BS2000-Kommandos zu:

```
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lmslib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhsmacrolib
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=systemmacrolib
```

Für die SESAM-Fassung benötigt DRIVE zum Ablaufzeitpunkt SESAM-Verbindungsmodule. Weisen Sie daher die Modulbibliothek, die die SESAM-Verbindungsmodule enthält, als SESAM-Bindemodul-Bibliothek zu.

► Verwenden Sie dazu das BS2000-Kommando:

```
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesamlib
```

► Weisen Sie nun die Konfiguration der SESAM-Datenbank zu, mit der Sie arbeiten wollen:

```
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesamkonf
```

Wenn Sie mit FHS-Formaten arbeiten, benötigt DRIVE die Laufzeitbibliothek von FHS und die Formatbibliothek mit den FHS-Formaten.

► Weisen Sie die Formatbibliothek mit folgenden BS2000-Kommando zu:

```
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=formatlib
```

und die Laufzeitbibliothek mit folgendem BS2000-Kommando:

```
/SET-FILE-LINK LINK-NAME=MROUTLIB,FILE-NAME=fhsrtslib
```

Diese Zuweisung der Laufzeitbibliothek ist nicht notwendig, wenn die Module aus der Anwenderdatei TASKLIB oder der Systemdatei $TSOS.TASKLIB geladen werden. (Zur Suchreihenfolge von FHS siehe „FHS“ [14].)

Wenn Sie mit Reports arbeiten, benötigt DRIVE die Bibliothek mit den Modulen für den Report-Generator.

► Weisen Sie diese Bibliothek mit folgendem BS2000-Kommando zu:

```
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
```

Wenn Sie Ihre Programm-Sourcen, COPY-Elemente, Übersetzungslisten, Zwischencode und Benutzerkennsätze in einer bestimmten Bibliothek abspeichern wollen, müssen Sie eine Benutzerbibliothek zuweisen.

► Weisen Sie sich eine eigene Benutzerbibliothek zu:

```
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usrlib
```

Bei Zuweisung dieser Benutzerbibliothek mit der DRIVE-Anweisung PARAMETER DYNAMIC LIBRARY ist obige Zuweisung nur notwendig, wenn benutzereigene Programme mit der DRIVE-Programmanweisung CALL MODULE aufgerufen werden.

Mit dem Kommando /START-PROGRAM rufen Sie die generierte DRIVE-Fassung auf. Mit Hilfe der Prozedur DRIPRC.INSTALL.DRIVE aus der Bibliothek SYSPRC.DRIVE.022 ist es möglich, dem generierten Bindelademodul von DRIVE einen beliebigen Namen zu geben (siehe Abschnitt „DRIVE für den TIAM-Betrieb generieren“ auf Seite 37). Dies ist insbesondere dann erforderlich, wenn unter einer Kennung mehrere Bindelademodule bereitgestellt werden.
Fragen Sie also gegebenenfalls Ihren Systemverwalter nach dem Namen des gewünschten Bindelademoduls.

► Starten Sie mit folgendem Kommando die generierte DRIVE-Fassung:

```
/START-PROGRAM FROM-FILE=*MOD(LIB=objlib,ELEM=modulname,PROG-MO=ANY,-
/  RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY,LO-IN=REF))
```

Die gewünschte DRIVE-Fassung wird nun aufgerufen, geladen und gestartet. Es folgt die Meldung, dass DRIVE geladen ist, und anschließend ein leerer Bildschirm mit einem Stern (*) als Bereit-Zeichen. DRIVE erwartet eine Eingabe.

3.1.1.2 Dialog parametrisieren

Durch die Anweisung PARAMETER ohne Zusatz wird auf dem Bildschirm folgendes ausgegeben:

```
PAR01                                  PARAMETER
_____________________________________________________________________________

  AUSWAHL DER PARAMETER-ANWEISUNG :


        ( ) PARAMETER STATIC

        ( ) PARAMETER DIAGNOSIS

        ( ) PARAMETER DYNAMIC

        ( ) PARAMETER KFKEY

        ( ) PARAMETER LOCK DIAOLG

        ( ) PARAMETER LOCK PROCEDURE



_____________________________________________________________________________
                                  ( A = ABBRECHEN ) ( )
LTG       EM:1                                                        TAST
```

Wenn Sie eine der Auswahlmöglichkeiten mit "X" markieren, verzweigen Sie in einen der sechs Folge-Bildschirme. Dorthin gelangen Sie auch direkt, wenn Sie PARAMETER mit dem entsprechenden Schlüsselwort eingeben, z.B. PAR STATIC.

Eine weitere Möglichkeit der Parametrisierung besteht darin, die einzelnen PARAMETER-Anweisungen direkt einzugeben, z.B. PARAMETER DYNAMIC LIBRARY= drivelib.

Sämtliche PARAMETER-Bildschirme sind bereits mit Werten vorbelegt, mit denen Sie arbeiten können. Wollen Sie jedoch mit anderen Werten arbeiten, so müssen Sie die Standardwerte überschreiben und anschließend DUE drücken.

Eine ausführliche Beschreibung von PARAMETER finden Sie im „DRIVE-Lexikon“ [3].

Beispiel

Sie wollen die Standard-Parametrisierung so verändern, dass Sie bei PARAMETER STATIC unter USER Ihren Namen angeben und bei PARAMETER DYNAMIC unter LIBRARY die DRIVE-Bibliothek PLAM.LIB.DRIVE.

Um diese Veränderungen vorzunehmen, können Sie sich entweder mit PARAMETER STATIC oder mit PARAMETER DYNAMIC die jeweiligen Bildschirme ausgeben lassen und die Standardwerte überschreiben. Oder Sie geben die folgenden Anweisungen direkt ein:

```
PARAMETER STATIC USER = 'MAIER'
PARAMETER DYNAMIC LIBRARY = "PLAM.LIB.DRIVE"
```

Zur Bestätigung der Parametrisierung gibt DRIVE in der Meldungszeile (unterste Bildschirmzeile) folgende Meldung aus:

```
% DRI0009 ANWEISUNG AUSGEFUEHRT
```

3.1.1.3 Dialog beenden

Den Dialog mit DRIVE beenden Sie durch die Anweisungen STOP oder EXIT.

Alle von DRIVE belegten Betriebsmittel werden freigegeben. Offene Transaktionen werden wie folgt behandelt:

STOP Das Beenden des Dialogs mit STOP ist nur möglich, wenn alle offenen Transaktionen geschlossen sind. Dies erreichen Sie mit COMMIT oder mit ROLLBACK WORK (siehe „DRIVE-SQL-Lexikon“ [4).

EXIT Die Anweisung EXIT beendet den Dialog mit DRIVE auch bei einer offenen Transaktion. Diese wird zurückgesetzt. Der Inhalt der EDT-Arbeitsdatei 0 wird nicht gesichert. EXIT ist nur im Dialog-Modus und im Programm-Modus nur bei Bildschirmeingabe in einem Programm erlaubt.

Nach Beenden des DRIVE-Dialogs mit STOP oder EXIT wird am Bildschirm folgende Meldung ausgegeben:

```
%   DRI0088 'xxxxxx' ORDNUNGSGEMAESS BEENDET
```

Das BS2000 beenden Sie mit dem Kommando /LOGOFF. Das BS2000 gibt daraufhin eine Meldung aus, dass der Prozess beendet ist.

3.1.2 Dialogablauf im UTM-Betrieb

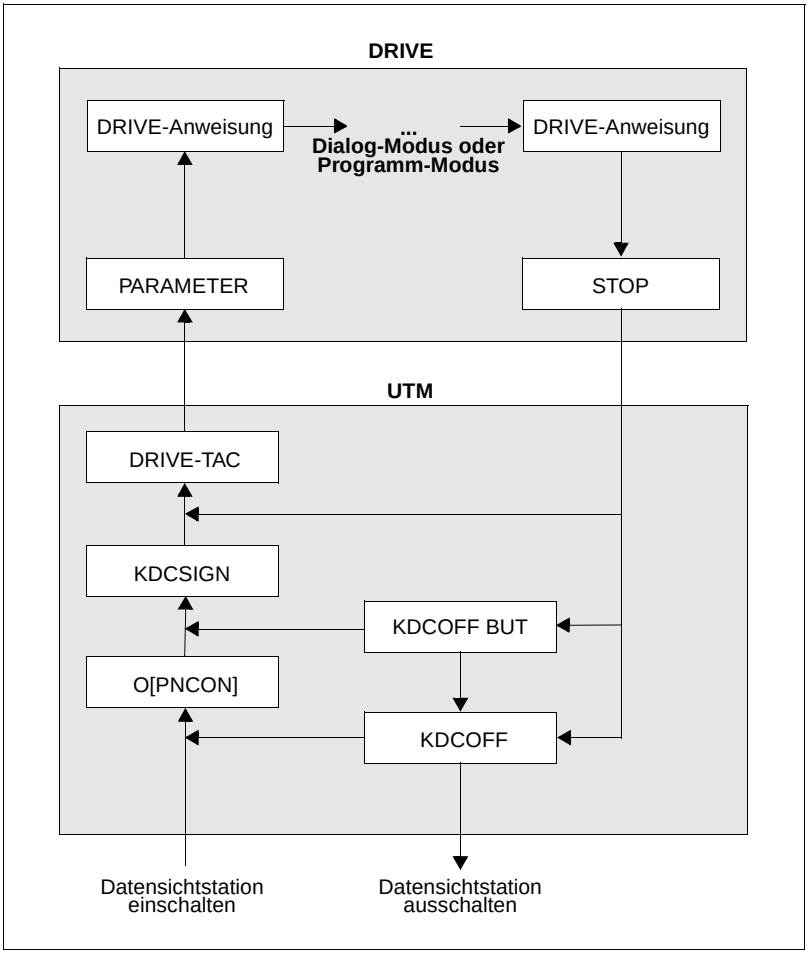
Im DRIVE-UTM-Betrieb erfolgt die Kommunikation mit DRIVE über den Universellen Transaktionsmonitor UTM. Das bedeutet,

– alle Daten, die Sie an der Datensichtstation eingeben, werden über UTM an DRIVE übermittelt.

– alle Daten, die DRIVE ausgibt, werden über UTM an Ihre Datensichtstation (oder an einen Drucker) übermittelt.

Bevor Sie also den Dialog mit DRIVE eröffnen, müssen Sie die Verbindung zu UTM herstellen. Entsprechend müssen Sie nach Beenden des DRIVE-Dialogs die Verbindung zu UTM lösen.

Das folgende Bild zeigt, welche Arbeitsschritte Sie vom Einschalten bis zum Ausschalten der Datensichtstation durchführen müssen, wenn Sie mit DRIVE im UTM-Betrieb arbeiten wollen:



Während des Dialogs mit DRIVE können Sie neben DRIVE-Anweisungen folgende UTM-Transaktionscodes (TACs) eingeben:

TAC	**Bedeutung**
KDCLAST	Wiederholen der letzten DRIVE-Ausgabe oder einer Asynchron-Nachricht
KDCOUT	Abrufen einer Asynchron-Nachricht
KDCDISP	Rekonstruktion des letzten Bildschirms nach dem Abrufen einer Asynchron-Nachricht
KDCOFF	Beenden des Dialogbetriebs. Es darf keine offene Transaktion mehr existieren. Kann auch von UTM automatisch abgesetzt werden, wenn bei Zeitüberschreitung der Dialog unterbrochen wird. Dies geschieht z.B. dann, wenn in einer offenen Transaktion längere Eingabepausen entstehen.

3.1.2.1 Dialog eröffnen

Das Eröffnen des Dialogs mit DRIVE gliedert sich in folgende Schritte:

– Verbindung zur UTM-Anwendung herstellen
– Bei der UTM-Anwendung anmelden
– DRIVE aufrufen

Verbindung zur UTM-Anwendung herstellen

Die Verbindung zur UTM-Anwendung stellen Sie mit folgender Anweisung her:

```
O[PNCON] anwendungsname[,pp/rr][PW=C'verbindungskennwort']
```

Dabei bedeutet:

anwendungsname	Name der UTM-Anwendung.
pp/rr	Prozessor- und Regionsnummer des Verarbeitungsrechners, auf dem die UTM-Anwendung abläuft.
verbindungskennwort	Nur erforderlich, falls vom Systemverwalter beim Start der UTM-Anwendung ein Verbindungskennwort angegeben wurde.

Nach erfolgreichem Verbindungsaufbau wird am Bildschirm folgende Meldung ausgegeben:

```
K002 VERBUNDEN MIT ANWENDUNG anwendungsname – BITTE KDCSIGN
```

Anmelden bei der UTM-Anwendung

Sie melden sich bei der UTM-Anwendung an, indem Sie das KDCSIGN-Kommando eingeben:

```
KDCSIGN benutzerkennung[,kennwort]
```

Dabei bedeutet:

benutzerkennung	UTM-Benutzerkennung.
kennwort	Kennwort, das der UTM-Benutzerkennung vom Systemverwalter zugeteilt wurde.

Nach einer ordnungsgemäßen KDCSIGN-Eingabe wird am Bildschirm folgende Meldung ausgegeben:

```
K008 KDCSIGN AKZEPTIERT – BITTE EINGABE
```

DRIVE aufrufen

Sie starten DRIVE, indem Sie den UTM-Transaktionscode (TAC) für DRIVE eingeben. Der vorgegebene UTM-Transaktionscode lautet DRISQL. Er kann jedoch bei der Generierung der UTM-Anwendung vom Systemverwalter verändert werden. Fragen Sie also gegebenfalls Ihren Systemverwalter nach dem gültigen UTM-Transaktionscode für DRIVE.

Nachdem Sie den Transaktionscode für DRIVE eingegeben haben, gibt DRIVE folgende Meldung aus: `% DRI0008 BITTE ANWEISUNG EINGEBEN`

DRIVE erwartet nun die Eingabe von DRIVE-Anweisungen.

Die oben angegebene Meldung (DRI0008) wird nur ausgegeben, wenn bei den UTM-Startparametern PERMIT auf OFF gesetzt wurde. Ist dies nicht der Fall, wird bei einer Mischbetriebs-Anwendung der PERMIT-Bildschirm ausgegeben.

3.1.2.2 Dialog parametrisieren

Parametrisiert wird DRIVE im UTM-Betrieb auf die gleiche Weise wie im TIAM-Betrieb (siehe Abschnitt „Dialogablauf im TIAM-Betrieb“ auf Seite 14). Im UTM-Betrieb werden jedoch einige DRIVE-Parameter bereits beim Starten der UTM-Anwendung vom Systemverwalter festgelegt (siehe Abschnitt „Startprozedur“ auf Seite 73). Darüber hinaus können im DRIVE-UTM-Betrieb durch Vorlaufprogramme anwenderspezifische DRIVE-Parameter an DRIVE übergeben werden (siehe Handbuch „DRIVE-Programmiersystem“ [1], Abschnitt „Datenschutz im TIAM-Betrieb“).

i PARAMETER STATIC-Operanden können nur einmal belegt werden. Dabei spielt es keine Rolle, ob der Operand in der UTM-Startprozedur, in einem Vorlaufprogramm oder im Dialog belegt wurde.

Die PARAMETER STATIC-Operanden FIRSTPAGE und LASTPAGE können nur in der UTM-Startprozedur als DRIVE-Startparameter belegt werden. Diese Operanden können Sie im DRIVE-Dialog nicht verändern.

Es gibt PARAMETER-Operanden, z.B. die PARAMETER DYNAMIC-Operanden, die innerhalb der DRIVE-Sitzung beliebig geändert werden können.

Wenn Sie gesetzte PARAMETER-Werte abfragen oder PARAMETER-Operanden neu versorgen wollen, geben Sie die Anweisung PARAMETER mit dem entsprechenden Schlüsselwort ein, z.B.:

```
PARAMETER STATIC
```

Das nun ausgegebene Bildschirmformat unterscheidet sich von dem PARAMETER STATIC-Bildschirmformat im TIAM-Betrieb, da hier andere Operanden von Belang sind.

In den PARAMETER-Bildschirmformaten sind die Operanden folgendermaßen dargestellt:

– modifizierbare Operanden-Werte: hell

– nicht-modifizierbare Operanden-Werte: halbhell

Da nur die Operanden-Werte von PARAMETER STATIC während eines DRIVE-Laufs nicht zu ändern sind, erscheinen nur diese halbhell auf dem Bildschirm. Bei den übrigen PARAMETER-Bildschirmformaten erscheinen die Werte dagegen hell.

Auf PARAMETER LOCK-Masken erscheint die Angabe ON ebenfalls halbhell, da eine gesperrte Anweisung während eines DRIVE-Laufes nicht wieder entsperrt werden kann.
Auf PARAMETER KFKEY-Masken erscheinen alle Werte halbhell, da nur in der UTM-Startprozedur K/F-Tasten belegt werden können.

3.1.2.3 Dialog beenden

Die Beendigung des Dialogs gliedert sich in zwei Schritte:

– Dialogende mit DRIVE

– Dialogende mit UTM

Dialogende mit DRIVE

Den Dialog mit DRIVE beenden Sie durch die Anweisungen STOP oder EXIT. Alle von DRIVE belegten Betriebsmittel werden freigegeben. Offene Transaktionen werden wie folgt behandelt:

STOP Das Beenden des Dialogs mit STOP ist nur möglich, wenn alle offenen Transaktionen geschlossen sind. Dies erreichen Sie mit COMMIT oder mit ROLLBACK WORK.
Die Angabe von STOP ohne das Schlüsselwort WITH im UTM-Betrieb bewirkt einen PEND FI. Der Vorgang und die Transaktion sind beendet.

EXIT Die Anweisung EXIT beendet den Dialog mit DRIVE auch bei einer offenen Transaktion. Diese wird zurückgesetzt.

Hinweise zum Beenden des Dialogs mit STOP

Im Programm-Modus können Sie bei der STOP-Anweisung einen Folge-TAC angeben:

```
STOP WITH folgetac
```

folgetac ist als Literal oder im Programm als Variable einzugeben. Es bewirkt einen PEND FC. Der DRIVE-Vorgang und die Transaktion sind beendet, der Dialogschritt soll im angeketteten Vorgang fortgesetzt werden.

Wollen Sie ein DRIVE-Programm mit STOP beenden und dazu ein bestimmtes Bildschirmformat ausgeben, so können Sie dies mit der STOP-Anweisung bewirken:

```
STOP WITH DISPLAY screenformat
```

screenformat ist der Name eines FHS-Formats (siehe „DRIVE-Programmiersprache“ [2]).

Ein mit DRIVE-Mitteln erstelltes Format können Sie mit folgender Anweisung ausgeben:

```
STOP WITH DISPLAY FORM formatname
```

Die Erstellung von Bildschirmformaten (DRIVE- und FHS-Formate) ist in DRIVE-Programmiersprache [2] beschrieben.

Nach STOP ohne DISPLAY meldet sich UTM folgendermaßen:

```
BITTE TRANSAKTIONSCODE EINGEBEN:
```

Dialogende mit UTM

Durch die Eingabe des Transaktionscodes KDCOFF melden Sie sich von der UTM-Anwendung ab. Die Verbindung zum Verarbeitungsrechner wird abgebaut. Am Bildschirm wird eine Bestätigung ausgegeben.

3.1.3 DRIVE-Aufruf mit BS2000-Prozeduren

Im TIAM-Betrieb können Sie DRIVE auch innerhalb von BS2000-Dialog- oder Batch-Prozeduren starten.

3.1.3.1 Dialog-Prozedur

Bearbeitungsvorgänge, die beim Starten von DRIVE häufig in ähnlicher Form abgearbeitet werden sollen, können Sie automatisieren. Dazu rufen Sie DRIVE in einer BS2000-Dialog-Prozedur auf.

In einer BS2000-Dialog-Prozedur können Sie beispielsweise

– Modulbibliotheken zuweisen
– DRIVE aufrufen
– PARAMETER-Operanden belegen
– DRIVE-Programme starten etc.

Bei Bedarf können Sie den Prozedurablauf über BS2000-Prozedur-Variablen steuern und am Bildschirm überwachen.

Die Anweisung DEBUG in einer BS2000-Dialog-Prozedur bewirkt, dass in den Debug-Modus verzweigt wird (siehe „DRIVE-Lexikon“ [3]). Sie können nun in gewohnter Weise eine Debug-Sitzung führen. Die DEBUG-Anweisung sollte die letzte Anweisung vor der Anweisung END-PROCEDURE sein, weil nach Beenden der Debug-Sitzung in den DRIVE-Dialog-Modus verzweigt wird und keine weiteren BS2000-Anweisungen aus der Prozedur gelesen werden.

Debug-Anweisungen sind in einer BS2000-Dialog-Prozedur nicht zugelassen.

Beispiel

Sie wollen DRIVE mit einer BS2000-Dialog-Prozedur starten.

Namen in Kleinbuchstaben müssen durch die gültigen Namen in der jeweiligen Umgebung ersetzt werden. Wollen Sie beispielsweise Unterprogramme in anderen Programmiersprachen benutzen, so müssen Sie die Prozedur entsprechend erweitern.

```
/BEGIN-PROC A
/SET-FILE-LINK LINK-NAME=DRIVEOML,-
/  FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,-
```

```
/  FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,-
/  FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,-
/  FILE-NAME=lmslib
/SET-FILE-LINK LINK-NAME=BLSLIB03,-
/  FILE-NAME=fhsmacrolib
/SET-FILE-LINK LINK-NAME=BLSLIB04,-
/  FILE-NAME=systemmacrolib
/SET-FILE-LINK LINK-NAME=SESAMOML,-
/  FILE-NAME=sesamlib
/SET-FILE-LINK LINK-NAME=SESCONF,-
/  FILE-NAME=sesamkonf
/SET-FILE-LINK LINK-NAME=FORMOML,-
/  FILE-NAME=formatlib
/SET-FILE-LINK LINK-NAME=MROUTLIB,-
/  FILE-NAME=fhsrtslib
/SET-FILE-LINK LINK-NAME=RSOML,-
/  FILE-NAME=SYSLIB.DRIVE.022
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROGRAM FROM-FILE=-
/  *MOD(LIB=objlib,ELEM=modulname,-
/  PROG-MO=ANY,RUN-MO=ADV(ALT-LIB=YES,-
/  NAME-COL=ABORT,UN-EXTRNS=DELAY,-
/  LO-IN=REF))
/END-PROC
```

Generierte DRIVE-Fassung (LLM) aufrufen. Der LLM-Name und der Bibliotheksname wurden bei der Generierung festgelegt. (Systemverwalter fragen!)

DRIVE erwartet nun Eingaben vom Datensichtgerät.

Soll DRIVE bestimmte Anweisungen automatisch ausführen, z.B. parametrisieren oder DRIVE-Programme aufrufen, so können Sie diese Anweisungen ebenfalls in der BS2000-Dialog-Prozedur angeben. Dies erfolgt dann nach dem START-PROG-Kommando ohne Schrägstrich, da es sich nicht um BS2000-Kommandos handelt. Jede im DRIVE-Dialog-Modus zulässige Anweisung ist hier möglich. Sind in einer BS2000-Dialog-Prozedur fehlerhafte DRIVE-Anweisungen enthalten, gibt DRIVE eine entsprechende Fehlermeldung aus. Alle weiteren Eingaben erwartet DRIVE dann vom Datensichtgerät.

Bei fehlerfreiem Ablauf liest DRIVE alle Eingaben der BS2000-Dialog-Prozedur. Nach dem Verarbeiten der letzten DRIVE-Anweisung aus der Prozedur geschieht folgendes:

– DRIVE wird beendet, wenn die letzte Anweisung in der Prozedur STOP war, oder

– Sie können mit DRIVE weiterarbeiten (kein STOP).

3.1.3.2 Batch-Prozedur

Langlaufende Bearbeitungsvorgänge können Sie automatisieren und als BS2000-Stapel-Prozess ablaufen lassen, indem Sie DRIVE in einer BS2000-Batch-Prozedur aufrufen.

Beispiel

Das langlaufende DRIVE-Programm MONATSSTAT soll im Hintergrund ablaufen. Der Name der SESAM-Datenbank, des Schemas und der Authorization müssen bekanntgegeben werden.

Namen in Kleinbuchstaben müssen durch die gültigen Namen in der jeweiligen Umgebung ersetzt werden.

```
/LOGON
/SET-FILE-LINK LINK-NAME=DRIVEOML,-
/  FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,-
/  FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,-
/  FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,-
/  FILE-NAME=lmslib
/SET-FILE-LINK LINK-NAME=BLSLIB03,-
/  FILE-NAME=fhsmacrolib
/SET-FILE-LINK LINK-NAME=BLSLIB04,-
/  FILE-NAME=systemmacrolib
/SET-FILE-LINK LINK-NAME=SESAMOML,-
/  FILE-NAME=sesamlib
/SET-FILE-LINK LINK-NAME=SESCONF,-
/  FILE-NAME=sesamkonf
/SET-FILE-LINK LINK-NAME=RSOML,-
/  FILE-NAME=SYSLIB.DRIVE.022
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROGRAM FROM-FILE=-
/  *MOD(LIB=objlib,ELEM=modulname,-
/  PROG-MO=ANY,RUN-MO=ADV(ALT-LIB=YES,-
/  NAME-COL=ABORT,UN-EXTRNS=DELAY,-
/  LO-IN=REF))
PARAMETER DYNAMIC LIBRARY=...
PARAMETER DYNAMIC SCHEMA=...
PARAMETER DYNAMIC CATALOG=...
PARAMETER DYNAMIC AUTHORIZATION=...
DO MONATSSTAT
STOP
/LOGOFF
```

Generierte DRIVE-Fassung (LLM) aufrufen. Der LLM-Name und der Bibliotheksname wurden bei der Generierung festgelegt. (Systemverwalter fragen!)

FHS-Formate dürfen hier von DRIVE nicht verwendet werden. Sie führen zum Programmabbruch. Alle anderen Ausgaben werden auf SYSOUT geschrieben.

Ausgaben auf Drucker (SYSLST) oder Ausgaben in eine Datei sind möglich. Die Ausgabedatei müssen Sie innerhalb der BS2000-Batch-Prozedur mit dem Kommando /ASSIGN-SYSLST=datei zuweisen.

Bei Erreichen von END OF FILE (EOF) auf SYSDTA wird der ENTER-Prozess abgebrochen.

Wird in einer BS2000-Batch-Prozedur DRIVE aufgerufen, muss der entsprechende DBH geladen sein, wenn Datenbankzugriffe erfolgen sollen. Ansonsten wird die BS2000-Batch-Prozedur in einen Wartezustand versetzt. In diesem Fall müssen Sie den BS2000-ENTER-Prozess abbrechen und, nachdem der DBH geladen wurde, neu starten.

Über SYSDTA können Eingaben gelesen werden. Dies kann z.B. bei der Arbeit mit automatisierten Testfällen genutzt werden.

3.2 DRIVE-Einsatz vorbereiten

(Änderungen zum Kapitel 13 von „DRIVE-Programmiersystem“ [1])

In diesem Kapitel werden vorbereitende Arbeiten für den DRIVE-Einsatz beschrieben.

Modulbibliotheken müssen Sie zuweisen. Außerdem müssen Sie eine DRIVE-Bibliothek einrichten und ebenfalls zuweisen. Alle anderen Arbeitsschritte sind wahlweise, die Sie nur ausführen müssen, wenn Sie sie für Ihre jeweilige Umgebung benötigen.

Dieses Kapitel wendet sich vornehmlich an den DRIVE-Administrator. Es setzt neben grundlegenden Kenntnissen über das Betriebssystem BS2000 auch Kenntnisse über die Adressraumverwaltung des BS2000 voraus.

In diesem Kapitel werden die erforderlichen Tätigkeiten beschrieben, um:

– den Speicherplatzbedarf von DRIVE zu minimieren (ab Seite 30)
– Modulbibliotheken zuzuweisen (ab Seite 32)
– Bibliotheken für DRIVE-Programme, COPY-Elemente, Benutzerkennsätze, Zwischencode und Übersetzungslisten einzurichten (ab Seite 33)
– Komponenten zur DRIVE-Protokollierung bereitzustellen (ab Seite 34)
– eine zentrale Druckdatei für den UTM-Betrieb bereitzustellen (ab Seite 35)
– eine Diagnosedatei bereitzustellen (ab Seite 36)
– den DRIVE-Dialog in einer gewünschten Sprache (Deutsch oder Englisch) festzulegen (ab Seite 36).

3.2.1 DRIVE-Module als Shared-Code laden

Wird DRIVE standardmäßig geladen, so sind bei mehreren parallel arbeitenden DRIVE-TIAM-Anwendern oder bei mehreren generierten DRIVE-UTM-Tasks mehrere identische DRIVE-Module im Klasse-6-Speicher vorhanden. Der Klasse-6-Speicher wird damit unnötig belastet.
Um dies zu verhindern, können Sie den zentralen DRIVE-Modul DRILLM22 als Shared-Code in Klasse-3/4-Speicher laden (Subsystemname = DRIVE22).

Das Laden von Shared-Code erfolgt - als Subsystem - mit Hilfe der Dynamischen Subsystem Verwaltung des BS2000 (DSSM) (siehe „BS2000 Systeminstallation“ [17]).

Um das Subsystem DRIVE22 einzusetzen, müssen Sie:
– das Subsystem in den BS2000-Subsystemkatalog eintragen
– das Subsystem laden (und entladen)

Für den Shared-Code-Einsatz im **Old-Style-Betrieb** muss das Subsystem für den Old-Style-Betrieb geladen werden (Subsystemname = DRIVE).

Für den Shared-Code-Einsatz im **Mischbetrieb** müssen die Subsysteme für den New-Style- und für den Old-Style-Betrieb geladen werden.

3.2.1.1 Subsystem in den Subsystemkatalog eintragen

Das Subsystem DRIVE22 ist in einer Objektdatei definiert, die mit SSCM (Static Subsystem Catalog Manager) erzeugt wurde. Diese Objektdatei müssen Sie in den Subsystemkatalog eintragen, sie hat den Namen SYSSSC.DRIVE.022.

Beispiel

Der vorhandene Subsystemkatalog "sskat" wird mit SSCM um die Objektdatei SYSSSC.DRIVE.022 erweitert.

```
...
/START-SSCM
//START-CATALOG-MOIFICATION CATALOG-NAME=sskat
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSC.DRIVE.022
//CHECK-CATALOG CATALOG-NAME=*CURRENT
//SAVE-CATALOG CATALOG-NAME=*CURRENT
//END
...
```

3.2.1.2 Subsystem laden und entladen

Um das Subsystem DRIVE22 während des laufenden BS2000-Betriebes als Shared-Code zu laden, steht Ihnen das BS2000-Systemverwalter-Kommando /CREATE-SS zur Verfügung:

```
/CREATE-SS SS-NAME=DRIVE22
```

Als Shared-Code geladene Subsysteme können Sie während des laufenden BS2000-Betriebes auch wieder entladen. Voraussetzung dafür ist allerdings, dass kein DRIVE-TIAM-Anwender damit arbeitet und keine DRIVE-UTM-Task an das Subsystem angekoppelt ist. Zum Entladen des Subsystems steht Ihnen das BS2000-Systemverwalter-Kommando /DELETE-SS zur Verfügung:

```
/DELETE-SS SS-NAME=DRIVE22
```

3.2.2 Modulbibliotheken zuweisen

Die folgende Tabelle zeigt, welche Modulbibliotheken und Dateien für eine laufende DRIVE-Session benötigt werden können. Namen in Kleinbuchstaben müssen durch die gültigen Namen in der jeweiligen Umgebung ersetzt werden.

Bibliothek / Datei enthält	**Name**	**Link-Name**
DRIVE-Module *	SYSLNK.DRIVE.022	DRIVEOML
DRIVE-Systemprogramme	SYSPRG.DRIVE.022	LIBOML
Anwendereigene Programmbibliothek *	usrlib	USEROML
SESAM-Module	sesamlib	SESAMOML
SESAM-Konfigurationsdatei	sesamkonf	SESCONF
Bibliothek für den Reportgenerator	SYSLIB.DRIVE.022	RSOML
Anwendereigene FHS-Formatbibliothek	formatlib	FORMOML
FHS-Module	fhslrtslib	MROUTLIB
Laufzeitbibliotheken zur Auflösung von offenen Externverweisen durch den dynamischen Bindelader	crtelib	BLSLIB01
	lmslib	BLSLIB02
	fhsmacrolib	BLSLIB03
	systemmacrolib (z.B. $TSOS.SYSLIB.TIAM.xxx)	BLSLIB04

In einer laufenden DRIVE-Session lädt DRIVE die jeweils benötigten Module dynamisch nach. Dazu durchsucht DRIVE die Modulbibliotheken in der unten angegebenen Reihenfolge. Die Module, die DRIVE dynamisch nachlädt, sind mit "*" gekennzeichnet.

Suchreihenfolge:

1. Modulbibliothek mit Dateikettungsnamen (= Link-Name)
2. Modulbibliothek, die mit /SET-TASKLIB ... als Bindemodul-Bibliothek zugewiesen wurde (Standardzuweisung: $TSOS.TASKLIB)
3. Bibliothek SYSLNK.DRIVE.022 auf der Kennung, von der aus DRIVE gestartet wurde
4. Bibliothek SYSLNK.DRIVE.022 der BS2000-Default-User-Identification
5. TASKLIB der BS2000-Default-User-Identification.

Die Suchreihenfolge beim Zugriff auf DRIVE-Systemprogramme ist folgende:

1. Modulbibliothek mit Dateikettungsnamen LIBOML
2. SYSPRG.DRIVE.022 auf der Kennung, von der aus DRIVE gestartet wurde.

Das Nachladen von Nicht-DRIVE-Modulen (SESAM- und FHS-Module) wird durch die Link-Strategie des jeweiligen Produkts bestimmt.

3.2.3 DRIVE-Bibliothek einrichten

DRIVE-Programme, COPY-Elemente, Benutzerkennsätze, Zwischencode und Übersetzungslisten werden in DRIVE-Bibliotheken abgespeichert und verwaltet. Als Bibliothek wird eine PLAM-Bibliothek verwendet, die als DRIVE-Bibliothek zugewiesen wird (siehe „DRIVE-Programmiersystem“ [1], Kapitel „DRIVE-Elemente in PLAM-Bibliotheken verwalten“).

PLAM-Bibliothek einrichten

Eine PLAM-Bibliothek richten Sie mit dem Software-Produkt LIBRARY MAINTENANCE SYSTEM (LMS) ein, siehe „LMS“ [15], z.B.:

```
/START-PROGRAM $LMS
$LIB DRI.PLAM, NEW
$END
```

LMS richtet eine PLAM-Bibliothek unter dem Namen DRI.PLAM ein.

DRIVE-Bibliothek zuweisen

Es gibt zwei Möglichkeiten, eine PLAM-Bibliothek als DRIVE-Bibliothek zuzuweisen:

– mit der DRIVE-Anweisung PARAMETER DYNAMIC LIBRARY=...
– mit der File-Zuweisung /SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=...

Wenn Sie mit PARAMETER DYNAMIC LIBRARY=... eine nicht existierende PLAM-Bibliothek als DRIVE-Bibliothek zuweisen, gibt DRIVE eine Fehlermeldung aus.

3.2.4 Komponenten zur Dialog-Protokollierung bereitstellen

Wahlweise protokolliert DRIVE Ein- und Ausgaben während einer DRIVE-Session in eine Protokolldatei. Diese DRIVE-Protokollierung führt das Hilfsprogramm SYSPRG.DRIVE.022.DRILOG durch. Das Programm schreibt die zu protokollierenden Daten in die jeweilige Protokolldatei.

SYSPRG.DRIVE.022.DRILOG wird als BS2000-Batch-Prozess gestartet. Unabhängig von der Zahl der parallel mit DRIVE arbeitenden TIAM-Anwender oder der Zahl der generierten DRIVE-UTM-Tasks, wird SYSPRG.DRIVE.022.DRILOG nur einmal geladen.

Erforderliche Dateien

Folgende Dateien zur DRIVE-Protokollierung werden ausgeliefert:

SYSENT.DRIVE.022.DRILOG
: Diese Datei enthält folgende BS2000-Batch-Prozedur:

```
/LOGON
/START-PROGRAM SYSPRG.DRIVE.022.DRILOG
/LOGOFF
```

: Mit Hilfe dieser BS2000-Batch-Prozedur wird das Programm SYSPRG.DRIVE.022.DRILOG als Batchprozess gestartet.

SYSPRG.DRIVE.022.DRILOG
: Programm, das die Protokollierung abwickelt.

SYSPRG.DRIVE.022.DRILOGP
: Programm zum Aufbereiten und Ausdrucken der Protokolldatei.

SYSPRG.DRIVE.022.DRIENDE
: Programm zum Beenden des Programms SYSPRG.DRIVE.022.DRILOG.

Zur Protokollierung des DRIVE-Dialogs im UTM-Betrieb müssen Sie diese Dateien unter der Kennung bereitstellen, unter der die DRIVE-UTM-Anwendung gestartet wird.

Starten des Programms SYSPRG.DRIVE.022.DRILOG

Um das Programm SYSPRG.DRIVE.022.DRILOG als BS2000 Batch-Prozess zu starten, haben Sie zwei Möglichkeiten:

– BS2000-Kommando /ENTER-JOB SYSENT.DRIVE.022.DRILOG
(nicht möglich für DRIVE-UTM)

– Batch-Prozess starten durch Einschalten der DRIVE-Protokollierung über die DRIVE-Anweisung PARAMETER DYNAMIC (siehe „DRIVE-Programmiersystem“ [1], Kapitel „DRIVE-Dialog protokollieren“).

  Voraussetzung:

  Die Datei SYSENT.DRIVE.022.DRILOG steht unter der Kennung zur Verfügung, von der aus DRIVE aufgerufen wurde (TIAM-Betrieb) oder von der aus die DRIVE-UTM-Anwendung gestartet wurde (UTM-Betrieb).

  Wird beim Einschalten der Protokollierung die Meldung DRILOG NICHT GELADEN ausgegeben, dann konnte der Batch-Prozess nicht gestartet werden. In diesem Fall erfolgt keine Protokollierung.

Beenden des Programms SYSPRG.DRIVE.022.DRILOG

Der Batch-Prozess SYSENT.DRIVE.022.DRILOG kann beendet werden:

– mit dem Programm SYSPRG.DRIVE.022.DRIENDE:
  `/START-PROGRAM SYSPRG.DRIVE.022.DRIENDE`

– durch den Operator, nachdem am Bedienplatz die Meldung
  `DRILOG KEIN ANWENDER MEHR? DRILOGP NUR BEENDEN MIT /INTR tsn,STOP`
  ausgegeben wurde.

3.2.5 Zentrale Druckdatei (LIST-Datei) für den UTM-Betrieb bereitstellen

Alle im UTM-Betrieb angestoßenen Druckausgaben werden in der zentralen Druckdatei zwischengespeichert. Diese Druckdatei können Sie mit folgenden Dateieigenschaften einrichten:

```
LINK-NAME      = DRILIST
ACCESS-METHOD  = ISAM
RECORD-FORMAT  = V
BUFFER-LENGTH  = STD(SIZE=b)    (b kann max. 16 sein)
SPACE          = REL(nn,nn)
KEY-POSITION   = 5
KEY-LENGTH     = 24
```

Findet DRIVE keine Datei mit dem Dateikettungsnamen DRILIST, so richtet DRIVE bei Bedarf selbst eine Druckdatei mit den genannten Dateieigenschaften ein unter dem Namen DRI.LIST.FILE. Es werden die Werte BUFFER-LENGTH = STD(SIZE=16), SPACE = REL(33,16) eingetragen.

3.2.6 Diagnosedatei (INTTRACE-Datei) bereitstellen

Die Druckausgaben, die durch den Parameter DIAGNOSIS erzeugt werden (siehe „DRIVE-Lexikon“ [3], Anweisung PARAMETER), werden in die INTTRACE-Datei geschrieben.

Als INTTRACE-Datei können Sie eine Datei mit folgenden Dateieigenschaften einrichten:

```
LINK-NAME       = INTTRACE
ACCESS-METHOD   = ISAM
RECORD-FORMAT   = V
BUFFER-LENGTH   = STD(SIZE=16)
SPACE           = REL(33,16)
KEY-POSITION    = 5
KEY-LENGTH      = 32
OPEN-MODE       = INOUT
SHARED-UPDATE   = YES
```

Findet DRIVE keine Datei mit dem Dateikettungsnamen INTTRACE, so richtet DRIVE bei Bedarf selbst eine Datei unter dem Namen DRI.INTTRACE.FILE mit den genannten Dateieigenschaften ein. Es werden die Werte BUFFER-LENGTH = STD(SIZE=16), SPACE = REL(33,16) eingetragen.

3.2.7 Sprache für den DRIVE-Dialog auswählen

Die Sprache der Ausgaben von DRIVE kann durch den Anwender festgelegt werden.

Die DRIVE-Meldungen werden standardmäßig in Englisch ausgegeben.

Wenn Sie die DRIVE-Meldungen auf Deutsch umstellen wollen, geben Sie im BS2000-Systemmodus folgendes Kommando ein:

```
/MODIFY-MSG-ATTRIBUTES,TASK-LANGUAGE=D
```

3.3 DRIVE für den TIAM-Betrieb generieren

(Änderungen zum Kapitel 14 von „DRIVE-Programmiersystem“ [1])

Zum Generieren von DRIVE steht Ihnen die Prozedur DRIPRC.INSTALL.DRIVE in der PLAM-Bibliothek SYSPRC.DRIVE.022 zur Verfügung. Abhängig von den einzugebenden Parameterwerten bindet diese BS2000-Prozedur die für die jeweilige DRIVE-Fassung erforderlichen DRIVE-Module zu einem Bindelademodul (LLM) zusammen.

Rufen Sie die Prozedur mit folgendem BS2000-Kommando auf:

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)
```

Geben Sie im Prozedurablauf folgende Parameterwerte ein, um eine DRIVE-Fassung für den TIAM-Betrieb zu generieren:

`&BETRIEB = TIAM`

`&STYLE = NEW`

`&OBJLIB = objlib` Geben Sie für *objlib* die Modulbibliothek an, in der der Binder das LLM *modulname* ablegen soll. Diese Modulbibliothek sollte nicht die Bibliothek SYSLNK.DRIVE.022 sein.

`&EDTLIB = edtlib` Geben Sie für *edtlib* die Modulbibliothek an, in der sich das EDT-Anschlussmodul IEDTGLE befindet.

`&DRIVELIB = drivelib` Geben Sie für *drivelib* die Modulbibliothek mit den DRIVE-Modulen an.

`&BINDER = binder` Geben Sie für *binder* den Dateinamen des Binders an.

`&STARTLLM = modulname`

Geben Sie für *modulname* den Namen des LLMs an, zu dem die generierte DRIVE-Fassung gebunden wird.
Dieser Name ist beim Starten im /START-PROG-Kommando anzugeben (siehe Abschnitt „Dialog-Prozedur“ auf Seite 26 und Abschnitt „Batch-Prozedur“ auf Seite 28).

modulname darf max. 32 Zeichen lang sein.

`&SESAMLIB = sesamlib` Geben Sie für *sesamlib* die Modulbibliothek mit den SESAM-Modulen an.

Beispiel

Sie generieren eine DRIVE-New-Style-Fassung für den Zugriff auf SESAM.

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)-
/  PROC-PARAM=(STYLE=NEW,BETRIEB=TIAM,OBJLIB=MYOBJLIB.DRIVE22,-
/               DRIVELIB=SYSLNK.DRIVE.022,-
/               BINDER=$TSOS.BINDER,STARTLLM=DRISES,-
/               SESAMLIB=$TSOS.SYSLNK.SESAMSQL.022)
```

Mischbetrieb

Um eine DRIVE-Fassung für den Mischbetrieb zu generieren, rufen Sie ebenfalls die Prozedur DRIPRC.INSTALL.DRIVE auf (siehe Seite 37). Dabei geben Sie die selben Parameter an wie für die oben beschriebene New-Style-Fassung.

3.3.1 Besonderheiten für den Old-Style-Betrieb

Um eine DRIVE-Fassung für den Old-Style-Betrieb zu generieren, rufen Sie ebenfalls die Prozedur DRIPRC.INSTALL.DRIVE auf (siehe Seite 37).

Die einzigen Unterschiede sind, dass Sie für die Parameter &STYLE und &FASSUNG die folgenden Parameterwerte angeben müssen:

`&STYLE = OLD`

`&FASSUNG = [ SESAM | LEASY | DMS ]`
: Geben Sie das Datenhaltungssystem ein, auf das Sie mit DRIVE zugreifen.

Außerdem müssen Sie die Parameter &PHASE, &TSOSLNK und ggf. &LEASYLIB angeben:

`&PHASE = modulname`
: Geben Sie für *modulname* den Namen der Phase an, zu dem die generierte DRIVE-Fassung gebunden wird.
Dieser Name ist beim Starten im /START-PROG-Kommando anzugeben (siehe Abschnitte „Dialog-Prozedur“ auf Seite 26 und „Batch-Prozedur“ auf Seite 28).

`&TSOSLNK = tsoslnkname`
: Geben Sie für *tsoslnkname* den Dateinamen des statischen Binders TSOSLNK an.
Der Standardname ist: $TSOS.TSOSLNK.

`&LEASYLIB = leasylib` Geben Sie für *leasylib* die Modulbibliothek mit den LEASY-Modulen an.
Nur erforderlich, wenn Sie &FASSUNG=LEASY angegeben haben.
Der Standardname ist: $TSOS.LEA.OML.

Die Beschreibung der weiteren Parameter finden Sie auf Seite 37. Die Parameter &BINDER, &STARTLLM und &OBJLIB entfallen.

Beispiel

Sie generieren eine DRIVE-Old-Style-Fassung für den Zugriff auf LEASY.

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)-
/  PROC-PARAM=(STYLE=OLD,BETRIEB=TIAM,FASSUNG=LEASY,-
/               DRIVELIB=SYSLNK.DRIVE.022,PHASE=DRILEA,-
/               TSOSLNK=$TSOS.TSOSLNK,-
/               LEASYLIB=$TSOS.LEA.OML)
```

3.4 DRIVE für den UTM-Betrieb generieren

(Änderung zu Kapitel 15 von „DRIVE-Programmiersystem“ [1])

Dieses Kapitel beschreibt

– allgemein die Generierung von DRIVE für den UTM-Betrieb (ab Seite 40)

– wie Sie die Generierung für den UTM-Betrieb vorbereiten (ab Seite 42)

– wie Sie DRIVE für den UTM-Betrieb generieren (ab Seite 52)

– die Generierung für den Mischbetrieb (ab Seite 54)

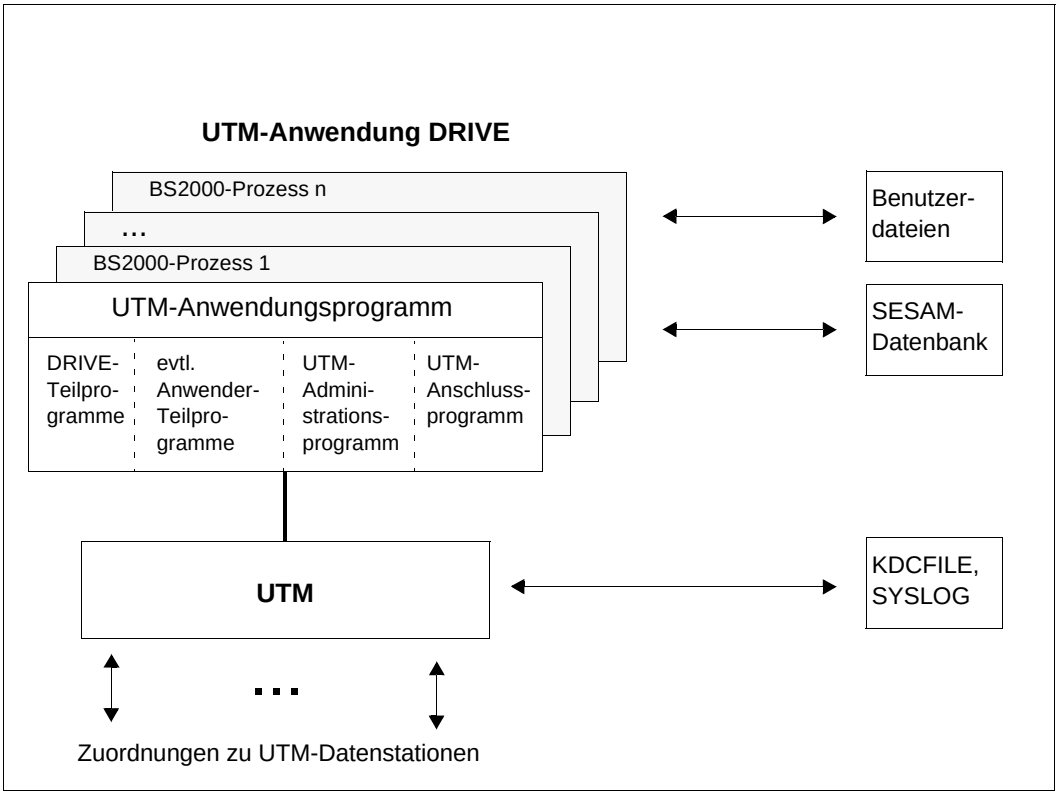
– die Generierung für den Old-Style-Betrieb (ab Seite 58)

– die Generierung für Verteilte Transaktionsverarbeitung (ab Seite 62)

Um DRIVE für den UTM-Betrieb zu generieren, müssen Sie eine UTM-Anwendung erstellen. Dafür werden die Komponenten einer UTM-Anwendung mit Ausnahme des UTM-Anschlussprogramms (KDCROOT) und der Anwendungskonfiguration (KDCFILE) ausgeliefert. KDCROOT und KDCFILE müssen generiert werden.

Daneben haben Sie auch die Möglichkeit, DRIVE in eine (bestehende) UTM-Anwendung mit anwenderspezifischen Teilprogrammen zu integrieren (siehe Abschnitt „DRIVE in eine bestehende UTM-Anwendung aufnehmen“ auf Seite 50).

Die folgende Abbildung zeigt schematisch eine UTM-Anwendung mit DRIVE:



Eine UTM-Anwendung besteht aus:

– der Beschreibung der Anwendungskonfiguration (KDCFILE)

– dem UTM-Anwendungsprogramm

Das UTM-Anwendungsprogramm besteht aus:

– dem UTM-Anschlussprogramm KDCROOT. Dies ist das Hauptprogramm, zu dem die Teilprogramme als Unterprogramme gebunden werden. Durch den Bindevorgang erhält man den Lademodul des UTM-Anwendungsprogramms.

– dem DRIVE-Teilprogramm DRIVROOT

– den Exits DRIVVORG, EXSTRT und EXSHUT

– dem UTM-Administrationsprogramm KDCADM. Für synchrone und asynchrone Administration.

Arbeitsschritte zur Erstellung einer UTM-Anwendung

Das Erstellen einer UTM-Anwendung mit DRIVE umfasst allgemein folgende Arbeitsschritte:

1. Einsatz von anwendereigenen Exits mit Hilfe des Makros ALLEX vorbereiten (siehe Abschnitt 3.4.1 auf Seite 42).
2. Erstellen der Anwendungskonfiguration (KDCFILE) und des UTM-Anschlussprogramms (KDCROOT) (siehe Abschnitt 3.4.2 auf Seite 44).
3. Übersetzen des UTM-Anschlussprogramms (siehe Abschnitt 3.4.3 auf Seite 51).

   Die Arbeitsschritte (2) und (3) können mittels einer von UTM ausgelieferten Generierungsprozedur erfolgen.
4. Generieren der UTM-Anwendung (siehe Abschnitt 3.4.4 auf Seite 52).

   Im UTM-Betrieb können SESAM-Datenbanken bearbeitet werden. Sie müssen die entsprechende DRIVE-Fassung in das UTM-Anwendungsprogramm einbinden.

   Für diesen Arbeitsschritt steht eine von DRIVE ausgelieferte Prozedur zur Verfügung.

3.4.1 Anwendereigene Exits integrieren

Für den Ablauf von DRIVE unter UTM bietet DRIVE zwei Exits:

– START-Exit

– SHUT-Exit.

Neben diesen zwei Standard-DRIVE-Exits können zusätzliche, anwendereigene START- oder SHUT-Exits existieren. Sie werden von den jeweiligen DRIVE-Exits aufgerufen, sobald die DRIVE-Exits abgearbeitet sind.
Ein Standardmodul für die DRIVE-Exits befindet sich in der Modulbibliothek SYSLNK.DRIVE.022.

Anwendereigene Exits können mit dem Makro ALLEX angeschlossen werden.

Die Source des Makros ALLEX befindet sich in der Bibliothek SIPLIB.DRIVE.022.

Aufbau des Makros ALLEX

```
[name]   ALLEX START=(DRIVSTRT,modul,...),
               SHUT= (DRIVSHUT,modul,...)
         END
```

Erläuterung:

name	beliebiger Modulname; kann auch weggelassen werden
modul	Name des jeweiligen Anwender-Exit-Moduls; maximale Anzahl: 9 Die DRIVE-Exits DRIVSTRT und DRIVSHUT sind zwingend erforderlich. Auch die Reihenfolge d. Module muss wie angegeben eingehalten werden.

Beispiel

Die folgende Prozedur dient zum Übersetzen des Makros ALLEX.

```
/BEGIN-PROC
/DELETE-SYSTEM-FILE OMF
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=ALTLIB,FILE-NAME=SIPLIB.DRIVE.022 ————————————————— (1)
/START-PROG $ASSEMBHC
*COMOPT ALTLIB
                    ———————————————————————————————————————————————————— (2)
*COMOPT SOURCE=ALLEX
*END HALT
/START-PROG $LMS
$LIB FILE=SYSLNK.DRIVE.022
                    ———————————————————————————————————————————————————— (3)
$ADDR *OMF
$END
/END-PROC
```

Erläuterung:

(1) Für die Assemblierung muss die PLAM-Bibliothek zugewiesen werden, die den Makro ALLEX enthält: SIPLIB.DRIVE.022.

(2) Assemblierung des Makros ALLEX.

Die Module MOD1 und MOD2 enthalten die Anwender-Exit-Routinen für START- und SHUT-Exit.
Makro-Aufrufe beginnen ab Spalte 10, deren Parameter ab Spalte 16.

(3) Eintragen des Bindemoduls in die Modulbibliothek SYSLNK.DRIVE.022.

i Das Makro ALLEX erzeugt ein ENTRY EXTAB, das in anderen Anwender-Teilprogrammen nicht vorkommen darf.

Die Reihenfolge der bei START angegebenen EXIT-Routinen muss mit der Reihenfolge der Startparameter in der Startprozedur der UTM-Anwendung übereinstimmen.

3.4.2 Anwendungskonfiguration und UTM-Anschlussprogramm erstellen

Die Anwendungskonfiguration (KDCFILE) enthält Informationen über:

– Anwendungseigenschaften
– Benutzerkennungen und Zugriffsschutz
– Eigenschaften von Datenstationen
– Eigenschaften von Transaktionscodes

Die KDCFILE wird in einer Datei mit dem Namen *basisname.KDCA* abgespeichert. Auf Anforderung wird sie doppelt geführt, d.h. zusätzlich in der Datei *basisname.KDCB*.
Bei Bedarf kann die KDCFILE auch gesplittet und auf mehrere Platten verteilt werden (siehe „*open*UTM - Anwendungen generieren und betreiben“ [13]).

Das UTM-Anschlussprogramm (KDCROOT) ist der Teil des UTM-Anwendungsprogramms, der die Verbindung zwischen UTM und den Teilprogrammen herstellt.

Das UTM-Dienstprogramm KDCDEF definiert die Anwendungskonfiguration (KDCFILE) und erzeugt das KDCROOT-Quellprogramm, das assembliert und gebunden werden muss.

Die von UTM ausgelieferte Generierungsprozedur SYSPRC.UTM.xxx(GEN) startet das Dienstprogramm KDCDEF. Dabei ist xxx die Versionsnummer von UTM, z.B. 040 oder 050. KDCDEF erzeugt die KDCFILE und das KDCROOT-Quellprogramm und übersetzt das KDCROOT-Quellprogramm (siehe „*open*UTM - Anwendungen generieren und betreiben“ [13]).

Zum Steuern von KDCDEF benötigt die Generierungsprozedur SYSPRC.UTM.xxx(GEN) eine Datei mit KDCDEF-Steueranweisungen (KDCDEF-Eingabedatei). Die zum Produktumfang von DRIVE gehörenden Elemente DRIKDCDEF.* enthalten DRIVE-spezifische Rahmen mit KDCDEF-Steueranweisungen und befinden sich in der Bibliothek SYSPRC.DRIVE.022. Eines dieser Elemente kann als KDCDEF-Eingabedatei verwendet werden, wenn es um anwenderspezifische KDCDEF-Steueranweisungen ergänzt wurde. Eine Musteranweisungsfolge finden Sie im Abschnitt „Beispiel“ auf Seite 48.

3.4.2.1 KDCDEF-Steueranweisungen

Die Steueranweisungen für das Dienstprogramm KDCDEF sind im Handbuch „*open*UTM - Anwendungen generieren und betreiben“ [13] beschrieben.

Für eine DRIVE-UTM-Anwendung sind folgende Angaben nötig:

MAX

`MAX TASKS ≥ 2`	Wenn eine Task belegt ist, dann kann mit einer anderen Task administriert werden
`MAX ASYNTASKS ≥ 1`	Asynchronverarbeitung ist erlaubt

`MAX KB ≥ 512`	Länge des Kommunikationsbereichs
`MAX SPAB = 32764`	Maximale Länge des SPAB
`MAX NB = 32700`	Maximale Länge des Nachrichtenbereichs
`MAX TRMSGLTH = 32700`	Maximale Nachrichtenlänge
`MAX LSSBS ≥ 200`	Maximale Anzahl LSSB's (abhängig von der Anwendung)
`MAX RECBUF ≥ (30,4096)`	Größe des Wiederanlaufbereichs in PAM-Seiten und Länge der Speicherbereiche in Byte pro Task
`MAX PGPOOL ≥ (1000,80,95)`	Pagepool-Größe in PAM-Seiten; sind davon 80% erreicht, erfolgt die erste, bei 95% die zweite Warnung
`MAX VGMSIZE ≥ 128`	Größe des Pufferbereichs für das Vorgangsgedächtnis von SESAM V2 in KByte.

DATABASE

Für das Datenbanksystem SESAM V2.x ist anzugeben:

```
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesamlib
```

PROGRAM

Folgende DRIVE-spezifische Teilprogramme sind anzugeben:

```
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT  ,COMP=ILCS
PROGRAM EXSHUT  ,COMP=ILCS
```

Ferner das UTM-spezifische Teilprogramm:

```
PROGRAM KDCADM,COMP=ILCS
```

MODULE

```
MODULE EXTAB   ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

EXIT

```
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
```

TAC

Folgende DRIVE-spezifische Transaktionscodes sind anzugeben:

```
TAC DRISQL  ,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC DRISQLF ,TYPE=D,STATUS=ON,CALL=NEXT ,PROGRAM=DRIVROOT,
TAC SQLNEXT ,TYPE=D,STATUS=ON,CALL=NEXT ,PROGRAM=DRIVROOT,
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC SQLRET,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIVROOT,TIME=300000
```

TAC-Name	Funktion
DRISQL	Beginn der Verarbeitung mit DRIVE. Anstelle von DRISQL dürfen auch anwenderspezifische TACs verwendet werden
DRISQLF	Fortsetzung der Verarbeitung mit DRIVE über PEND PA. Anstelle von DRISQLF dürfen auch anwenderspezifische TACs verwendet werden
SQLNEXT	Aufrufen des DRIVE-Dialog-Modus
SQLENTER	Asynchronbetrieb von DRIVE
SQLLIST	Asynchronvorgang zur Ausgabe benutzerspezifischer LIST-Sätze
SQLRET	Rückkehr zum DRIVE-Auftraggeber nach Aufrufen eines anwendereigenen UTM-Teilprogramms in C/COBOL in derselben Anwendung (CALL TAC) Dieser TAC muss nur angegeben werden, wenn im DRIVE-Programm eine Anweisung CALL TAC lokal verarbeitet wird, siehe auch KDCDEF-Rahmen DRIKDCDEF.CALL.LOCAL.TAC in der Bibliothek SYSPRC.DRIVE.022.

Folgende Transaktionscodes für die synchrone Administration von UTM sind anzugeben:

```
TAC KDCTAC  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCLTERM,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCPTERM,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSWTCH,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCUSER ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSEND ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCAPPL ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCDIAG ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCLOG  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCINF  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
```

```
TAC KDCHELP ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSHUT ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCTCL  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
```

Folgende Transaktionscodes für die asynchrone Administration von UTM sind anzugeben:

```
TAC KDCTACA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCLTRMA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCPTRMA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSWCHA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCUSERA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSENDA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCAPPLA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCDIAGA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCLOGA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCINFA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCHELPA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSHUTA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCTCLA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
```

SFUNC

```
SFUNC taste,RET=returncode
```

Die bei PARAMETER KFKEY angegebene Returncode-Taste muss mit der jeweiligen SFUNC-Angabe übereinstimmen,
z. B.: `SFUNC K1,RET=2OZ`

USER

```
USER username[,STATUS=ADMIN]
```

Mindestens ein Benutzer mit Administrationsberechtigung muss vorhanden sein.

PTERM

```
PTERM ptermname,PRONAM=prozessorname,PTYPE=partnertyp,LTERM=ltermname
```

Die Namen für die PTERM-Parameter müssen der PDN-Generierung entnommen werden. Es können alle von VTSU und FHS unterstützten Terminaltypen verwendet werden.

LTERM

`LTERM ltermname` Der LTERM-Name muss identisch mit dem LTERM-Namen der PTERM-Anweisung sein.

END

Der Name der Datei mit den KDCDEF-Anweisungen (KDCDEF-Eingabedatei) muss mit OPTION DATA=datei dem Programm KDCDEF zur Verarbeitung übergeben werden (siehe „*open*UTM - Anwendungen generieren und betreiben“ [13]).

Die KDCDEF-Steueranweisungen für die Aufnahme von DRIVE in eine **bestehende** UTM-Anwendung sind im Abschnitt „DRIVE in eine bestehende UTM-Anwendung aufnehmen“ auf Seite 50 beschrieben.

Die KDCDEF-Steueranweisungen, die Sie angeben müssen, wenn der Compiler DRIVE-COMP eingesetzt wird, sind im Handbuch „DRIVE-Compiler“ [5] beschrieben.

Zum Produktumfang von DRIVE gehören die Rahmen DRIKDCDEF.*, die KDCDEF-Steueranweisungen enthalten und die sich in der Bibliothek SYSPRC.DRIVE.022 befinden. Sie müssen vom Anwender an die konkreten Verhältnisse angepasst werden.

Beispiel

Dieser Abschnitt enthält ein Beispiel für KDCDEF-Steueranweisungen, die benötigt werden, wenn DRIVE im UTM-Betrieb auf eine Datenbank von SESAM V2.x (New Style) zugreifen soll.

```
REM
REM ******** MAXIMALWERTE EINSTELLEN ***********************************
REM
MAX TASKS=6,ASYNTASKS=1
MAX KB=512,SPAB=32767,NB=32700
MAX VGMSIZE=128
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10
MAX LOGACKWAIT=600,BRETRYNR=10
REM
REM ************ DATENBANK DEFINIEREN ***********************************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesamlib
REM
REM ************ TEILPROGRAMME DEFINIEREN *******************************
REM
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM KDCADM,COMP=ILCS
REM
```

```
REM ********* DRIVE-MODULE LADEN ***************************************
REM
MODULE EXTAB   ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM ************** USER-EXITS DEFINIEREN *********************************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *********** TRANSAKTIONSCODES FUER DRIVE ****************************
REM
DEFAULT TAC PROGRAM=DRIVROOT,STATUS=ON,
TAC DRISQL,TYPE=D  ,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF,TYPE=D ,CALL=NEXT,
TAC SQLNEXT,TYPE=D ,CALL=NEXT,
TAC SQLENTER,TYPE=A,CALL=FIRST,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,CALL=FIRST,EXIT=DRIVVORG
REM
REM *********** SYNCHRONE ADMINISTRATION ********************************
REM
DEFAULT TAC ADMIN=Y,PROGRAM=KDCADM,TYPE=D,STATUS=ON,CALL=BOTH,DBKEY=UTM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
TAC KDCTCL
REM
REM *********** ASYNCHRONE ADMINISTRATION *******************************
REM
DEFAULT TAC ADMIN=Y,PROGRAM=KDCADM,TYPE=A,STATUS=ON,CALL=FIRST,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
```

```
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
TAC KDCTCLA
REM
REM *********** BELEGUNG DER FUNKTIONSTASTEN ****************************
REM
SFUNC K1,RET=20Z
SFUNC F1,CMD=KDCOFF
SFUNC K2,RET=KDCOUT
REM
REM *********** ZUGELASSENE USER ****************************************
REM
USER user1
USER user2
USER user3
USER adm     ,STATUS=ON,PERMIT=ADMIN
USER adm1    ,STATUS=ON,PERMIT=ADMIN
USER adm2    ,STATUS=ON,PERMIT=ADMIN
USER admin   ,STATUS=ON,PERMIT=ADMIN
REM
REM *********** PTERMS UND LTERMS DEFINIEREN ****************************
REM
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm1
LTERM driterm1
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm2
LTERM driterm2
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm3
LTERM driterm3
END
```

3.4.2.2 DRIVE in eine bestehende UTM-Anwendung aufnehmen

Modifizieren und ergänzen Sie in der Eingabedatei zur Steuerung des UTM-Dienstprogramms KDCDEF die Anweisungen MAX, PROGRAM, MODULE, EXIT, TAC und SFUNC. Die notwendigen Angaben sind beschrieben im Abschnitt „KDCDEF-Steueranweisungen“ auf Seite 44.

3.4.3 UTM-Anschlussprogramm übersetzen

Sie haben zwei Möglichkeiten, das UTM-Anschlussprogramm zu übersetzen. Zum einen können Sie mittels SYSPRC.UTM.xxx(GEN), zum anderen unabhängig davon assemblieren (xxx=Version, z.B. 040).

– Assemblierung über SYSPRC.UTM.xxx(GEN) gesteuert

  Das folgende Beispiel zeigt einen Ausschnitt aus der Installationsprozedur SYSPRC.UTM.040(GEN) (siehe „*open*UTM - Anwendungen generieren und betreiben“ [13]). Es erfolgt die Abfrage: ASSEMBLING KDCROOT ? Bei Y wird die KDCROOT-Datei assembliert. Es muss dabei eine zweite ALTLIB-Zuweisung (ALTLIB2) erfolgen: auf die Bibliothek, die den KDCDB-Makro des entsprechenden Datenbanksystems enthält.

```
...
/REMARK TEXT=&ASSEMB= Y/N  ASSEMBLING KDCROOT?
/SKIP-COMMANDS TO-LABEL=RT&ASSEMB
/.RTY REMARK
/DELETE-FILE FILE-NAME=SYSLST.KDCROOT
/SET-JOB-STEP
/DELETE-SYSTEM-FILE FILE-NAME=OMF
/SET-JOB-STEP
/ASSIGN-SYSLST TO-FILE=SYSLST.KDCROOT
/SET-FILE-LINK LINK-NAME=ALTLIB,FILE-NAME=SYSLIB.UTM.040.ASS-
/                              ,ACCESS-METHOD=BY-CATALOG
/SET-FILE-LINK LINK-NAME=ALTLIB2,FILE-NAME=kdcdblib
/START-PROGRAM FROM-FILE=$ASSEMBHC
*COMOPT FLGLST,ATXREF,ALTLIB,ALTLIB2
*COMOPT SOURCE=ROOT.SRC.ASSEMB.&ROOTELEM
*END HALT
...
```

– Assemblierung unabhängig von SYSPRC.UTM.xxx(GEN)

```
/BEGIN-PROC C,YES,(&SRC),c'&' ------------------------------------------------  (1)
/DELETE-SYSTEM-FILE FILE-NAME=OMF
/SET-FILE-LINK LINK-NAME=ALTLIB,-              |
/    FILE-NAME=SYSLIB.UTM.040.ASS,-            |
/    ACCESS-METHOD=BY-CATALOG                  |-------------------------------  (2)
                                               |
/SET-FILE-LINK ALTLIB,SYSLIB.UTM.040.ASS       |
/SET-FILE-LINK LINK-NAME=ALTLIB2,FILE-NAME=kdcdblib ------------------------  (3)
/PARAMETER ALTLIB=YES
/ASSIGN-SYSDTA *SYSCMD
```

```
/START-PROGRAM $ASSEMBHC
*COMOPT SOURCE=&SRC
*COMOPT MODULE=rootlibname                                    (4)
*COMOPT ALTLIB2
*END HALT
/SET-JOB-STEP
/ASSIGN-SYSDTA *PRIMARY
/END-PROC
```

Erläuterung:

(1) Die Variable &SRC muss mit dem Namen der KDCROOT-Datei versorgt werden.

(2) Die UTM-Makros werden in der Bibliothek SYSLIB.UTM.040.ASS bereitgestellt.

(3) Angabe der Bibliothek, die den KDCDB-Makro enthält.

(4) Das Quellprogramm aus der Datei &SRC wird übersetzt.

Der durch die Übersetzung entstandene Bindemodul wird in die Modulbibliothek rootlibname abgelegt.

3.4.4 UTM-Anwendung generieren

Sie erstellen eine UTM-Anwendung, indem Sie zum UTM-Anschlussprogramm eine DRIVE-Fassung binden.

Zum Generieren der gewünschten DRIVE-Fassung steht Ihnen die Prozedur DRIPRC.INSTALL.DRIVE in der PLAM-Bibliothek SYSPRC.DRIVE.022 zur Verfügung. Abhängig von den einzugebenden Parameterwerten bindet diese BS2000-Prozedur die für die jeweilige DRIVE-Fassung erforderlichen DRIVE-Module mit dem UTM-Anschlussprogramm zu einem Bindelademodul (LLM) zusammen.

Rufen Sie die Prozedur mit folgendem BS2000-Kommando auf:

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)
```

Geben Sie im Prozedurablauf folgende Parameterwerte ein, um eine DRIVE-Fassung für den UTM-Betrieb zu generieren:

```
&STYLE = NEW
```

```
&BETRIEB = UTM
```

`&ROOTELEM = rootname` *rootname* bezeichnet das übersetzte KDCROOT-Tabellenmodul. Geben Sie den CSECT-Namen des KDCROOT-Tabellenmoduls an, der mit dem Parameter &ROOTELEM in der UTM-Generierungs-

prozedur SYSPRC.UTM.xxx(GEN) oder mit der KDCDEF-Steueranweisung ROOT rootname festgelegt wurde (siehe *open*UTM - Anwendungen generieren und betreiben [13]).

`&ROOTLIB = rootlibname`
Geben Sie die Bibliothek an, in der das ROOT-Element enthalten ist und in die der Binder das LLM *modulname* ablegen soll.

`&DRIVELIB = drivelib` Geben Sie für *drivelib* die Modulbibliothek mit den DRIVE-Modulen an.
Der Standardname für die Bibliothek ist: SYSLNK.DRIVE.022.

`&BINDER = binder` Geben Sie für *binder* den Dateinamen des Binders an.
Der Standardname für den Binder ist: $TSOS.BINDER.

`&EDTLIB = edtlib` Geben Sie für *edtlib* die Modulbibliothek an, in der sich das EDT-Anschlussmodul IEDTGLE befindet.
Der Standardname für die Bibliothek ist: $TSOS.EDTLIB.

`&STARTLLM = modulname` Geben Sie für *modulname* den Namen des LLMs an, zu dem die UTM-Anwendung gebunden wird.
Mit diesem Namen starten Sie die UTM-Anwendung. (Siehe die Beispiele im Abschnitt „Startprozedur“ auf Seite 73).

`&UTMLIB = utmlib` Geben Sie für *utmlib* die UTM-Modulbibliothek an.
Der Standardname für die Bibliothek ist SYSLNK.UTM.xxx.

`&UTMSPLLIB = splrtslib`
Geben Sie für *splrtslib* die SPLRTS-Bibliothek von UTM an.
Der Standardname für die Bibliothek ist SYSLNK.UTM.xxx.SPLRTS.

3.4.5 DRIVE zu einer bestehenden UTM-Anwendung binden

Die KDCDEF-Eingabedatei muss um die DRIVE-spezifischen KDCDEF-Steueranweisungen ergänzt werden, siehe Seite 55.

In der benutzereigenen Bindeprozedur müssen die DRIVE-spezifischen Bindeanweisungen eingebracht werden. Dazu enthält die Bibliothek SYSPRC.DRIVE.022 entsprechende Rahmendateien:

– DRIPRC.INSTALL.RAHMEN.SESAM für DRIVE-New-Style

– DRIPRC.INSTALL.RAHMEN.MIX für DRIVE-Mischbetrieb

3.4.6 Besonderheiten für den Mischbetrieb

Die Generierung einer UTM-Anwendung mit DRIVE für den Mischbetrieb entspricht im wesentlichen der beschriebenen Generierung für den New-Style-Betrieb. Deshalb sind hier nur die Unterschiede aufgezeigt. Die ausführliche Beschreibung finden Sie auf den Seiten 42 bis 52.

3.4.6.1 Anwendereigene Exits integrieren

Für den Ablauf von DRIVE im Mischbetrieb unter UTM bietet DRIVE zwei START-Exits, zwei SHUT-Exits und einen FORMAT-Exit an.

Neben diesen DRIVE-Exits können zusätzliche, anwendereigene START- oder SHUT-Exits existieren. Sie werden von den jeweiligen DRIVE-Exits aufgerufen, sobald die DRIVE-Exits abgearbeitet sind.

Ein Standardmodul für die DRIVE-Exits befindet sich in der Modulbibliothek SYSLNK.DRIVE.022.

Für den Mischbetrieb SESAM V2/LEASY befindet sich das Standardmodul EXTAB.MIXSQLLEA und für den Mischbetrieb SESAM V2/DMS das Standardmodul EXTAB.MIXSQLDMS in der Bibliothek SIPLIB.DRIVE.022. das entsprechende Modul muss unter dem Namen EXTAB in die Bibliothek SYSLNK.DRIVE.022 kopiert werden.

Anwendereigene Exits können mit dem Makro ALLEX angeschlossen werden.

Die Source des Makros ALLEX befindet sich in der Bibliothek SIPLIB.DRIVE.022.

Aufbau des Makros ALLEX

```
[name]  ALLEX START=(DRIVSTRT,exit,modul,...),
              SHUT =(DRIVSHUT,DRISHUT,modul,...),
              FORM =(DRIFORM,modul,...)
        END
```

Erläuterung:

name	beliebiger Modulname; kann auch weggelassen werden
exit	Für exit geben Sie an: DRISTARD für DMS-Fassung DRISTARL für LEASY-Fassung DRISTARS für SESAM V2-Fassung
modul	Name des jeweiligen Anwender-Exit-Moduls, maximale Anzahl: 8

Die DRIVE-Exits DRIVSTRT, DRIVSHUT, DRISHUT und DRIFORM sind zwingend erforderlich. Außerdem muss DRISTARD, DRISTARL oder DRISTARS eingegeben werden. Die Reihenfolge der Module muss wie angegeben eingehalten werden.

3.4.6.2 KDCDEF-Steueranweisungen

Die KDCDEF-Steueranweisungen für DRIVE im Mischbetrieb entsprechen im wesentlichen den KDCDEF-Steueranweisungen für den New-Style-Betrieb. Deshalb sind hier nur die Unterschiede aufgezeigt. Die Beschreibung der KDCDEF-Steueranweisungen für den New-Style-Betrieb finden Sie im Abschnitt „KDCDEF-Steueranweisungen“ auf Seite 44.

Modifizieren oder ergänzen Sie in der Eingabedatei zur Steuerung des UTM-Dienstprogramms KDCDEF folgende Anweisungen:

MAX

Für den Mischbetrieb sind folgende Werte erforderlich:

```
KB ≥ 512
SPAB = 32767
NB = 32700
TRMSGLTH = 32700
LSSBS ≥ 200
MAX VGMSIZE ≥ 128
```

DATABASE

Im Mischbetrieb ist zusätzlich zu SESAM V2 ein Zugriff auf LEASY und DMS möglich.

Für das Datenbanksystem SESAM V2 ist anzugeben:

```
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesamlib
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=sesamlib
```

Für LEASY ist anzugeben:

```
DATABASE TYPE=LEASY,ENTRY=LEASY
```

Für DMS entfällt die DATABASE-Anweisung.

PROGRAM

Folgende PROGRAM-Anweisungen sind zusätzlich anzugeben:

```
PROGRAM DRIKROOT,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM DRIVORG,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM EXFORM,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

MODULE

Folgende MODULE-Anweisungen sind zusätzlich anzugeben:

```
MODULE DRIFORM,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRISHUT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIEXT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2INT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2LINK,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIC51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2REF,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

DRIDUM51 entfällt.

Für das Datenbanksystem SESAM V2.x ist zusätzlich anzugeben:

```
MODULE DRISTARS,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SESORT,LIB=sesamlib,LOAD=STATIC
```

Für DMS ist zusätzlich anzugeben:

```
MODULE DRISTARD,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

Für LEASY ist zusätzlich anzugeben:

```
MODULE DRISTARL,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

Für LEASY entfällt: DRIC51.

EXIT

Folgende EXIT-Anweisung ist zusätzlich anzugeben:

```
EXIT PROGRAM=EXFORM,USAGE=FORMAT
```

TAC

Folgende TAC-Anweisungen sind zusätzlich anzugeben:

Für New-Style:

```
TAC DRISQLM,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIVROOT
```

Für Old-Style:

```
TAC DRIVE    ,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,EXIT=DRIVORG
TAC DRISES   ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRISEQ   ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRINEXT  ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT,TIME=300000
TAC DRIRTP   ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRISQL51 ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRIENTER ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,TIME=300000,EXIT=DRIVORG
TAC DRILIST  ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,TIME=300000,EXIT=DRIVORG
```

SFUNC

```
SFUNC K1,RET=2OZ
```

3.4.6.3 UTM-Anwendung generieren

Um eine UTM-Anwendung für den Mischbetrieb zu generieren, rufen Sie die BS2000-Prozedur DRIPRC.INSTALL.DRIVE auf (siehe Seite 52).

Der einzige Unterschied ist, dass Sie für den Parameter &FASSUNG folgenden Parameterwert angeben müssen:

```
&FASSUNG = MIX
```

Die Beschreibung der weiteren Parameter finden Sie auf Seite 52. Wie DRIVE zu einer bestehenden UTM-Anwendung gebunden werden soll, ist auf Seite 53 beschrieben.

3.4.7 Besonderheiten für den Old-Style-Betrieb

Die Generierung einer UTM-Anwendung mit DRIVE für den Old-Style-Betrieb entspricht im wesentlichen der beschriebenen Generierung für den New-Style-Betrieb. Deshalb sind hier nur die Unterschiede aufgezeigt. Die ausführliche Beschreibung finden Sie auf den Seiten 42 bis 52.

3.4.7.1 Anwendereigene Exits integrieren

Für den Ablauf von DRIVE im Old-Style-Betrieb unter UTM bietet DRIVE drei Exits: START-Exit, SHUT-Exit und FORMAT-Exit.

Neben diesen drei Standard-DRIVE-Exits können zusätzliche, anwendereigene START-, FORMAT- oder SHUT-Exits existieren. Sie werden von den jeweiligen DRIVE-Exits aufgerufen, sobald die DRIVE-Exits abgearbeitet sind. Ein Standardmodul für die DRIVE-Exits befindet sich in der Modulbibliothek SYSLNK.DRIVE.022.

Anwendereigene Exits können mit dem Makro ALLEX angeschlossen werden.

Die Source des Makros ALLEX befindet sich in der DRIVE-Systembibliothek SIPLIB.DRIVE.022.

Aufbau des Makros ALLEX

```
[name]   ALLEX START=(exit,modul,...),
               SHUT =(DRISHUT,modul,...),
               FORM =(DRIFORM,modul,...)
         END
```

Erläuterung:

name — beliebiger Modulname; kann auch weggelassen werden

exit — Für exit geben Sie an:

DRISTARD für DMS-Fassung
DRISTARL für LEASY-Fassung
DRISTARS für SESAM V2-Fassung

modul — Name des jeweiligen Anwender-Exit-Moduls; maximale Anzahl: 9

Die DRIVE-Exits DRISHUT und DRIFORM sind zwingend erforderlich. Außerdem muss DRISTARD, DRISTARL oder DRISTARS eingegeben werden. Die Reihenfolge der Module muss wie angegeben eingehalten werden.

3.4.7.2 KDCDEF-Steueranweisungen

Die KDCDEF-Steueranweisungen für das Dienstprogramm KDCDEF sind in „*open*UTM - Anwendungen generieren und betreiben“ [13] beschrieben.

Für eine UTM-Anwendung im Old-Style-Betrieb sind folgende KDCDEF-Steueranweisungen notwendig:

MAX

```
MAX KB ≥ 512
MAX SPAB ≥ 10000
MAX NB ≥ 4632
MAX TRMSGLTH ≥ 4632
MAX LSSBS ≥ 50
```

DATABASE

Für das Datenbanksystem SESAM V2.x ist anzugeben:

```
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=sesamlib
```

Für LEASY ist anzugeben:

```
DATABASE TYPE=LEASY,ENTRY=LEASY
```

Für DMS entfällt die DATABASE-Anweisung.

PROGRAM

```
PROGRAM DRIKROOT,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM DRIVORG,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM EXFORM,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM KDCADM,COMP=ILCS
```

MODULE

```
MODULE DRIFORM,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRISHUT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIEXT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2INT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2LINK,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIC51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

```
MODULE EXTAB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIVMC,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2REFX,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

Die KDCDEF-Steueranweisung MODULE EXTAB muss vor der Anweisung MODULE DRIVMC stehen.

Für das Datenbanksystem SESAM V2.x ist zusätzlich anzugeben:

```
MODULE DRISTARS,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SESUTMC,LIB=sesamlib,LOAD=STATIC
MODULE SESORT,LIB=sesamlib,LOAD=STATIC
```

Für DMS ist zusätzlich anzugeben:

```
MODULE DRISTARD,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

Für LEASY ist zusätzlich anzugeben:

```
MODULE DRISTARL,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

Für LEASY entfällt: DRIC51.

EXIT

```
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
EXIT PROGRAM=EXFORM,USAGE=FORMAT
```

TAC

```
DEFAULT TAC PROGRAM=DRIKROOT
TAC DRIVE   ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVORG
TAC DRISES  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRISEQ  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRINEXT ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRIRTP  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRIENTER,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVORG
TAC DRILIST ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVORG
```

SFUNC

```
SFUNC K1,RET=2OZ
```

DRIVE in eine bestehende UTM-Anwendung aufnehmen

Modifizieren oder ergänzen Sie in der Eingabedatei zur Steuerung des UTM-Dienstprogramms KDCDEF die Anweisungen MAX, PROGRAM, MODULE, EXIT, TAC und SFUNC. Die notwendigen Angaben sind beschrieben im Abschnitt „Anwendungskonfiguration und UTM-Anschlussprogramm erstellen“ auf Seite 44.

3.4.7.3 UTM-Anschlussprogramm übersetzen

Das Übersetzen des UTM-Anschlussprogramms für DRIVE im Old-Style-Betrieb entspricht im wesentlichen dem Übersetzungsvorgang im New-Style-Betrieb. Deshalb sind hier nur die Unterschiede aufgezeigt. Die ausführliche Beschreibung des Übersetzungsvorgangs finden Sie im Abschnitt „UTM-Anschlussprogramm übersetzen“ auf Seite 51.

– Assemblierung über SYSPRC.UTM.xxx(GEN) gesteuert

  Bei der DMS-Fassung entfällt die ALTLIB2-Zuweisung.

– Assemblierung unabhängig von SYSPRC.UTM.xxx(GEN)

  Bei der DMS-Fassung entfällt ALTLIB2.

3.4.7.4 UTM-Anwendung generieren

Um eine UTM-Anwendung für den Old-Style-Betrieb zu generieren, rufen Sie die BS2000-Prozedur DRIPRC.INSTALL.DRIVE auf (siehe Seite 52).

Die einzigen Unterschiede sind, dass Sie für die Parameter &STYLE und &FASSUNG die folgenden Parameterwerte angeben müssen:

&STYLE = OLD

`&FASSUNG = [ SESAM | LEASY | DMS ]`
: Geben Sie das Datenhaltungssystem, auf das DRIVE zugreifen soll.

Außerdem müssen Sie die Parameter &PHASE, &EDTLIB, &TSOSLNK und ggf. &LEASYLIB angeben:

`&PHASE = modulname`
: Geben Sie für *modulname* den Namen der Phase, zu dem die generierte DRIVE-Fassung gebunden wird.
Dieser Name ist beim Starten im /START-PROG-Kommando anzugeben (siehe Abschnitt „Startprozedur“ auf Seite 73).

`&TSOSLNK = tsoslnkname`
: Geben Sie für *tsoslnkname* den Dateinamen des statischen Binders TSOSLNK an.
Der Standardname ist: $TSOS.TSOSLNK.

`&EDTLIB = edtlib`
: Geben Sie für *edtlib* die Modulbibliothek an, in der sich das EDT-Anschlussmodul IEDTGLE befindet.
Der Standardname für die Bibliothek ist: $TSOS.EDTLIB.

`&LEASYLIB = leasylib`
: Geben Sie für *leasylib* die Modulbibliothek mit den LEASY-Modulen an.
Nur erforderlich, wenn Sie &FASSUNG=LEASY angegeben haben.

Die Beschreibung der weiteren Parameter finden Sie auf Seite 52.

3.4.8 VTV-Anwendung generieren

Eine UTM-Anwendung für VTV wird genauso generiert wie eine UTM-Anwendung ohne VTV (siehe Seiten 42 bis 53). Das UTM-Anschlussprogramm KDCROOT muss allerdings (wegen UTM-D) mit der Prozedur SYSPRG.UTM-D.xxx.KDCDEF übersetzt werden. Außerdem müssen die KDCDEF-Eingabedateien erweitert werden um Steueranweisungen zum Adressieren der jeweiligen Auftraggeber und Auftragnehmer.

3.4.8.1 Auftragnehmer adressieren

Mit dem KDCS-Aufruf APRO adressiert der Auftraggeber den Auftragnehmer

```
APRO KCRN=<tac-name-in-der-partneranwendung>,KCPI=<vorgangs-id>

     KCPA=<partner-anwendungsname>
```

Die Angabe <tac-name-in-der-partneranwendung> muss mit der LTAC-Angabe in der KDCDEF des Auftraggebers übereinstimmen.

Für die einstufige Adressierung muss der <partner-anwendungsname> in der KDCDEF des Auftraggebers angegeben werden (LPAP-Angabe in der LTAC-Anweisung). Für die zweistufige Adressierung brauchen Sie den Namen nicht in der KDCDEF des Auftraggebers anzugeben. In diesem Fall müssen Sie mit der Anweisung PARAMETER DISTRIBUTION den Namen der Partneranwendung als Auftragnehmer angeben (siehe „DRIVE-Lexikon“ [3]).

3.4.8.2 KDCDEF-Steueranweisungen

Die allgemeinen KDCDEF-Steueranweisungen für eine DRIVE-UTM-Generierung sind im Abschnitt „KDCDEF-Steueranweisungen“ auf Seite 44 beschrieben.

Zusätzlich zu den dort beschriebenen DRIVE-spezifischen Transaktionscodes müssen Sie für VTV für den Auftragnehmer die folgenden TACs angeben:

```
TAC SQLRMT,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=DRIVROOT,TIME=30000
TAC SQLRMTA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG,TIME=300000
```

TAC-Name	**Funktion**
SQLRMT	Synchroner Aufruf einer DRIVE-Auftragnehmer-Anwendung (CALL)
SQLRMTA	Asynchroner Aufruf einer DRIVE-Auftragnehmer-Anwendung (ENTER)

Mit den folgenden KDCDEF-Anweisungen adressieren Sie die Auftraggeber-/ Auftragnehmer-Anwendungen für die VTV:

```
UTMD RSET=LOCAL
```

für Auftraggeber und -nehmer, damit bei ROLLBACK WORK und fehlerhaftem Programmabbruch der RSET-Aufruf von DRIVE nur lokal auf den aktuellen Vorgang wirkt (d.h. die lokale Transaktion wird zurückgesetzt und nicht alle beteiligten Transaktionen wie bei ROLLBACK WORK WITH RESET über PEND RS)

Andernfalls bricht UTM den aktuellen Vorgang mit dem Fehlercode 87Z (K320) ab.

```
LTAC partner-tac-name [,LPAP=partner-anwendungsname]
```

definiert lokale Namen für Transaktionscodes in der entfernten Partneranwendung (LPAP-Angabe nur bei einstufiger Adressierung, bei zweistufiger Adressierung wird die Partneranwendung mit der Anweisung PARAMETER DISTRIBUTION bekanntgegeben).

```
SESCHA sescha-name,CONNECT=Y,PLU=Y/N
```

definiert Session-Eigenschaften.

PLU=Y/N :
PLU=N bedeutet, dass diese Anwendung für den Aufbau der Session zuständig ist, PLU=Y bedeutet, dass die Partneranwendung für den Aufbau der Session zuständig ist. In einem Rechner muss PLU=Y angegeben werden, im anderen PLU=N. Diese Angaben sind unabhängig davon, welche Anwendung Auftraggeber und welche Auftragnehmer ist.

Empfehlung: Geben Sie in einem Rechner alle Verbindungen mit PLU=Y an und im Partnerrechner entsprechend PLU=N.

```
LSES lokaler-session-name,LPAP=partner-anwendungsname,
     RSES=session-name-in-der-partneranwendung
```

definiert Sessionnamen für die Verbindung von zwei Anwendungen.

```
LPAP partner-anwendungsname,SESCHA=sescha-name
```

legt lokale Namen für die entfernte Partneranwendung fest.

```
BCAMAPPL bcam-anwendungsname,T-PROT=ISO
```

legt einen zusätzlichen lokalen Anwendungsnamen für die Transport-Verbindung fest (neben dem Namen, der mit MAX APPLINAME vergeben wurde). Dadurch gibt es zwischen zwei Anwendungen nicht nur eine, sondern zwei oder mehrere Transportverbindungen, so dass Aufträge parallel abgewickelt werden können. Pro paralleler Verbindung müssen Sie eine BCAMAPPL-Anweisung, eine CON-Anweisung und eine LSES-Anweisung angeben. Die LPAP- und SESCHA-Anweisungen sind nur einmal notwendig (Ausnahme siehe Hinweis unten).

```
CON bcam-partner-anwendungsname,LPAP=partner-anwendungsname,
    BCAMAPPL=bcam-anwendungsname,PRONAM=prozessorname
```

definiert die logische Transport-Verbindung zwischen lokaler und entfernter Anwendung.

Sollen in einem Anwendungssysstem zwei Partneranwendungen sowohl Auftraggeber als auch Auftragnehmer zueinander sein, sollten Sie parallele Sessions generieren, d.h. zwei SESCHA- und LPAP-Anweisungen angeben und entsprechende BCAMAPPL-, CON- und LSES-Anweisungen. Andernfalls kann es leicht zu Engpässen bei den Verbindungen kommen.

Beispiel für KDCDEF-Rahmen bei VTV

Dieser Abschnitt enthält Beispiele für KDCDEF-Steueranweisungen, die benötigt werden, wenn DRIVE bei VTV auf eine Datenbank von SESAM V2.x zugreifen soll. Abhängig davon, ob die DRIVE-Anwendung Auftraggeber oder Auftragnehmer ist, sind die KDCDEF-Steueranweisungen unterschiedlich.

Auftraggeber

```
REM
REM *** Maximalwerte einstellen *********************
REM
MAX KB=512,SPAB=32767,NB=32700
MAX TASKS=6,ASYNTASKS=1
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700,
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
```

```
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60, CONRTIME=10
MAX VGMSIZE=128
REM
REM ***RSET darf nur lokal wirken ***
REM
UTMD RSET=LOCAL
REM
REM *** Datenbank SESAM V2.2 zuweisen *****************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=$TSOS.SYSLNK.SESAMSQL.022
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
REM
REM *** Teilprogramme definieren ********************
REM
PROGRAM KDCADM  ,COMP=ILCS
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT  ,COMP=ILCS
PROGRAM EXSHUT  ,COMP=ILCS
REM
REM *** USER-EXITS definieren ***********************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *** DRIVE-Module laden **************************
REM
MODULE EXTAB  ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM *** Transaktionscodes fuer DRIVE ****************
REM
DEFAULT TAC PROGRAM=DRIVROOT
TAC SQLRET,TYPE=D,STATUS=ON,CALL=NEXT
REM
TAC DRISQL  ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF ,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT ,TYPE=D,CALL=NEXT,TIME=300000
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVVORG
REM
REM *** Synchrone Administration ********************
REM
DEFAULT TAC TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,PROGRAM=KDCADM, DBKEY=UTM
TAC KDCTAC
```

```
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
TAC KDCTCL
REM
REM *** Asynchrone Administration ******************
REM
DEFAULT TAC TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
TAC KDCTCLA
REM
REM *** PROGRAM/MODULE/TAC/LTAC-Anweisungen für *****
REM *** C - DRIVE ***********************************
REM
PROGRAM CMAINPR, COMP=ILCS
TAC SNDTACCC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CMAINPR
TAC RECTACCC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CMAINPR
REM
REM *** PROGRAM/MODULE/TAC/LTAC-Anweisungen für *****
REM *** DRIVE -C ************************************
REM
LTAC CANTAC
REM
REM *** PROGRAM/MODULE/TAC/LTAC-Anweisungen für *****
REM *** DRIVE - C via PEND PA lokal *****************
REM
PROGRAM CTACMAIN, COMP=C
```

```
TAC CLOCTAC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CTACMAIN
REM
REM *** Belegung der Funktionstasten ***************
REM
SFUNC K1,RET=20Z
SFUNC F1,CMD=KDCOFF
SFUNC K2,CMD=KDCOUT
REM
REM *** Zugelassene USER ****************************
REM
USER user1
USER user2
USER user3
USER admin1,STATUS=ON,PERMIT=ADMIN
USER admin2,STATUS=ON,PERMIT=ADMIN
USER admin3,STATUS=ON,PERMIT=ADMIN
REM
REM *** DRIVE-Remote-TAC ****************************
REM
LTAC SQLRMT
LTAC SQLRMTA,TYPE=A
REM
REM *** Adressierung der BS2000-Partner-Anwendung ***
REM *** auf dem gleichen Rechner ********************
REM
SESCHA SESBSAP1, CONNECT=YES, PLU=Y
LPAP LPABSAP2, SESCHA=SESBSAP1
REM
LSES LSBS1AP1, LPAP=LPABSAP2, RSES=LSBS1AP2
LSES LSBS2AP1, LPAP=LPABSAP2, RSES=LSBS2AP2
LSES LSBS3AP1, LPAP=LPABSAP2, RSES=LSBS3AP2
LSES LSBS4AP1, LPAP=LPABSAP2, RSES=LSBS4AP2
BCAMAPPL BCBS1AP1
BCAMAPPL BCBS2AP1
BCAMAPPL BCBS3AP1
BCAMAPPL BCBS4AP1
CON BCBS1AP2, BCAMAPPL=BCBS1AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS2AP2, BCAMAPPL=BCBS2AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS3AP2, BCAMAPPL=BCBS3AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS4AP2, BCAMAPPL=BCBS4AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
REM
REM *** PTERM's und LTERM's definieren **************
REM
TPOOL LTERM=DRIUSER, NUMBER=9, PRONAM=D255S156, PTYPE=T9750
TPOOL LTERM=DRIUSE , NUMBER=9, PRONAM=D255S247, PTYPE=T9750
REM
```

```
TPOOL LTERM=DRI, NUMBER=10, PRONAM=D016KR20, PTYPE=T9750
REM
END
```

Auftragnehmer

```
REM *** KDCFILE und ROOT erzeugen ******************
REM
OPTION GEN=ALL
ROOT rootname
REM
MAX APPLINAME=bs2anw, KDCFILE=utm.bs2anw
REM
REM *** Maximalwerte einstellen ********************
REM
MAX KB=512, SPAB=32767, NB=32700
MAX TASKS=6,ASYNTASKS=1
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10,
MAX LOGACKWAIT=600,BRETRYNR=10
MAX VGMSIZE=128
REM
REM *** RSET darf nur lokal wirken ***
UTMD RSET=LOCAL
REM
REM *** Verwendete Datenbanken beschreiben: SESAM V2.2 *
REM
DATABASE ENTRY=SESSQL,TYPE=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
DATABASE ENTRY=SESAM,TYPE=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
REM
REM *** Teilprogramme definieren ********************
REM
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM KDCADM,COMP=ILCS
REM
REM *** Anweisungen fuer Verbindung von DRIVE-Auftraggeber *
REM *** mit C-Teilprogrammen als Auftragnehmer ******
REM
PROGRAM CANMAIN,COMP=ILCS
TAC CANTAC,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=CANMAIN
REM
REM *** USER-EXITS definieren ***********************
```

```
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *** DRIVE-Module laden *************************
REM DRIVE-Bibliothek
MODULE EXTAB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART,...
MODULE EXSHUTE,...
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM *** Transaktionscodes und Tacclass fuer DRIVE ***
REM
DEFAULT TAC PROGRAM=DRIVROOT
TAC SQLRMT,TYPE=D,STATUS=ON,CALL=BOTH,TIME=30000
TAC SQLRMTA,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
REM
TAC DRISQL,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT,TYPE=D,CALL=NEXT,TIME=300000
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
TAC SQLLIST,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
REM
REM *** SYNCHRONE ADMINISTRATION ********************
REM
DEFAULT TAC TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,PROGRAM=KDCADM,DBKEY=UTM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
TAC KDCTCL
REM
REM *** ASYNCHRONE ADMINISTRATION ********************
REM
DEFAULT TAC TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
```

```
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
TAC KDCTCLA
REM
REM *** ZUGELASSENE USER ***************************
REM
USER user1
USER user2
USER user3
USER admin1,STATUS=ON,PERMIT=ADMIN
USER admin2,STATUS=ON,PERMIT=ADMIN
USER admin3,STATUS=ON,PERMIT=ADMIN
REM
REM *** Funktionstasten belegen *********************
REM
SFUNC K1,RET=2OZ              "BREAK-Taste"
SFUNC F1,CMD = KDCOFF
SFUNC K2,CMD = KDCOUT
REM
TPOOL LTERM=DRIUSER,NUMBER=9,PRONAM=D255S156,PTYPE=T9750
TPOOL LTERM=DRIUSE,NUMBER=9,PRONAM=D255S247,PTYPE=T9750
TPOOL LTERM=DRI,NUMBER=10,PRONAM=D016KR19,PTYPE=T9750
REM
REM *** Partneranwendung adressieren ****************
REM *** DRIVE oder C als Auftraggeber im BS2000 *****
REM
SESCHA SESBSAP2,CONNECT=Y,PLU=N
REM
LPAP LPABSAP1, SESCHA=SESBSAP2
LSES LSBS1AP2, LPAP=LPABSAP1, RSES=LSBS1AP1
LSES LSBS2AP2, LPAP=LPABSAP1, RSES=LSBS2AP1
LSES LSBS3AP2, LPAP=LPABSAP1, RSES=LSBS3AP1
LSES LSBS4AP2, LPAP=LPABSAP1, RSES=LSBS4AP1
REM
REM
BCAMAPPL BCBS1AP2
BCAMAPPL BCBS2AP2
BCAMAPPL BCBS3AP2
BCAMAPPL BCBS4AP2
```

```
REM
CON BCBS1AP1,BCAMAPPL=BCBS1AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS2AP1,BCAMAPPL=BCBS2AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS3AP1,BCAMAPPL=BCBS3AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS4AP1,BCAMAPPL=BCBS4AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
END
```

3.4.8.3 UTM-Anschlussprogramm KDCROOT übersetzen

Das UTM-Anschlussprogramm müssen Sie wegen UTM-D mit der Prozedur SYSPRG.UTM-D.xxx.KDCDEF übersetzen.

3.4.8.4 Fehlerbehandlung

Bei KDCS-Aufrufen wertet DRIVE die Felder KCRCCC (KDCS-Fehlercode) und KCRCDC (interner Fehlercode) aus, die von UTM im KB-Rückgabebereich versorgt werden.

KDCS-Fehlercodes bei APRO-Aufrufen

Die Fehlercodes 40Z,44Z und 46Z führen zu Programmabbruch, alle anderen Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei MPUT-Aufrufen

Alle Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei FPUT-Aufrufen

Die Fehlercodes 40Z und 44Z führen zu Programmabbruch, alle anderen Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei MGET- und FGET-Aufrufen

Alle Fehlercodes führen zum Vorgangsende.

3.5 Einsatz einer DRIVE-UTM-Anwendung

(Änderungen zu Kapitel 16 von „DRIVE-Programmiersystem“ [1])

3.5.1 Voraussetzungen für den Start

Folgende Dateien müssen der UTM-Anwendung zur Verfügung stehen:

– Datei mit dem Lademodul des UTM-Anwendungsprogramms
Der Dateiname wurde mit dem Parameter &STARTLLM = modulname oder &PHASE = modulname in der Prozedur DRIPRC.INSTALL.DRIVE festgelegt. Siehe Abschnitte „UTM-Anwendung generieren“ auf Seite 52, Seite 57 (Mischbetrieb), Seite 61 (Old-Style) und Abschnitt „VTV-Anwendung generieren“ auf Seite 62.

– KDCFILE (Suffix des Dateinamens: .KDCA)

– ggf. Benutzerprotokolldatei (Suffix des Dateinamens: .USLA)

– Systemprotokolldatei SYSLOG

– Modulbibliothek $TSOS.SYSLNK.UTM.xxx (xxx=Version)

– ggf. Benutzerdateien der UTM-Anwendung (nur im Mischbetrieb)

Das Laden von Modulen als Shared Code in den Klasse-3/4-Speicher mit Hilfe von DSSM ist optional (siehe Abschnitt „DRIVE-Module als Shared-Code laden“ auf Seite 30).

3.5.2 Anwendung starten

Damit die Datensichtstation, von der aus die UTM-Produktivanwendung gestartet wurde, nicht für die Dauer der UTM-Anwendung blockiert wird, sollte die UTM-Anwendung aus einer BS2000-Batch-Prozedur gestartet werden (siehe „Startprozedur“ auf Seite 73).

Wie die Startprozedur einer bestehenden UTM-Anwendung für DRIVE modifiziert werden muss, ist beschrieben im Abschnitt „Startprozedur einer bestehenden UTM-Anwendung für DRIVE modifizieren“ auf Seite 76.

Eine Rahmen-Startprozedur mit dem Namen DRIPRC.START befindet sich in der Bibliothek SYSPRC.DRIVE.022.

Der Name der Datei, die die Startprozedur enthält, muss als UTM-Startparameter angegeben werden.

3.5.2.1 Startprozedur

Eine BS2000-Prozedur zum Starten einer UTM-Anwendung hat folgende Komponenten:

- Zuweisen von Modulbibliotheken (siehe Abschnitt „Modulbibliotheken zuweisen“ auf Seite 32)
- Einrichten der System-Protokolldatei SYSLOG
- Bekanntgeben des Datenbank-Konfigurationsnamens (für SESAM)
- evtl. Einrichten einer zentralen Druckdatei DRILIST (siehe Abschnitt „Zentrale Druckdatei (LIST-Datei) für den UTM-Betrieb bereitstellen“ auf Seite 35)
- Aufrufen des UTM-Anwendungsprogramms
- Startparameter für die UTM-Anwendung
- Startparameter für das Formatierungssystem
- DRIVE-Startparameter

  Mit DRIVE-Startparametern stellen Sie anwendungsweite, d.h. für alle Anwender der UTM-Anwendung gültige DRIVE-Parameter ein, z.B. Zuweisen der DRIVE-Bibliothek, Anfordern DRIVE-spezifischer Speicherbereiche (DRIVE-Cache) oder Belegen von K/F-Tasten mit DRIVE-Funktionen.

  Außerdem müssen Sie die für den Ablauf eines DRIVE-Programms notwendigen DRIVE-Anweisungen (z.B. dynamische Parameter) angeben, wenn es entweder dem Anwender nicht erlaubt ist, diese Anweisungen selbst im Dialog-Modus einzugeben (siehe PARAMETER LOCK) oder wenn diese Anweisungen nicht in Vorlauf-Programm abgearbeitet werden (siehe „DRIVE-Programmiersystem“ [1], Abschnitt „Datenschutz im UTM-Betrieb“).
- Sprunganweisung zum Prozedurstart nach Beendigung durch Fehler.

Das UTM-Anwendungsprogramm liest die Startparameter über SYSDTA.

Beispiel für eine Startprozedur einer UTM-Produktivanwendung mit DRIVE New-Style

```
/LOGON ————————————————————————————————————————————————————————————————————————————— (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=basisname.SYSLOG,SHARED-UPDATE=YES ┐
                                                                             ├———— (2)
/SET-FILE-LINK FILE-NAME=basisname.KDCA,SHARED-UPDATE=YES                    ┘
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$kennung.SYSMSA.ESQL-COBOL.022          ┐
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$kennung.SYSMSA.SESAMSQL.022            ├———— (3)
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$kennung.SYSMSA.SES-SQL-GEM.022         ┘
```

```
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lmslib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhsmacrolib
                                                                    (4)
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=systemmacrolib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usrlib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=formatlib
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=listdatei
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesamlib
                                                                    (5)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesamkonf
/.NEU REMARK
/START-PROGRAM FROM-FILE=*MOD(LIB=rootlib,ELEM=modulname,PROG-MO=ANY,-
/              RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY,-   (6)
/              LO-IN=REF))
.UTM START FILEBASE=basisname
.UTM START TASKS=3
                                                                    (7)
.UTM START ASYNTASKS=1
.UTM START STARTNAME=enterdatei
.FHS DE=NO
                                                                    (8)
.FHS MAPLIB=formatlib
.UTM END                                                            (9)
.DRIVE PAR DYNAMIC LIBRARY=libname
.DRIVE PAR DYNAMIC CATALOG=projekt SCHEMA=mitarb
.DRIVE PAR DYNAMIC AUTHORIZATION=geheim
.DRIVE PAR KFKEY='K1' ACTION=BREAK UTMRC='20Z'
                                                                    (10)
.DRIVE PAR KFKEY='K3' ACTION=EXIT UTMRC='33Z'
.DRIVE PAR KFKEY='K4' UTMRC='25Z'
.DRIVE ACQUIRE MEMORY 120 USER 5
.DRIVE END
/SKIP-COMMANDS .NEU                                                 (11)
/LOGOFF
```

Erläuterung:

(1) Die Startprozedur sollte eine Batch-Prozedur sein, damit die Datenstation, von der aus sie gestartet wird, nicht durch einen Dialog-Prozess blockiert ist.

(2) Zuweisen von Systemdateien

(3) Zuweisen von SESAM-Meldungsdateien (notwendig bei SESAM V2.x)

(4) Zuweisen von Modulbibliotheken und Einrichten der zentralen Druckdatei DRILIST

(5) Zuweisen der SESAM-Modulbibliothek

(6) Mit dem Kommando /START-PROGRAM rufen Sie die generierte UTM-Anwendung auf.

(7) UTM-Startparameter:

- Der Basisname für die KDCFILE und die Benutzerprotokolldatei wurde mit dem Parameter &FILEBASE bei SYSPRC.UTM.xxx(GEN) oder mit der KDCDEF-Steueranweisung MAX KDCFILE=basisname festgelegt.
- Es werden drei Tasks, davon eine Task für asynchrone Vorgänge generiert.
- Für *enterdatei* geben Sie an: Name der Datei, die die Batch-Prozedur für den Start der UTM-Anwendung enthält.

(8) FHS-Startparameter

(9) Ende der Eingabe von UTM-Startparametern und Startparametern für das Formatierungssystem. Die UTM-Anwendung wird gestartet, d.h. als nächstes wird der START-Exit aktiviert.

(10) DRIVE-Startparameter (siehe Anweisungen PARAMETER DYNAMIC, PARAMETER KFKEY, PARAMETER STATIC und ACQUIRE im „DRIVE-Lexikon“ [3].)

Die UTMRC-Angaben bei PAR KFKEY müssen mit den SFUNC-Angaben KDCDEF übereinstimmen (siehe Abschnitt „SFUNC“ auf Seite 47).

(11) Sprung nach /.NEU

Falls die UTM-Anwendung auf Grund eines Fehlers abgebrochen wird, wird sofort ein neuer Start eingeleitet.

Startprozedur einer bestehenden UTM-Anwendung für DRIVE modifizieren

Modifizieren und ergänzen Sie ggf. die UTM-Startprozedur je nach DRIVE-Fassung um folgende Kommandos und Anweisungen:

```
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lmslib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhsmacrolib
                                                                    (1)
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=systemmacrolib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usrlib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=formatlib
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=listdatei
```

Für das Datenbanksystem SESAM ist anzugeben:

```
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesamlib
                                                                    (2)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesamkonf
.FHS DE=NO
                                                                    (3)
.FHS MAPLIB=formatlib
.DRIVE startparameter                                               (4)
.DRIVE END
```

Erläuterung:

(1) Zuweisen von Modulbibliotheken) und Einrichten der zentralen Druckdatei DRILIST

(2) Zuweisen der Modulbibliothek für das Datenbanksystem

(3) Startparameter für das Formatierungssystem

(4) DRIVE-Startparameter (siehe Seite 73)

3.5.2.2 Startprozedur für den Mischbetrieb

In einer UTM-Startprozedur für den DRIVE-Mischbetrieb müssen Sie neben den New-Style- auch die Old-Style-Startparameter angeben. Fehlerhafte Startparameter führen zu Fehlermeldungen nach SYSLST und zum Abbruch des Starts der UTM-Anwendung.

Beispiel für eine Startprozedur einer UTM-Testanwendung (SESAM V2.x-Fassung):

```
/LOGON ------------------------------------------------------------------------ (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=basisname.SYSLOG,SHARED-UPDATE=YES
                                                                          ------ (2)
/SET-FILE-LINK FILE-NAME=basisname,KDCA,SHARED-UPDATE=YES
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crtelib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lmslib
                                                                  ------------- (3)
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhsmacrolib
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=systemmacrolib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usrlib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=formatlib
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=listdatei.21
/SET-FILE-LINK LINK-NAME=DRUCK,FILE-NAME=listdatei.51
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesamlib
                                                                  ------------- (4)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesamkonf
/.NEU REMARK
/START-PROGRAM FROM-FILE=*MOD(LIB=rootlib,ELEM=modulname,PROG-MO=ANY,-
/              RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY,-  ------- (5)
/              LO-IN=REF))
.UTM START FILEBASE=basisname
.UTM START TASKS=3
                                                                          ------ (6)
.UTM START ASYNTASKS=1
.UTM START STARTNAME=enterdatei
.FHS DE=NO
.FHS MAPLIB=formatlib
.UTM END ---------------------------------------------------------------------- (7)
.DRIVE PAR DYNAMIC LIBRARY=libname
.DRIVE PAR DYNAMIC CATALOG=projekt SCHEMA=mitarb
.DRIVE PAR DYNAMIC AUTHORIZATION=geheim
.DRIVE PAR KFKEY='K1' ACTION=BREAK UTMRC='20Z'                            (8)
.DRIVE PAR KFKEY='K3' ACTION=EXIT UTMRC='33Z'
.DRIVE ACQUIRE MEMORY 120 USER 5
.DRIVE END
```

```
.DRIVE SEQUENCE
.DRIVE PAR PLAMLIB='biblink',FORMLIB='flibname
.DRIVE PAR DD=ON,PASSWORD=OFF
.DRIVE PAR KFKEY='K1',ACTION=BREAK,UTMRC='2OZ'
.DRIVE PAR KFKEY='K3',ACTION=EXIT,UTMRC='33Z'
.DRIVE ACQUIRE MEMORY WITH 32 FOR VIEWS ———————————————— (9)
.DRIVE ACQUIRE MEMORY WITH 120 FOR 5 USER
.DRIVE ACQUIRE LIST FILE WITH DRUCK
.DRIVE ACQUIRE FILE TEST2 WITH T2
.DRIVE ACQUIRE PLAM FILE 'PLAM.BIBL1' WITH PLIB
.DRIVE END SEQUENCE
/SKIP-COMMANDS .NEU ———————————————————————————————————— (10)
/LOGOFF
```

Erläuterung:

(1) Die Startprozedur sollte eine Batch-Prozedur sein, damit die Datenstation, von der aus sie gestartet wird, nicht durch einen Dialogprozess blockiert ist.

(2) Zuweisen von Systemdateien

(3) Zuweisen von Modulbibliotheken

(4) Zuweisen der Datenbank

(5) Mit dem Kommando /START-PROGRAM rufen Sie die generierte UTM-Anwendung auf.

(6) UTM-Startparameter:

- Der Basisname für die KDCFILE wurde bei SYSPRC.UTM.xxx(GEN) beim Parameter &FILEBASE festgelegt
- Es werden drei Tasks, davon eine Task für asynchrone Vorgänge generiert.
- Für *enterdatei* geben Sie an: Name der Datei, die die Batch-Prozedur für den Start der UTM-Anwendung enthält.

(7) Ende der Eingabe von UTM-Startparametern sowie des Startparameters für das Formatierungssystem. Die UTM-Anwendung wird gestartet, d.h. als nächstes wird der START-Exit aktiviert.

(8) DRIVE-Startparameter für den New-Style-Betrieb. Siehe Anweisungen PARAMETER DYNAMIC, PARAMETER KFKEY, und ACQUIRE im „DRIVE-Lexikon“ [3]. Die UTMRC-Angaben bei PAR KFKEY müssen übereinstimmen mit den SFUNC-Angaben in der KDCDEF (siehe Abschnitt „SFUNC“ auf Seite 47).

(9) DRIVE-Startparameter für den Old-Style-Betrieb. Siehe auch Anmerkungen unten.

PARAMETER
Die Angabe PLAMLIB definiert für die Old-Style-Fassung die PLAM-Bibliothek, die als DRIVE-Bibliothek für die Prozeduren verwendet wird. Ohne Angabe gibt es keine Bibliothek für Prozeduren.

FORMLIB legt die PLAM-Bibliothek fest, in der die FHS-Formate abgelegt werden.

PASSWORD=OFF schaltet für die SESAM-Fassung den Kennwortschutz anwendungsweit ab.

KFKEY-Angaben sind sowohl im Old- als auch im New-Style erforderlich.

ACQUIRE
MEMORY FOR VIEWS definiert die Anfangsgröße des Platzhalters für benutzerspezifische Viewdefinitionen. DRIVE erweitert den Speicherplatz automatisch auf die benötigte Größe.

MEMORY FOR USER definiert die Größe des DRIVE-Cache. Wird dieser Startparameter auch bei den Startparametern für den New-Style-Betrieb angegeben, müssen die beiden Angaben übereinstimmen. Die folgende Tabelle gibt die Formeln zur Berechnung der Größe des DRIVE-Cache an:

Einsatzvariante	**Formel**	**gerundet auf**
Old-Style	mlength x 1024 x cn +96 +cn x 24	1 MByte
New-Style	mlength x 1025 x cn	1 MByte

mlength: Größe eines Speicherbereiches innerhalb des Cache-Speichers in Kilobyte

cn: Anzahl der DRIVE-UTM-Anwender, deren interne Systemdaten bei UTM-Dialogschrittwechsel parallel im Cache-Speicher zwischengespeichert werden sollen.

LIST FILE legt die zentrale Druckdatei fest. Die Old-Style-Druckdatei muss eine andere sein als die New-Style-Druckdatei.

(10) Sprung nach /.NEU

Falls die UTM-Anwendung auf Grund eines Fehlers abgebrochen wird, wird sofort ein neuer Start eingeleitet.

Die DRIVE-Startparameter für den New-Style-Betrieb müssen vor den DRIVE-Startparametern für den Old-Style-Betrieb stehen.

3.5.2.3 Startprozedur für den Old-Style-Betrieb

In einer UTM-Startprozedur für den Old-Style-Betrieb müssen Sie die Old-Style-Startparameter angeben. Fehlerhafte Startparameter führen zu Fehlermeldungen nach SYSLST und zum Abbruch des Starts der UTM-Anwendung.

Wollen Sie den DRIVE-Old-Style-Betrieb in einer UTM-Anwendung nutzen, so müssen Sie die UTM-Startprozedur wie folgt erstellen:

```
/LOGON ————————————————————————————————————————————————————————————————————— (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=basisname.SYSLOG,SHARED-UPDATE=YES ┐
                                                                             ├——— (2)
/SET-FILE-LINK FILE-NAME=basisname.KDCA,SHARED-UPDATE=YES                    ┘
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usrlib                            ┐
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022 ————————————————— (3)
/SET-FILE-LINK LINK-NAME=DRUCK,FILE-NAME=listdatei.51                        ┘
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesamlib                         ┐
                                                                             ├——— (4)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesamkonf                         ┘
/.NEU REMARK
/START-PROG modulname
.UTM START FILEBASE=basisname                                                ┐
.UTM START TASKS=3                                                           │
                                                                             ├——— (5)
.UTM START ASYNTASKS=1                                                       │
.UTM START STARTNAME=enterdatei                                              ┘
.UTM END ——————————————————————————————————————————————————————————————————— (6)
.DRIVE SEQUENCE                                                              ┐
.DRIVE PAR PLAMLIB='biblink',FORMLIB='flibname                               │
.DRIVE PAR PASSWORD=OFF                                                      │
.DRIVE PAR KFKEY='K1',ACTION=BREAK,UTMRC='20Z'                               │
.DRIVE PAR KFKEY='K3',ACTION=EXIT,UTMRC='33Z'                                │
.DRIVE ACQUIRE MEMORY WITH 32 FOR VIEWS ——————————————————————————————————— (7)
.DRIVE ACQUIRE MEMORY WITH 120 FOR 5 USER                                    │
.DRIVE ACQUIRE LIST FILE WITH DRUCK                                          │
.DRIVE ACQUIRE FILE TEST2 WITH T2                                            │
.DRIVE ACQUIRE PLAM FILE 'PLAM.BIBL1' WITH PLIB                              │
.DRIVE END SEQUENCE                                                          ┘
/SKIP-COMMANDS .NEU ———————————————————————————————————————————————————————— (8)
/LOGOFF
```

Erläuterung:

(1) Die Startprozedur sollte eine Batch-Prozedur sein, damit die Datenstation, von der aus sie gestartet wird, nicht durch einen Dialogprozess blockiert ist.

(2) Zuweisen von Systemdateien

(3) Zuweisen von Modulbibliotheken (siehe Abschnitt „Modulbibliotheken zuweisen“ auf Seite 32)

(4) Zuweisen der Datenbank

(5) UTM-Startparameter:

- Der Basisname für die KDCFILE wurde bei SYSPRC.UTM.xxx(GEN) beim Parameter &FILEBASE festgelegt.
- Es werden drei Tasks, davon eine Task für asynchrone Vorgänge generiert.
- Für *enterdatei* geben Sie an: Name der Datei, die die Batch-Prozedur für den Start der UTM-Anwendung enthält.

(6) Ende der Eingabe von UTM-Startparametern sowie des Startparameters für das Formatierungssystem. Die UTM-Anwendung wird gestartet, d.h. als nächstes wird der START-Exit aktiviert.

(7) DRIVE-Startparameter für den Old-Style-Betrieb. Siehe auch Anmerkungen unten.

PARAMETER
Die Angabe PLAMLIB definiert für die Old-Style-Fassung die PLAM-Bibliothek, die als DRIVE-Bibliothek für die Prozeduren verwendet wird. Ohne Angabe gibt es keine Bibliothek für Prozeduren.

FORMLIB legt die PLAM-Bibliothek fest, in der die FHS-Formate abgelegt werden.

PASSWORD=OFF schaltet für die SESAM-Fassung den Kennwortschutz anwendungsweit ab.

ACQUIRE
MEMORY FOR VIEWS definiert die Anfangsgröße des Platzhalters für benutzerspezifische Viewdefinitionen. DRIVE erweitert den Speicherplatz automatisch auf die benötigte Größe.

MEMORY FOR USER definiert die Größe des DRIVE-Cache. Die folgende Tabelle gibt die Formel zur Berechnung der Größe des DRIVE-Cache an:

Einsatzvariante	**Formel**	**gerundet auf**
Old-Style	mlength x 1024 x cn +96 +cn x 24	1 MByte

mlength: Größe eines Speicherbereichs innerhalb des Cache-Speichers in Kilobyte

cn: Anzahl der DRIVE-UTM-Anwender, deren interne Systemdaten bei UTM-Dialogschrittwechsel parallel im Cache-Speicher zwischengespeichert werden sollen.

LIST FILE legt die zentrale Druckdatei fest.

(8) Sprung nach /.NEU

Falls die UTM-Anwendung auf Grund eines Fehlers abgebrochen wird, wird sofort ein neuer Start eingeleitet.

3.5.3 Dialog eröffnen und beenden

Wie Sie den Dialog mit einer UTM-Produktivanwendung eröffnen und beenden, ist beschrieben im Abschnitt „Dialogablauf im UTM-Betrieb“ auf Seite 20.

3.5.4 Anwendung beenden

Um die UTM-Anwendung normal zu beenden, stehen dem Systemverwalter folgende Möglichkeiten zur Verfügung:

Ein **normales** Beenden der UTM-Anwendung ist möglich durch:

– KDCSHUT NORMAL von einer Datenstation aus

– BCLOSE von der Console aus (nur durch den Operator).

Ein **abnormales** Beenden der UTM-Anwendung ist möglich durch:

– das Administrationskommando KDCSHUT KILL

– interne UTM-Fehler

– Fehler in der Systemumgebung

– Anwenderfehler.

4 DRIVE-SQL-Anweisungen

Dieses Kapitel enthält alle neuen und geänderten DRIVE-SQL-Anweisungen. Die folgende Aufstellung gibt eine Überblick.

Neue DRIVE-SQL-Anweisungen

ALTER SPACE

ALTER STOGROUP

CREATE INDEX

CREATE SPACE

CREATE STOGROUP

CREATE SYSTEM_USER

CREATE USER

DROP INDEX

DROP SPACE

DROP STOGROUP

DROP SYSTEM_USER

DROP USER

REORG STATISTICS

UTILITY

Geänderte DRIVE-SQL-Anweisungen

ALTER TABLE

CREATE SCHEMA

CREATE TABLE

CREATE TEMPORARY VIEW

DECLARE CURSOR

DROP SCHEMA

DROP TABLE

DROP VIEW

GRANT

PRAGMA

REVOKE

ALTER SPACE - Space-Parameter ändern

ALTER SPACE ändert Parameter für einen Anwender-Space.

Welche Anwender-Spaces definiert sind, erfahren Sie im View SPACES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6].

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Space sein. Wird die Storage Group geändert, muss der aktuelle Berechtigungsschlüssel das Sonder-Privileg USAGE für die neue Storage Group besitzen.

```
ALTER SPACE space
          [ { PCTFREE prozent | NO LOG } ...]
          [ USING STOGROUP stogroup ]
```

Sie müssen mindestens einen der Parameter PCTFREE, NO LOG oder USING STOGROUP angeben und dürfen jeden Parameter nur einmal angeben.

space
Name des Space, für den Parameter geändert werden sollen.

Der einfache Spacename kann durch einen Datenbanknamen qualifiziert werden.

PCTFREE *prozent*
Freiplatzreservierung der Space-Datei in Prozent. *prozent* muss eine vorzeichenlose Ganzzahl von 0 bis 70 sein. Die geänderte Freiplatzreservierung wird erst bei der nächsten Reorganisation mit der Utility-Anweisung REORG ausgewertet.

PCTFREE *prozent* nicht angegeben:
Die Einstellung der Freiplatzreservierung bleibt unverändert.

NO LOG
Logging ausschalten.

Das Logging wird unmittelbar nach dem Beenden der aktuellen Transaktion mit der COMMIT-Anweisung ausgeschaltet.

NO LOG nicht angegeben:
Die Logging-Einstellung bleibt unverändert.

USING STOGROUP *stogroup*

Name der Storage Group, deren Platten für die Space-Datei verwendet werden sollen. Die neue Storage Group wird erst beim nächsten Reorganisieren mit der Utility-Anweisungen REORG ausgewertet.

Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden. Dieser Datenbankname muss mit dem Datenbanknamen des Space übereinstimmen.

USING STOGROUP *stogroup* nicht angegeben:
Die Storage Group für den Space bleibt unverändert.

Beispiel

Im Beispiel wird die Freiplatzreservierung und die Storage Group für einen Space geändert.

```
ALTER SPACE meinspace
    PCTFREE 60
    USING STOGROUP abraxas
```

Siehe auch

CREATE SPACE, CREATE STOGROUP

ALTER STOGROUP - Storage Group ändern

ALTER STOGROUP ändert die Definition einer Storage Group. Die Definition einer Storage Group kann nicht geändert werden, wenn die Storage Group in der Medientabelle eingetragen ist.

Welche Storage Groups definiert sind, erfahren Sie im View STOGROUPS des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE STOGROUP besitzen und Eigentümer der Storage Group sein.

```
ALTER STOGROUP stogroup

       { ADD VOLUMES (volumename,... [ON gerätetyp]) |
         DROP VOLUMES (volumename,...) |
         PUBLIC |
         TO catid
       }
```

---

stogroup
Name der Storage Group, deren Definition geändert werden soll. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

ADD VOLUMES (*volumename*,...)
Fügt neue Privatplatten zur Storage Group hinzu. *volumename* gibt das Datenträgerkennzeichen der Platten als alphanumerisches Literal an. Jedes Datenträgerkennzeichen darf nur einmal für eine Storage Group angegeben werden. Bestand die Storage Group bisher aus Privatplatten, so müssen die neu hinzugefügten Platten den gleichen Gerätetyp besitzen.
Eine Storage Group darf insgesamt max. 100 Platten umfassen.

ON *gerätetyp*
Gerätetyp der Privatplatten als alphanumerisches Literal.
Sie müssen den Gerätetyp angeben, wenn die Storage Group bisher auf gemeinschaftlichen Platten (PUBLIC) eingerichtet war.
Bestand die Storage Group bisher aus Privatplatten, so müssen Sie ON *gerätetyp* nicht angeben. Falls ON *gerätetyp* angegeben wird, muss es derselbe Gerätetyp wie bisher sein.

ON *gerätetyp* nicht angegeben:
Die Storage Group besteht aus Privatplatten, die alle den bisherigen Gerätetyp besitzen.

DROP VOLUMES (*volumename*,...)
Löscht einzelne Privatplatten aus der Definition der Storage Group. *volumename* gibt das Datenträgerkennzeichen der Platte als alphanumerisches Literal an.
Die letzte Platte einer Storage Group können Sie nicht löschen.

PUBLIC
Die Storage Group wird auf das voreingestellte Public Volume Set (PVS) der BS2000-Benutzerkennung gesetzt, unter der der DBH läuft. Alle Privatplatten werden aus der Definition der Storage Group gelöscht.

TO *catid*
Die neue Katalogkennung für die Platten wird in die Definition der Storage Group eingetragen. *catid* gibt die neue Katalogkennung als alphanumerisches Literal an.

Bei Privatplatten wird die neue Katalogkennung nur für die Katalogisierung der Dateien verwendet, die Dateien selbst sind weiterhin auf den Privatplatten gespeichert. Bei PVS wird die Katalogkennung des PVS geändert, auf dem die Storage Group liegt.

Auswirkungen von ALTER STOGROUP

Die Anweisung ALTER STOGROUP ändert nur die Definition der Storage Group. Bestehende Spaces, die die Platten der Storage Group verwenden, sind nicht betroffen.

Aus der Storage Group gelöschte Platten werden allerdings nicht für neue Speicherplatzzuweisungen an die Spaces verwendet. Platten können explizit durch DROP VOLUME aus der Storage Group gelöscht werden oder implizit durch den Wechsel von gemeinschaftlichen Platten (PUBLIC) auf Privatplatten bzw. umgekehrt.

Die neue Definition der Storage Group wird wirksam, wenn Dateien (Spaces oder Sicherungen) auf der Storage Group angelegt werden.

Beispiele

1. Das Beispiel ändert die Storage Group von Privatplatten auf das PVS mit der Katalogkennung O. Dies geschieht in zwei Schritten.

```
ALTER STOGROUP abraxas PUBLIC
ALTER STOGROUP abraxas TO 'O'
```

2. Das folgende Beispiel ändert die Storage Group ORION von PUBLIC auf Privatplatten. Die Katalogkennung für die Dateien der Storage Group bleibt unverändert.

```
ALTER STOGROUP meinedb.orion
   ADD VOLUMES ('DXO17A','DXO17B') ON 'D3475'
```

Siehe auch

CREATE STOGROUP

ALTER TABLE - Basistabelle ändern

ALTER TABLE ändert eine bestehende Basistabelle. Sie können Spalten hinzufügen, ändern oder löschen und Integritätsbedingungen hinzufügen oder löschen.
Der mit CREATE SPACE .. PCTFREE für die Freiplatzreservierung festgelegte Wert wird dabei berücksichtigt.

Bei einer CALL-DML-Tabelle können Sie nur Spalten hinzufügen, ändern oder löschen. Die für CALL-DML-Tabellen geltenden Einschränkungen sind im Abschnitt „Besonderheiten für CALL-DML-Tabellen“ auf Seite 98 beschrieben.

Beim Hinzufügen, Ändern oder Löschen der Spalte einer Tabelle (ADD, ALTER, DROP) können Sie das Pragma UTILITY MODE verwenden. Durch Einschalten des Pragmas (UTILITY MODE ON) wird die zugehörige Anweisung wie eine Utility-Anweisung außerhalb einer Transaktion durchgeführt. Sie unterbinden damit die normale Transaktionssicherung für die entsprechende Anweisung, wodurch die Performance bei umfangreichen Datenänderungen erheblich beschleunigt werden kann. Im Fehlerfall ist das Rücksetzen der Anweisung allerdings nicht mehr möglich. Der Space, auf dem die zu ändernde Basistabelle liegt, ist defekt und muss repariert werden (siehe „Pragmaklausel UTILITY MODE“ auf Seite 152).

Die Tabellenart können Sie mit ALTER TABLE nicht ändern! Änderungen der Tabellenart nehmen Sie mit der UTILITY-Anweisung MIGRATE vor (siehe Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7])

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem die Basistabelle gehört.

Diese Anweisung kann deklarative Anweisungen eines DRIVE-Programms zerstören. Ein statischer Cursor muss im Deklarationsteil eines DRIVE-Programms angelegt werden. Ändert die ALTER TABLE-Anweisung im Ausführungsteil des DRIVE-Programms die Tabelle, auf die der Cursor deklariert wurde, kann anschließend nicht mehr auf den Cursor zugegriffen werden.

Die entsprechenden Möglichkeiten bei SESAM/SQL stellen eine Erweiterung der SQL2-Norm dar.

```
ALTER TABLE tabelle

      { ADD [COLUMN] spaltendefinition,... |

        ALTER [COLUMN] { spalte
                         { DROP DEFAULT |
                           SET [(anzahl)] datentyp [CALL DML call_dml_voreinst] |
                           SET DEFAULT { alphanumerisches_literal |
                                         numerisches_literal |
                                         zeit_literal |
                                         CURRENT_DATE |
                                         CURRENT_TIME(3) |
                                         CURRENT_TIMESTAMP(3) |
                                         USER |
                                         CURRENT_USER |
                                         SYSTEM_USER |
                                         NULL
                                       }
                         }
                       }, ...
                     [ USING FILE fehlerdatei [ PASSWORD kennwort ] ] } |

        DROP [COLUMN] spalte,... { CASCADE | RESTRICT } |

        ADD [ CONSTRAINT integritätsbedingungsname ] tabellenbedingung |

        DROP CONSTRAINT integritätsbedingungsname { CASCADE | RESTRICT }

      }
```

tabelle
Name einer Basistabelle.

ADD [COLUMN] *spaltendefinition*,...
Fügt der Basistabelle neue Spalten hinzu. Die neuen Spalten werden an die vorhandenen Spalten angefügt. *spaltendefinition* definiert die Spalten.

Sie dürfen in *spaltendefinition* keine Primärschlüsselbedingung definieren.

Jeder Berechtigungsschlüssel, der Tabellenprivilegien für die zu Grundeliegende Basistabelle besitzt, erhält automatisch die entsprechenden Privilegien für die neu hinzugefügten Spalten.

ALTER [COLUMN] *spalte*
spalte ist der Name der Spalte, die geändert werden soll.

Falls temporäre Views[1] existieren, die auf der Basistabelle basieren, so werden diese gelöscht.

Die Änderungen der Spalte werden in der folgenden Reihenfolge ausgeführt:

– DROP DEFAULT

– SET *datentyp*

– SET *voreinstellung*

Für eine Spalte darf dieselbe Änderungsart nur einmal veranlasst werden.

DROP DEFAULT
Löscht die Voreinstellung (SQL-Defaultwert) für die Spalte.

Die zu Grundeliegende Basistabelle darf keine CALL-DML-Tabelle sein.

SET *datentyp*
Neuer Datentyp der Spalte.

Die Spalte, deren Datentyp geändert werden soll, darf nicht Spalte eines Primärschlüssels sein. Bei Nur-CALL-DML-Tabellen kann auch die Spalte eines Primärschlüssels angegeben werden.

Die Spalte darf nicht in Views, Indizes und Integritätsbedingungen vorkommen.

Auch der Datentyp einer multiplen Spalte darf geändert werden. Bei Änderung eines Datentyps in den Datentyp einer multiplen Spalte weist SESAM/SQL dem ersten Spaltenelement die Positionsnummer 1 zu. Die Anzahl der Spaltenelemente entspricht der Dimension des neuen Datentyps.

Eine atomare Spalte kann den Datentyp einer multiplen Spalte erhalten und umgekehrt. SESAM/SQL betrachtet den atomaren Wert dann wie den Wert einer multiplen Spalte der Dimension 1.

Der ursprüngliche Datentyp einer Spalte ist nur in bestimmte Zieldatentypen änderbar. Folgende Tabelle zeigt, welche ursprünglichen Datentypen Sie mit welchen neuen Datentypen kombinieren dürfen und welche Kombinationen nicht oder nur eingeschränkt zugelassen sind:

Ursprünglicher Datentyp	**Neuer Datentyp**						
	INTEGER SMALLINT DECIMAL NUMERIC	REAL DOUBLE PRECISION FLOAT	VAR-CHAR	CHAR	DATE	TIME(3)	TIME-STAMP(3)

[1] Temporäre Views werden ab SESAM/SQL V3.0 nicht mehr unterstützt

	Neuer Datentyp						
INTEGER SMALLINT DECIMAL NUMERIC	ja	ja [1]	nein	ja	nein	nein	nein
REAL DOUBLE PRECISION FLOAT	ja	ja	nein	ja	nein	nein	nein
VARCHAR	nein	nein	ja [2]	nein	nein	nein	nein
CHAR	ja	ja **[1]**	nein	ja	ja **[1]**	ja **[1]**	ja **[1]**
DATE	nein	nein	nein	ja	ja	nein	nein
TIME(3)	nein	nein	nein	ja	nein	ja	nein
TIME-STAMP(3)	nein	nein	nein	ja	nein	nein	ja

[1] Eine Spalte darf in die numerischen Datentypen REAL, DOUBLE PRECISION und FLOAT sowie in die Zeit-Datentypen DATE, TIME und TIMESTAMP nur dann geändert werden, wenn die zu Grunde-liegende Basistabelle eine SQL-Tabelle ist.

[2] Eine Spalte des Datentyps VARCHAR darf nur in den neuen Datentyp VARCHAR mit *neue_länge* ≥ *alte_länge* geändert werden.
Die übrigen Datentypen dürfen nicht in den Datentyp VARCHAR geändert werden und umgekehrt.

SESAM/SQL konvertiert satzweise alle Werte von *spalte* in den neuen Datentyp. Bei multiplen Spalten konvertiert SESAM/SQL die signifikanten Werte aller Ausprägungen, deren Positionsnummer kleiner oder gleich der Dimension des neuen Datentyps ist. Dabei kann sich die Position eines Elements innerhalb der multiplen Spalte ändern: Ist das Ergebnis der Konvertierung für ein Spaltenelement der NULL-Wert, werden alle nachfolgenden Elemente, deren Positionsnummer kleiner oder gleich der Dimension des neuen Datentyps ist, nach links verschoben, der NULL-Wert wird hinten angehängt.

Die Regeln zur Konvertierung eines Spaltenwertes sind in Abschnitt „Datenkonvertierung bei SQL-Anweisungen“ auf Seite 171 beschrieben.

Tritt ein Konvertierungsfehler auf, erhalten Sie eine Fehlermeldung oder Warnung.

Das Runden eines Wertes ist kein Konvertierungsfehler.

Beispiel

Eine Spalte vom Datentyp NUMERIC wird in den Datentyp INTEGER geändert. SESAM/SQL konvertiert den ursprüngliche Spaltenwert 450,25 in den Wert 450 ohne eine Warnung auszugeben.

Bei Konvertierungsfehlern unterscheidet SESAM/SQL zwischen abgeschnittenen Zeichenketten, abgeschnittenen Spaltenelementen und nicht konvertierbaren Werten:

- Abgeschnittene Zeichenketten
  Eine Spalte vom Datentyp CHARACTER soll in einen neuen CHARACTER-Datentyp mit kürzerer Länge geändert werden. Zugehörige Spaltenwerte, die länger sind, werden auf die Länge des neuen Datentyps gekürzt. Werden Zeichen entfernt, die keine Leerzeichen sind, gibt SESAM/SQL eine Warnung aus.

  Beispiel

  Der Wert 'Kundendienst' einer Spalte vom Datentyp CHARACTER(12) soll in den Datentyp CHARACTER(6) konvertiert werden. Der ursprüngliche Spaltenwert wird ersetzt durch den Wert 'Kunden'. SESAM/SQL gibt eine Warnung aus.

- Abgeschnittene Spaltenelemente
  Eine multiple Spalte enthält mindestens ein Spaltenelement, dessen Positionsnummer größer ist als die Dimension des neuen Datentyps und das einen signifikanten Wert ungleich dem NULL Wert enthält.

  Beispiel

  Eine multiple Spalte des Datentyps (7) CHARACTER (20) soll in den Datentyp (5) CHARACTER (20) konvertiert werden. Bei einigen Tabellensätzen enthalten alle 7 Elemente der multiplen Spalte einen alphanumerischen Wert.

- Nicht konvertierbare Werte
  Durch eine Datentypänderung kommt es bei bestimmten Spaltenwerten zu einem Wertverlust mit Fehlermeldung (data exception).

  Beispiele

  – Der Betrag des Wertes einer ursprünglich numerischen Spalte ist für den numerischen Zieldatentyp zu groß.

    Beispiel

    Der Wert 9999 einer Spalte vom Datentyp INTEGER soll in den Datentyp NUMERIC(2,0) konvertiert werden.

  – Eine Spalte vom Datentyp CHARACTER wird in einen numerischen Datentyp geändert. Der ursprüngliche Wert der Spalte ist nicht als numerischer Wert darstellbar.

Beispiel

Der Wert 'Otto' einer Spalte vom Datentyp CHARACTER(4) soll in den Datentyp INTEGER konvertiert werden.

– Die Länge des Wertes einer ursprünglich numerischen Spalte oder der Spalte eines Zeit-Datentyps ist für den alphanumerischen Zieldatentyp CHARACTER zu groß.

Beispiel

Der Wert 9999 einer Spalte vom Datentyp INTEGER soll in den Datentyp CHARACTER(2) konvertiert werden.

Enthält die Spaltendefinition von *spalte* eine Voreinstellung, darf der neue Datentyp keine Dimension enthalten.
Ist der voreingestellte SQL-Defaultwert ein alphanumerisches, numerisches oder Zeit-Literal, so wird er in den neuen Datentyp konvertiert. Die Konvertierung darf nicht zu einem Konvertierungsfehler führen.
Ist der voreingestellte SQL-Defaultwert eine Zeitfunktion, ein Spezial-Literal oder der NULL-Wert, wird er beibehalten.
Nach der Konvertierung muss der SQL-Defaultwert - bezogen auf den neuen Datentyp - den Zuweisungsregeln für Defaultwerte genügen.

CALL DML *call_dml_voreinst*

Ändert den nicht signifikanten Wert der Spalte einer CALL-DML-Tabelle. Darf nur für CALL-DML-Tabellen angegeben werden!
call_dml_voreinst entspricht dem nicht-signifikanten Attributwert der SESAM/SQL-Version 1.x.
Sie geben *call_dml_voreinst* als alphanumerisches Literal an.

CALL DML *call_dml_voreinst* nicht angegeben:
Bezieht sich die Datentypänderung auf die Spalte einer CALL-DML/SQL-Tabelle, so behält *spalte* den ihr in der Spaltendefinition zugeordneten nicht-signifikanten Attributwert.
Bezieht sich die Datentypänderung auf die Spalte einer Nur-CALL-DML-Tabelle, also einer Tabelle mit „alten" Attributformaten aus den SESAM-Versionen < V13.1, so erhält *spalte* folgenden nicht-signifikanten Attributwert:
– das Leerzeichen, wenn der Datentyp von *spalte* alphanumerisch ist
– die Ziffer 0, wenn der Datentyp von *spalte* numerisch ist.

SET DEFAULT *voreinstellung*

Legt einen neuen SQL-Defaultwert für die Spalte fest.

Die zu Grundeliegende Basistabelle darf keine CALL-DML-Tabelle sein.

spalte darf keine multiple Spalte sein.

voreinstellung muss den Zuweisungsregeln für Defaultwerte genügen.

Die Voreinstellung wird zu dem Zeitpunkt ausgewertet, wenn ein Satz eingefügt bzw. geändert wird und für die Spalte *spalte* der Defaultwert verwendet wird.

USING FILE *fehlerdatei* [PASSWORD *kennwort*]

Legt den Namen für die Fehlerdatei fest. *fehlerdatei* müssen Sie als alphanumerisches Literal angeben.
SESAM/SQL legt die Fehlerdatei an bzw. verwendet sie nur, wenn eine Spaltenänderung mit SET *datentyp* zu einem oder mehreren Konvertierungsfehlern führt (siehe Seite 93).

Bei Angabe einer Fehlerdatei wird eine Anweisung, die zu einem Konvertierungsfehler führt, fortgesetzt. SESAM/SQL gibt eine Warnung aus und ersetzt in der betroffenen Basistabelle die ursprünglichen Spaltenwerte durch neue Werte:

- Abgeschnittene Zeichenketten werden jeweils durch den entsprechenden gekürzten Wert ersetzt.
- Nicht konvertierbare Werte werden durch den NULL-Wert ersetzt.
- Spaltenelemente einer multiplen Spalte, deren Positionsnummer größer ist als die Dimension des neuen Datentyps, werden abgeschnitten.

SESAM/SQL protokolliert die ursprünglichen Spaltenwerte sowie abgeschnittene Spaltenelemente zusammen mit der zugehörigen Warnung oder Fehlermeldung in die Fehlerdatei.

Auch wenn UTILITY MODE ON eingeschaltet ist, wird eine Anweisung, die zu einem Konvertierungsfehler führt, nicht abgebrochen. Der Space, auf dem die zu ändernde Basistabelle liegt, bleibt in diesem Fall intakt.

Eine genaue Beschreibung der Fehlerdatei und ihres Inhalts finden Sie im Abschnitt „Fehlerdatei der SQL-Anweisung ALTER TABLE“ auf Seite 99.

PASSWORD *kennwort*

BS2000-Kennwort für die Fehlerdatei. *kennwort* muss als alphanumerisches Literal angegeben werden.

Für die Angabe von *kennwort* gibt es folgende Möglichkeiten:

- 'C''*zeichenkette*'''
  zeichenkette enthält vier abdruckbare Zeichen.
- 'X''*hex-zeichenkette*'''
  hex-zeichenkette enthält acht sedezimale Zeichen.
- '*n*'
  n bezeichnet eine ganze Zahl von - 2147483648 bis + 2147483647

USING FILE *fehlerdatei* nicht angegeben:

Führt eine Spaltenänderung mit SET *datentyp* zu Konvertierungsfehlern, protokolliert SESAM/SQL die betroffenen Spaltenwerte bzw. Spaltenelemente nicht in eine Fehlerdatei.
Abgeschnittene Zeichenketten werden auf die Länge des neuen Datentyps gekürzt und SESAM/SQL gibt eine Warnung aus.
Bei Konvertierungsfehlern wegen nicht konvertierbarer Werte oder wegen abgeschnittener Spaltenelemente bricht SESAM/SQL die zugehörige Anweisung mit Fehlermeldung ab.

DROP [COLUMN] *spalte*,... { CASCADE | RESTRICT }

Löscht eine oder mehrere Spalten der Basistabelle und die zugehörigen Indizes.
spalte ist der Name der Spalte, die gelöscht werden soll. Derselbe Spaltenname darf nur einmal angegeben werden.

Für *spalte* darf keine Primärschlüsselbedingung definiert sein.

Sie dürfen nicht alle Spalten der Basistabelle angeben.

Das Löschen einer Spalte entzieht dem aktuellen Berechtigungsschlüssel die Spaltenprivilegien UPDATE und FOREIGN KEY ... REFERENCES für diese Spalte. Wurden diese Privilegien weitergegeben, werden auch die weitergegebenen Privilegien entzogen.
Mit dem Löschen einer Spalte werden außerdem alle Views gelöscht, in deren Definition *spalte* verwendet wird, sowie alle Views, deren Definition den Namen eines solchen „übergeordneten“ View enthält.

Die Anordnung der verbleibenden Spalten der Tabelle kann sich ändern: Entsteht durch das Löschen einer Spalte eine Lücke, so werden alle nachfolgenden Spalten nach links aufgeschlossen.

CASCADE

Die benannte(n) Spalte(n) und zugehörige Indizes werden gelöscht.
Integritätsbedingungen anderer Tabellen oder Spalten, die *spalte* verwenden, werden ebenfalls gelöscht.

Die Verwendung des Pragmas UTILITY MODE ist nicht möglich. Schalten Sie das Pragma UTILITY MODE ein, so erhalten Sie eine Fehlermeldung und die Anweisung wird abgebrochen.

RESTRICT

Das Löschen einer Spalte ist nur eingeschränkt möglich:
Die Spalte kann nicht gelöscht werden, wenn sie in einer View-Definition verwendet wird. Für die zu löschende Spalte darf nur dann ein Index definiert sein, wenn keine der verbleibenden Spalten der Basistabelle in der betreffenden Indexdefinition benannt ist. Entsprechendes gilt für Integritätsbedingungen.

Das Pragma UTILITY MODE kann eingeschaltet werden.

ADD CONSTRAINT-Klausel

Fügt eine Integritätsbedingung zur Basistabelle hinzu.

CONSTRAINT *integritätsbedingungsname*

Vergibt einen Namen für die Integritätsbedingung. Der einfache Name der Integritätsbedingung kann durch einen Datenbank- und Schemanamen qualifiziert werden. Der Datenbank- und Schemaname muss mit dem Datenbank- und Schemanamen der Basistabelle übereinstimmen.

CONSTRAINT *integritätsbedingungsname* nicht angegeben:
Die Integritätsbedingung erhält einen Namen nach folgendem Schema:

{ UN | FK | CH } *integritätsbedingungsnummer*

wobei UN für UNIQUE, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16stellige Nummer.

tabellenbedingung

Gibt eine Integritätsbedingung für die Tabelle an. *tabellenbedingung* darf keine Primärschlüsselbedingung definieren.

DROP CONSTRAINT *integritätsbedingungsname* { CASCADE | RESTRICT }

Löscht die Integritätsbedingung *integritätsbedingungsname*.
integritätsbedingungsname darf keine Primärschlüsselbedingung benennen.

CASCADE

Ist *integritätsbedingungsname* eine Eindeutigkeitsbedingung, und bezieht sich die Referenzbedingung einer anderen Tabelle auf die Spalte(n), für die *integritätsbedingungsname* definiert wurde, so wird implizit auch die Referenzbedingung der anderen Tabelle gelöscht.

RESTRICT

Sie dürfen keine Eindeutigkeitsbedingung für eine Spalte löschen, solange eine Referenzbedingung einer anderen Tabelle sich auf die betreffende(n) Spalte(n) bezieht.

Besonderheiten für CALL-DML-Tabellen

Bei der ALTER TABLE-Anweisung müssen für CALL-DML-Tabellen folgende Einschränkungen berücksichtigt werden:

- Es sind nur die Klauseln ADD [COLUMN], DROP [COLUMN] und ALTER [COLUMN] mit SET *datentyp* erlaubt.
- Eine neu hinzugefügte Spalte muss eine CALL-DML-Klausel enthalten.
- Es sind nur die Datentypen CHARACTER, NUMERIC, DECIMAL, INTEGER oder SMALLINT erlaubt.
- Für die Spalte darf keine Integritätsbedingung und mit DEFAULT kein voreingestellter Wert definiert werden.
- Der Spaltenname muss sich vom Integritätsbedingungsnamen der Tabellenbedingung unterscheiden, da dieser Name als Name des zusammengesetzten Primärschlüssels verwendet wird.
- Der Datentyp der Spalte einer CALL-DML-Tabelle darf nur in den Datentyp einer CALL-DML/SQL-Tabelle geändert werden. Insbesondere darf der Datentyp einer CALL-DML-Tabelle nicht in ein „altes Attributformat“ geändert werden, d.h. in ein Attributformat der SESAM-Version <13.1.
- Ein „altes Attributformat“ einer Nur-CALL-DML-Tabelle kann in folgende Datentypen geändert werden:
  - CHARACTER mit *neue_länge* ≥ *alte_länge*
  - NUMERIC mit *alter_bruchteil=neuer_bruchteil*
  - DECIMAL mit *alter_bruchteil=neuer_bruchteil*
  - INTEGER
  - SMALLINT
- Für Spalten einer CALL-DML-Tabelle können Sie einen neuen nicht-signifikanten Attributwert vergeben. Den symbolischen Attributnamen dürfen Sie nicht ändern.
- Führt eine Datentypänderung dazu, dass der Wert einer CALL-DML-Spalte den nicht-signifikanten Attributwert annimmt, gilt der Wert der betreffenden Spalte als nicht konvertierbar. Wurde keine Fehlerdatei angegeben, gibt SESAM/SQL eine Fehlermeldung aus und bricht die Anweisung ab. Bei Angabe einer Fehlerdatei verfährt SESAM/SQL wie bei nicht konvertierbaren Werten einer SQL-Tabelle (siehe Seite 95).
- Die Tabellenart können Sie weder mit der ALTER [COLUMN]-Klausel ändern noch mit der DROP [COLUMN]-Klausel. Auch wenn die Spalten einer Nur-CALL-DML-Tabelle so geändert bzw. gelöscht wurden, dass keine Spalte ein „altes Attributformat“ enthält, bleibt die Tabellenart „Nur-CALL-DML-Tabelle“ erhalten. Änderungen der Tabellenart nehmen Sie mit der UTILITY-Anweisung MIGRATE vor (siehe Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7]).

Konvertierung von „alten“ Attributen in einer Nur-CALL-DML-Tabelle

Ein Attribut einer Nur-CALL-DML-Tabelle hat keinen expliziten Typ; der Typ wird lediglich durch die Art der Abspeicherung vorgegeben. Der Anwender sorgt selbst für die richtige Interpretation der Werte.

Bei ALTER COLUMN kann keine Änderung des Typs stattfinden, sondern nur eine Übertragung in den angegebenen Typ, wobei Werte mit passendem Typ übernommen und Werte mit nicht passendem Typ abgewiesen werden (SQLSTATE 22SA5).
Deshalb darf nur der passenden Typ angegeben werden. Eine Konvertierung in einen anderen Typ ist nur durch einen zweiten ALTER COLUMN und Angabe eines neuen Datentyps möglich.
Z.B. lässt sich ein binärer Wert nur umwandeln in INTEGER, SMALLINT. Nach dem zweiten ALTER COLUMN lässt er sich auch umwandeln in NUMERIC, DECIMAL und CHARACTER.

Bei der Bearbeitung des ALTER COLUMN wird jeder Wert gelesen und entsprechend seiner Definition in der Nur-CALL-DML-Tabelle aufbereitet. Bündigkeit, Füllbytes usw. werden dabei berücksichtigt. Es findet aber keine Konvertierung statt. Danach wird überprüft, ob der gelesene Wert dem angegebenen Format entspricht oder nicht.

Da die Attribute der Nur-CALL-DML-Tabelle auch Werte mit verschiedenem Typ enthalten können, ist es sinnvoll, bei ALTER COLUMN für „alte“ Attribute immer USING FILE *fehlerdatei* anzugeben. Alle nicht passenden Werte werden dann in der Fehlerdatei abgelegt.

Ist keine Fehlerdatei vorhanden, wird der ALTER COLUMN beim ersten nicht passenden Wert abgebrochen.

Fehlerdatei der SQL-Anweisung ALTER TABLE

Beim Ändern einer Spalte (ALTER COLUMN) können Sie den Namen einer Fehlerdatei angeben. Bei Bedarf können Sie die Fehlerdatei über ein BS2000-Kennwort schützen. Die Fehlerdatei dient zur Aufnahme von Spaltenwerten, bei denen Konvertierungsfehler auf Grund einer Datentypänderung zu Datenverlust führten.

Haben Sie eine Fehlerdatei benannt und treten bei Datentypänderungen Konvertierungsfehler auf, richtet SESAM/SQL die Fehlerdatei in der Kennung des DBH als SAM-Datei ein, sofern sie noch nicht vorhanden ist.

Soll die Fehlerdatei nicht in der DBH-Kennung liegen, müssen Sie sie vor Absetzen der ALTER TABLE-Anweisung mit dem SDF-Kommando CREATE-FILE in der gewünschten Kennung als SAM-Datei einrichten. (Die BS2000-Benutzerkennung des DBH muss die erforderlichen Zugriffsrechte besitzen).

Bei Angabe einer Fehlerdatei wird eine Anweisung, die zu einem Konvertierungsfehler führt, nicht abgebrochen. SESAM/SQL gibt eine Warnung aus und ersetzt in der betroffenen Basistabelle die ursprünglichen Spaltenwerte durch neue Werte, abhängig vom Fehlertyp jeweils durch einen gekürzten Wert oder durch den NULL-Wert.

SESAM/SQL protokolliert die ursprünglichen Spaltenwerte zusammen mit der zugehörigen Fehlermeldung oder Warnung in die Fehlerdatei. Ist bereits eine Fehlerdatei vorhanden, so wird ihr Inhalt nicht überschrieben. Die neuen Einträge fügt SESAM/SQL an die bereits vorhandenen Einträge an.

Die Fehlerdatei unterliegt nicht der Transaktionssicherung. Sie bleibt erhalten, auch wenn implizit oder explizit die Transaktion zurückgesetzt wird, in der SESAM/SQL Einträge in die Fehlerdatei schreibt.

Den Inhalt der Fehlerdatei können Sie sich mit dem SDF-Kommando SHOW-FILE am Datensichtgerät anzeigen lassen.

Inhalt der Fehlerdatei

Zu jedem protokollierten Spaltenwert enthält die Fehlerdatei einen Eintrag. Der Eintrag besteht aus dem jeweiligen SQL-Statuscode sowie aus Komponenten, die den Spaltenwert innerhalb der zugehörigen Basistabelle identifizieren.

---

eintrag :: =

satz_id
spaltenname [posnr]
sqlstate
spaltenwert

satz_id :: = { *primärschlüssel* | *satzzähler* }

---

satz_id

Identifiziert den Satz der Tabelle, die *spaltenwert* enthält.
Bei Tabellen mit Primärschlüssel ist *satz_id* der Primärschlüsselwert, der den entsprechenden Satz eindeutig identifiziert. Seine Darstellung in der Fehlerdatei entspricht der Darstellung von *spaltenwert* (siehe dort). Das gleiche gilt auch für Compound Keys.

Bei Tabellen ohne Primärschlüssel bezeichnet *satz_id* den Zähler des Satzes, der *spaltenwert* enthält. SESAM/SQL nummeriert alle Tabellensätze sequenziell durch. Der erste Satz der Tabelle erhält den Wert 1 als *satzzähler*.
satzzähler ist eine vorzeichenlose Ganzzahl.

spaltenname
Name der Spalte, die *spaltenwert* enthält. Bei multiplen Spalten enthält *spaltenname* auch die Positionsnummer des betreffenden Spaltenelements als vorzeichenlose Ganzzahl. Dabei hat das erste Element der multiplen Spalte die Positionsnummer 1.

sqlstate
SQLSTATE der zugehörigen Fehlermeldung bzw. Warnung.

spaltenwert
Ursprünglicher Spaltenwert, für den die ALTER TABLE-Anweisung zu einem Konvertierungsfehler geführt hat.
Abhängig vom Datentyp der zugehörigen Spalte, wird *spaltenwert* in der Fehlerdatei folgendermaßen dargestellt:

Datentyp der Spalte, die den ursprünglichen Wert enthält	**Darstellung des *spaltenwert* in der Fehlerdatei**
Datentyp einer CALL-DML-Tabellen-Spalte der SESAM-Version ≤ 13.1	Zeichenkette mit einer maximalen Länge von 54 Zeichen
CHARACTER CHARACTER VARYING	Zeichenkette mit einer maximalen Länge von 54 Zeichen
INTEGER, SMALLINT, NUMERIC, DECIMAL,	entsprechendes numerisches Literal (Ganzzahl oder Festpunktzahl)
FLOAT, REAL, DOUBLE PRECISION	entsprechendes numerisches Literal (Gleitpunktzahl)
DATE	Datum-Zeitliteral
TIME	Uhrzeit-Zeitliteral
TIMESTAMP	Zeitstempel-Zeitliteral

Zeichenketten werden in der Fehlerdatei ohne umgebende Hochkommata dargestellt.

Ist der ursprüngliche Wert eine Zeichenkette, die verdoppelte Hochkommata enthält, so werden diese in der Fehlerdatei als einfache Hochkommata dargestellt.

Beispiel

Das folgende Beispiel zeigt eine Fehlerdatei, die die ursprünglichen Spaltenwerte der Basistabelle „konvert“ enthält.

Die Basistabelle „konvert“ hat folgenden Aufbau:

```
CREATE TABLE KONVERT
( PKEY1             CHAR(1)
 ,PKEY2             SMALLINT
 ,PKEY3_UNIQUE1_REF SMALLINT
.
```

```
.
 ,DATUM_CHAR        CHAR(10)
    DEFAULT '1994-05-02'
    CONSTRAINT INTEGOO2 NOT NULL
.
.
 ,CONSTRAINT PKOO1 PRIMARY KEY
    (PKEY1,PKEY2,PKEY3_UNIQUE1_REF)
)
```

Die Einträge entstanden, nachdem die folgende Anweisung zu Konvertierungsfehlern führte:

```
ALTER TABLE KONVERT -
ALTER COLUMN DATUM_CHAR DROP DEFAULT, -
DATUM_CHAR SET NUMERIC(10,2) -
USING FILE 'O.KONVERTEST'
```

Auszug aus der Fehlerdatei O.KONVERTEST:

satz_id [1]	*spaltenname*	*sqlstate*	*spaltenwert*
A,2,1	DATUM_CHAR	22SA5	1994-05-02
A,3,2	DATUM_CHAR	22SA5	1994-05-02
A,4,3	DATUM_CHAR	22SA5	1994-05-02
...			
A,23,22	DATUM_CHAR	22SA5	1994-05-02
A,24,23	DATUM_CHAR	22SA5	1994-05-02

[1] Es handelt sich um einen Compound Key, der aus den Feldern PKEY1, PKEY2 und PKEY3_UNIQUE1_REF zusammengesetzt ist

Bei der Umwandlung von DATUM_CHAR von CHAR(10) in NUMERIC(10,2) kann bei den Sätzen mit dem angegebenen Compound Key der Wert 1994-05-02 nicht konvertiert werden.

Beispiel

Das folgende Beispiel löscht die Nicht-NULL-Integritätsbedingung der Spalte FIRMA der Tabelle KUNDE. Der Name der Integritätsbedingung steht im View CHECK_CONSTRAINTS des INFORMATION-SCHEMA.

```
ALTER TABLE kunde
   DROP CONSTRAINT kunde.firma_notnull RESTRICT
```

Siehe auch

CREATE TABLE

CREATE INDEX - Index erzeugen

CREATE INDEX erzeugt einen Index für eine Basistabelle. Der Index kann von SESAM/SQL verwendet werden, um Bedingungen auf einer oder mehreren Spalten des Index ohne Zugriff auf die Basistabelle auszuwerten oder um die Sätze der Tabelle in der Reihenfolge der Werte der Indexspalten auszugeben.

Die für CALL-DML-Tabellen geltenden Einschränkungen und Besonderheiten sind im Abschnitt „Besonderheiten für CALL-DML-Tabellen“ auf Seite 104 beschrieben.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem die Basistabelle gehört.

Wird der Space für den Index angegeben, muss der aktuelle Berechtigungsschlüssel Eigentümer des Space sein.

```
CREATE INDEX { index ({ spalte [ LENGTH länge ] },...) }
       ON TABLE tabelle [ USING SPACE space ]
```

---

index ({ *spalte* [LENGTH *länge*] },...)
Definition eines Index.

Wenn der Index nur eine Spalte umfasst, darf diese Spalte nicht länger als 256 Zeichen sein. Umfasst der Index mehrere Spalten, darf die Summe der Spaltenlängen plus die Gesamtzahl der Spalten 256 nicht überschreiten.

index
Name des neuen Index. Der einfache Indexname muss innerhalb des Schemas eindeutig sein. Der Indexname kann durch einen Datenbank- und Schemanamen qualifiziert werden. Datenbank- und Schemaname müssen mit dem Datenbank- und Schemanamen der Basistabelle übereinstimmen, für die der Index erzeugt wird.

Wenn Sie die CREATE INDEX-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dürfen Sie den Indexnamen nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifizieren.

spalte
Name der Spalte der Basistabelle, die zum Index gehören soll.

Eine Spalte darf nicht mehrmals im gleichen Index auftreten. Sie können mehrere Spalten für einen Index angeben (=zusammengesetzter Index). In diesem Fall darf der Index keine multiplen Spalten enthalten.

LENGTH *länge*

Gibt an, bis zu welcher Länge die Spalte in den Index einbezogen werden soll. *länge* muss eine vorzeichenlose Ganzzahl von 1 bis zur Spaltenlänge sein. Sie dürfen die Länge nur für folgende Datentypen der Spalte einschränken: CHARACTER, VARCHAR oder Datentypen von SESAM ≤ V12.

LENGTH *länge* nicht angegeben:
Die Spalte wird in ihrer ganzen Länge in den Index einbezogen.

ON TABLE *tabelle*

Name der Basistabelle, für die ein Index definiert wird.

Bei expliziter Qualifikation des einfachen Tabellennamens mit einem Datenbank- und Schemanamen muss diese mit dem Datenbank- und Schemanamen des Index übereinstimmen.

Wenn Sie die CREATE INDEX-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dürfen Sie den Tabellennamen nur dann mit dem Datenbank- und Schemanamen qualifizieren, wenn diese mit dem Datenbank- und Schemanamen der CREATE SCHEMA-Anweisung übereinstimmen.

USING SPACE *space*

Name des Space, in dem der Index gespeichert werden soll.

Der einfache Spacename kann mit dem Datenbanknamen qualifiziert werden. Dieser Datenbankname muss mit dem Datenbanknamen der Basistabelle übereinstimmen.

Der Space muss bereits für die Datenbank, zu der die Tabelle gehört, definiert sein. Der aktuelle Berechtigungsschlüssel muss Eigentümer des Space sein.

USING SPACE *space* nicht angegeben:
Der Index wird im Space der Basistabelle gespeichert.

Besonderheiten für CALL-DML-Tabellen

Bei der CREATE INDEX-Anweisung müssen für CALL-DML-Tabellen folgende Einschränkungen und Besonderheiten berücksichtigt werden:

- Jeder Index darf nur eine Spalte enthalten.
- Jede Spalte darf nur einmal in einem Index auftreten.
- Als Spaltenname im Index darf der Name der Primärschlüsselbedingung einer Datenbank mit Compound Key angegeben werden. Dadurch wird ein Index über den Primärschlüssel gelegt.

Index und Integritätsbedingung

Wenn für eine Tabelle die Integritätsbedingung UNIQUE definiert ist, wird dadurch implizit ein Index mit den bei UNIQUE angegebenen Spalten definiert. Wenn Sie mit CREATE INDEX explizit einen Index definieren, der dieselben Spalten enthält, so wird der implizit definierte Index gelöscht. Der explizite Index wird zusätzlich für die Integritätsbedingung verwendet.

Beispiel

Das folgende Beispiel erzeugt einen Index für die Spalte FIRMA der Tabelle KUNDE.

```
CREATE INDEX ind1 (firma)
    ON TABLE kunde
    USING SPACE meinspace
```

Siehe auch

DROP INDEX

CREATE SCHEMA - Schema erzeugen

CREATE SCHEMA erzeugt ein Schema. Gleichzeitig können Tabellen, Views, Privilegien und Indizes definiert werden. Das Schema kann zu einem späteren Zeitpunkt mit den jeweiligen CREATE-, ALTER- und DROP-Anweisungen verändert werden.

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE SCHEMA besitzen.

```
CREATE SCHEMA { schema [ AUTHORIZATION berechtigungsschlüssel ] |
                AUTHORIZATION berechtigungsschlüssel
              }

              [ { create_table_anweisung |
                  create_view_anweisung |
                  create_index_anweisung |
                  grant_anweisung
                } ...
              ]
```

---

schema
: Name für das Schema. Der einfache Schemaname muss innerhalb der Datenbank eindeutig sein. Der Schemaname kann durch einen Datenbanknamen qualifiziert werden.

  schema nicht angegeben:
  Der Name des Berechtigungsschlüssels in der AUTHORIZATION-Klausel wird als Schemaname verwendet.

AUTHORIZATION *berechtigungsschlüssel*
: Der Berechtigungsschlüssel wird Eigentümer des Schemas.

  Dieser Berechtigungsschlüssel wird auch als Name des Schemas verwendet, wenn kein Schemaname angegeben wird.

  AUTHORIZATION *berechtigungsschlüssel* nicht angegeben:
  Wurde für die Übersetzungseinheit ein Berechtigungsschlüssel definiert, wird dieser der Eigentümer des Schemas, sonst der aktuelle Berechtigungsschlüssel.

create/grant-anweisungen
: Werden in den CREATE- und GRANT-Anweisungen einfache Namen von Tabellen und Indizes verwendet, werden die Namen automatisch mit dem Datenbank- und Schemanamen des Schemas qualifiziert.

create_table_anweisung
: CREATE TABLE-Anweisung, die eine Basistabelle für das Schema erzeugt.

create_view_anweisung
CREATE VIEW-Anweisung, die einen View für das Schema erzeugt.

grant_anweisung
GRANT-Anweisung, die Privilegien für eine Basistabelle oder einen View vergibt. Die GRANT-Anweisung darf keine Sonder-Privilegien vergeben.

create_index_anweisung
CREATE INDEX-Anweisung, die einen Index für eine Basistabelle erzeugt.

create/grant-anweisungen nicht angegeben:
Es wird ein leeres Schema eingerichtet.

Arbeitsweise von CREATE SCHEMA

Die Anweisungen CREATE TABLE, CREATE VIEW, GRANT und CREATE INDEX, die innerhalb der CREATE SCHEMA-Anweisung angegeben werden, werden genau in der angegebenen Reihenfolge ausgeführt. Anweisungen, die sich auf bereits existierende Tabellen oder Views beziehen, müssen deshalb nach der Anweisung stehen, die diese Tabellen bzw. Views erzeugt.

Beispiel

Das Beispiel erzeugt das Schema ANDROMEDA, definiert eine Tabelle für das Schema und vergibt Privilegien dafür.

```
CREATE SCHEMA andromeda

CREATE TABLE telefonliste

   (name CHARACTER (25),
   telefon CHARACTER (15),
   fax CHARACTER (15))

GRANT ALL PRIVILEGES ON telefonliste TO hugo
```

Siehe auch

CREATE TABLE, CREATE VIEW, CREATE INDEX, GRANT, DROP SCHEMA

CREATE SPACE - Space erzeugen

CREATE SPACE erstellt einen neuen Eintrag für einen neuen Anwender-Space im Catalog der Datenbank und erzeugt die zugehörige Datei auf Betriebssystemebene.

Sie können max. 199 Anwender-Spaces pro Datenbank definieren.

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg USAGE für die verwendete Storage Group besitzen.

Wurde der Catalog der Datenbank in einer eigenen DB-Kennung angelegt, müssen Sie die Dateien der Anwender-Spaces zu dieser Datenbank vor der jeweiligen SQL-Anweisung CREATE SPACE mit dem SDF-Kommando CREATE-FILE anlegen. Wichtig ist, dass Sie die Dateien der Anwender-Spaces in der Datenbank-Kennung shareable anlegen und mit Schreibrechten versehen. Wurde die Datei des Catalog Space mit Kennwort angelegt, so ist auch für die Dateien der Anwender-Spaces ein Kennwort erforderlich. Das Kennwort muss dem BS2000-Kennwort für die Datei des Catalog Space entsprechen.

```
CREATE SPACE space
          [ AUTHORIZATION berechtigungsschlüssel ]
          [ { PRIMARY zuweisung |
              SECONDARY zuweisung |
              PCTFREE prozent |
              [NO] SHARE |
              [NO] DESTROY |
              NO LOG
            } ...
          ]
          [ USING STOGROUP stogroup ]
```

---

space
: Name für den Space. Die ersten 12 Zeichen des einfachen Spacenamens müssen innerhalb der Datenbank eindeutig sein. Der Spacename kann mit dem Datenbanknamen qualifiziert werden.

AUTHORIZATION *berechtigungsschlüssel*
: Name des Berechtigungsschlüssels, der als Eigentümer des Space eingetragen wird.

  AUTHORIZATION *berechtigungsschlüssel* nicht angegeben:
  Der aktuelle Berechtigungsschlüssel wird als Eigentümer eingetragen.

Sie dürfen jeden der folgenden Parameter PRIMARY, SECONDARY, PCTFREE, [NO] SHARE, [NO] DESTROY oder NO LOG nur einmal angeben.

PRIMARY *zuweisung*
Primärzuweisung der Space-Datei in 2K-Einheiten (BS2000-Halfpage). *zuweisung* muss eine vorzeichenlose Ganzzahl von 1 bis 8388607 sein.

PRIMARY *zuweisung* nicht angegeben:
Es gilt PRIMARY 24.

SECONDARY *zuweisung*
Sekundärzuweisung der Space-Datei in 2K-Einheiten (BS2000-Halfpage). *zuweisung* muss eine vorzeichenlose Ganzzahl von 1 bis 32767 sein.

SECONDARY *zuweisung* nicht angegeben:
Es gilt SECONDARY 24.

PCTFREE *prozent*
Freiplatzreservierung der Space-Datei in Prozent. *prozent* muss eine vorzeichenlose Ganzzahl von 0 bis 70 sein.

PCTFREE *prozent* nicht angegeben:
Es gilt PCTFREE 20.

[NO] SHARE
SHARE gibt an, dass die Space-Datei gemeinsam benutzbar ist, d.h. dass nicht nur von der BS2000-Benutzerkennung des DBH auf die Space-Datei zugegriffen werden kann.
NO SHARE gibt an, dass die Space-Datei nicht gemeinsam benutzbar ist.

Aus Datenschutzgründen ist NO SHARE zu empfehlen.

[NO] SHARE nicht angegeben:
Es gilt NO SHARE.

[NO] DESTROY
DESTROY gibt an, dass beim Löschen der Space-Datei der Speicherplatz mit binär Null überschrieben wird.
NO DESTROY gibt an, dass beim Löschen der Space-Datei nur der Speicherplatz freigegeben wird.

[NO] DESTROY nicht angegeben:
Es gilt DESTROY.

NO LOG
Kein Logging.

NO LOG nicht angegeben:
Es gilt die Logging-Einstellung, die für die Datenbank festgelegt wurde.

USING STOGROUP *stogroup*

Name der Storage Group, deren Platten für die Erzeugung der Space-Datei verwendet werden sollen.

Wird nur der einfache Name der Storage Group angegeben, wird der Name automatisch mit dem Datenbanknamen des Space qualifiziert. Wenn Sie den einfachen Namen der Storage Group mit einem Datenbanknamen qualifizieren, muss dieser mit dem Datenbanknamen des Space übereinstimmen.

USING STOGROUP *stogroup* nicht angegeben:
Es wird die voreingestellte Storage Group D0STOGROUP verwendet.

Space-Datei auf Betriebssystemebene

Die Space-Datei wird entweder unter der BS2000-Benutzerkennung angelegt, unter der der DBH läuft oder unter einer eigenen DB-Kennung. Im ersten Fall legt der DBH die Datei an, im zweiten Fall muss sie der Datenbankverwalter zuvor mit CREATE-FILE anlegen.

Die Space-Datei erhält folgenden Namen:

:*catid*:{ *dbh* | *db* }-*bs2000-benutzerkennung.datenbank.einf_spacename*

Vom einfachen Spacenamen werden nur die ersten 12 Zeichen für den Dateinamen verwendet.

Beispiel

Das Beispiel erzeugt eine Space-Datei mit den voreingestellten Primär- und Sekundärzuweisungen. Die Datei soll mehrfach benutzbar sein und beim Löschen nicht mit binär Null überschrieben werden.

```
CREATE SPACE meinspace SHARE NO DESTROY
```

Siehe auch

ALTER SPACE, CREATE STOGROUP

CREATE STOGROUP - Storage Group erzeugen

CREATE STOGROUP erzeugt eine neue Storage Group. Eine Storage Group beschreibt entweder ein PVS (Public Volume Set) oder einen Satz von Privatplatten. Die Privatplatten einer Storage Group müssen alle denselben Gerätetyp besitzen (siehe auch „SESAM/SQL-Server - Basishandbuch“ [8]).

Die Storage Group D0STOGROUP ist immer vorhanden.

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE STOGROUP besitzen.

```
CREATE STOGROUP stogroup
          { VOLUMES (volumename,...) ON gerätetyp | PUBLIC }
          [ ON catid ]
```

---

stogroup
: Name für die Storage Group. Der einfache Name der Storage Group muss innerhalb einer Datenbank eindeutig sein. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

  Der aktuelle Berechtigungsschlüssel wird Eigentümer der Storage Group und erhält das Sonder-Privileg USAGE für diese Storage Group.

VOLUMES (*volumename*,...)
: Die Storage Group wird auf Privatplatten angelegt. *volumename* gibt das Datenträgerkennzeichen der Platten als alphanumerisches Literal an. Jedes Datenträgerkennzeichen darf nur einmal angegeben werden. Sie dürfen max. 100 Platten angeben.

  Alle Platten einer Storage Group müssen den gleichen Gerätetyp besitzen.

  ON *gerätetyp*
  : Gerätetyp der Privatplatten.

PUBLIC
: Die Storage Group umfasst ein Public Volume Set (PVS).

ON *catid*

catid als alphanumerisches Literal.

Ist PUBLIC angegeben, ist dies die Katalogkennung des PVS, auf dem die Storage Group definiert ist und auf dem die Dateien angelegt werden. Bei Privatplatten (VOLUMES) ist dies das PVS, auf dem die Dateien katalogisiert werden. Die Dateien selbst liegen auf den angegebenen Privatplatten.

ON *catid* nicht angegeben:
Es wird die Katalogkennung verwendet, die der BS2000-Benutzerkennung zugewiesen ist, unter der der DBH läuft.

Beispiel

Das Beispiel erzeugt eine neue Storage Group ABRAXAS mit den angegebenen Privatplatten. Die Katalogkennung `P` soll zur Katalogisierung der Space-Dateien verwendet werden, die auf der Storage Group erzeugt werden.

```
CREATE STOGROUP abraxas
   VOLUMES ('DY130A','DY130B','DY130C','DY130D') ON 'D3475'
   ON 'P'
```

Siehe auch

DROP STOGROUP

CREATE SYSTEM_USER - Systemzugang erzeugen

Die Anweisung CREATE SYSTEM_USER definiert einen Systemzugang, d.h. sie ordnet Systembenutzern Berechtigungsschlüssel für eine Datenbank zu. Der gleiche Berechtigungsschlüssel kann für mehrere Systembenutzer gelten, und ein Systembenutzer kann mehrere Berechtigungsschlüssel besitzen.

Ein lokaler UTM-Systembenutzer wird durch den lokalen Rechnernamen, den lokalen UTM-Anwendungsnamen und die UTM-Benutzerkennung identifiziert.

Ein UTM-Systembenutzer, der über UTM-D mit SESAM-Datenbanken arbeitet, wird durch den lokalen Rechnernamen, den lokalen UTM-Anwendungsnamen und den lokalen UTM-Session-Namen (LSES) identifiziert.

Ein BS2000-(TIAM-)Systembenutzer wird durch den Rechnernamen und die BS2000-Benutzerkennung identifiziert.

Beachten Sie, dass vor dem Umzug der Datenbank auf einen anderen Rechner, zuerst ein für den neuen Rechner gültiger Systemzugang definiert werden muss.

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen. Wenn einem Berechtigungsschlüssel mit dem Sonder-Privileg CREATE USER und mit GRANT-Berechtigung (siehe Abschnitt „GRANT - Privilegien vergeben“ auf Seite 139) ein Systembenutzer zugeordnet werden soll, so muss der aktuelle Berechtigungsschlüssel ebenfalls die GRANT-Berechtigung besitzen.

```
CREATE SYSTEM_USER { utm_benutzer | bs2000_benutzer }
          FOR berechtigungsschlüssel
          AT CATALOG catalog

utm_benutzer::=( { rechnername|* },{ utm_anwendungsname|* },{ utm_benutzerkennung|* } )
bs2000_benutzer::=( { rechnername|* },[*],{ bs2000_benutzerkennung|* } )
```

---

utm_benutzer
Systemzugang eines UTM-Systembenutzers definieren.

rechnername
Symbolischer Rechnername als alphanumerisches Literal.

Falls DCAM auf dem Rechner nicht verfügbar ist, ist dem Rechner der symbolische Name ‘HOMEPROC’ fest zugeordnet.

Bei UTM-D: Angabe des lokalen Rechners, auf dem der SESAM/SQL-Datenbankanschluss generiert wurde.

* Alle Rechner.

utm_anwendungsname
Name der UTM-Anwendung als alphanumerisches Literal.

Bei UTM-D: Name der lokalen UTM-Anwendung.

* Alle UTM-Anwendungen.

utm_benutzerkennung
Bei lokalen UTM-Systembenutzern geben Sie die UTM-Benutzerkennung als alphanumerisches Literal an, die mit KDCSIGN definiert wurde. Bei UTM-D geben Sie den lokalen UTM-Session-Namen (LSES) an.

* Alle UTM-Benutzerkennungen.

bs2000_benutzer
Systemzugang eines BS2000-Systembenutzers definieren.

rechnername
Symbolischer Rechnername als alphanumerisches Literal.

Falls DCAM auf dem Rechner nicht verfügbar ist, ist dem Rechner der symbolische Name ‘HOMEPROC’ fest zugeordnet.

* Alle Rechner.

bs2000_benutzerkennung
BS2000-Benutzerkennung als alphanumerisches Literal.

* Alle BS2000-Benutzerkennungen.

FOR *berechtigungsschlüssel*
Name des bereits definierten Berechtigungsschlüssels, der dem Systembenutzer zugeordnet wird.

AT CATALOG *catalog*
Name der Datenbank, für die die Zuordnung des Berechtigungsschlüssels zum Systembenutzer gilt.

Beispiel

Im Beispiel werden zwei bereits definierten Berechtigungsschlüsseln Systembenutzer zugeordnet.

```
CREATE SYSTEM_USER (*,,'einkauf') FOR hugo AT CATALOG meinedb

CREATE SYSTEM_USER (*,,'verkauf') FOR berta AT CATALOG meinedb
```

Aus der BS2000-Benutzerkennung EINKAUF darf der Berechtigungsschlüssel HUGO auf die Datenbank MEINEDB zugreifen. Der Berechtigungsschlüssel BERTA darf von der BS2000-Benutzerkennung VERKAUF aus auf die Datenbank MEINEDB zugreifen.

Siehe auch

DROP SYSTEM_USER, CREATE USER

CREATE TABLE - Basistabelle erzeugen

CREATE TABLE erzeugt eine Basistabelle, in der die Daten permanent gespeichert sind. SESAM/SQL unterscheidet zwischen:

– SQL-Tabellen, die nur mit SQL bearbeitet werden können.

– CALL-DML/SQL-Tabellen, die mit CALL-DML und eingeschränkt mit SQL bearbeitet werden können.

– Nur-CALL-DML-Tabellen, die nur mit CALL-DML bearbeitet werden können. Diese CALL-DML-Tabellen können nicht mit CREATE TABLE erzeugt werden. Sie werden mit der MIGRATE-Anweisung erzeugt (siehe Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7]).

Nur-CALL-DML-Tabellen und CALL-DML/SQL-Tabellen werden unter dem Namen CALL-DML-Tabellen zusammengefasst.

Die für CALL-DML-Tabellen geltenden Einschränkungen bei CREATE TABLE sind im Abschnitt „Besonderheiten für CALL-DML-Tabellen“ auf Seite 118 beschrieben.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein. Wird der Space für die Basistabelle angegeben, muss der aktuelle Berechtigungsschlüssel Eigentümer des Space sein.

```
CREATE [CALL DML] TABLE tabelle

    ( { spaltendefinition | [CONSTRAINT integritätsbedingungsname] tabellenbedingung }, ...)

    [USING SPACE space]
```

CALL DML
: Erzeugt eine CALL-DML-Tabelle.
Mit SESAM CALL-DML können nur CALL-DML-Tabellen bearbeitet werden. Die Spaltendefinitionen und Integritätsbedingungen müssen bestimmte Einschränkungen erfüllen (siehe Abschnitt „Besonderheiten für CALL-DML-Tabellen“ auf Seite 118).

  CALL DML nicht angegeben:
  Eine SQL-Tabelle wird erzeugt.
  SQL-Tabellen können nur mit SQL bearbeitet werden.

TABLE *tabelle*

Name der neuen Basistabelle. Der einfache Tabellenname muss innerhalb der Basistabellen- und Viewnamen des Schemas eindeutig sein und darf nicht mit dem einfachen Namen eines temporären View übereinstimmen, falls ein solcher noch existiert[1]. Der einfache Tabellenname kann durch einen Datenbank- und Schemanamen qualifiziert werden.

Wenn Sie die CREATE TABLE-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dürfen Sie den Tabellennamen nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifizieren.

spaltendefinition

Definiert Spalten für die Basistabelle.

Sie müssen mindestens eine Spalte definieren. Eine Basistabelle kann max. 26134 Spalten eines Datentyps außer VARCHAR und max. 1000 Spalten des Datentyps VARCHAR enthalten.

Der aktuelle Berechtigungsschlüssel erhält alle Tabellen-Privilegien für die definierten Spalten.

CONSTRAINT *integritätsbedingungsname*

Vergibt einen Integritätsbedingungsnamen für die Tabellenbedingung. Der einfache Name der Integritätsbedingung muss innerhalb des Schemas eindeutig sein. Der Name der Integritätsbedingung kann durch einen Datenbank- und Schemanamen qualifiziert werden. Dieser Datenbank- und Schemaname muss mit dem Datenbank- und Schemanamen der Basistabelle übereinstimmen, für die die Integritätsbedingung erzeugt wird.

CONSTRAINT *integritätsbedingungsname* nicht angegeben:
Die Integritätsbedingung erhält einen Namen nach folgendem Schema:

{ UN | PK | FK | CH } *integritätsbedingungsnummer*

wobei UN für UNIQUE, PK für PRIMARY KEY, FK für FOREIGN KEY und CH für CHECK steht. *integritätsbedingungsnummer* ist eine 16stellige Nummer.

tabellenbedingung

Definiert eine Integritätsbedingung, die für die Basistabelle gilt.

[1] Temporäre Views werden ab SESAM/SQL V3.0 nicht mehr unterstützt

USING SPACE *space*

Name des Space, in dem die Tabelle gespeichert werden soll. Der Space muss bereits für die Datenbank definiert sein, zu dem die Tabelle gehört. Der einfache Spacename kann mit dem Datenbanknamen qualifiziert werden. Dieser Datenbankname muss mit dem Datenbanknamen der Basistabelle übereinstimmen.

USING SPACE *space* nicht angegeben:
Die Tabelle wird im voreingestellten Space des aktuellen Berechtigungsschlüssels auf der Storage Group D0STOGROUP gespeichert.
Der voreingestellte Space ist D0*berechtigungsschlüssel* mit den ersten 10 Zeichen des Berechtigungsschlüssels. Existiert dieser Space noch nicht, wird er erzeugt, wenn der aktuelle Berechtigungsschlüssel das Sonder-Privileg USAGE für die Storage Group D0STOGROUP besitzt.

Besonderheiten für CALL-DML-Tabellen

Bei der Anweisung CREATE TABLE müssen für CALL-DML-Tabellen folgende Einschränkungen berücksichtigt werden:

- Es sind nur die Datentypen CHARACTER, NUMERIC, DECIMAL, INTEGER oder SMALLINT erlaubt.
- Für eine Spalte darf mit DEFAULT kein voreingestellter Wert definiert werden.
- Eine Spalte, die nicht Primärschlüssel ist, muss eine CALL-DML-Klausel enthalten.
- Die Tabelle muss genau eine Primärschlüsselbedingung als Spaltenbedingung oder als Tabellenbedingung enthalten.
- Die Tabellenbedingung definiert einen zusammengesetzten Primärschlüssel und muss einen Namen erhalten, der dem Namen des zusammengesetzten Primärschlüssels in SESAM/SQL V1.x entspricht.
- Ein Spaltenname muss sich vom Integritätsbedingungsnamen der Tabellenbedingung unterscheiden, da dieser Name als Name des zusammengesetzten Primärschlüssels verwendet wird.

Beispiel

1. Dieses Beispiel zeigt die CREATE TABLE-Anweisung für die SQL-Tabelle AUFTRAG der Beispieldatenbank.

```
CREATE TABLE auftrag

    (anr          INTEGER CONSTRAINT anr_primary PRIMARY KEY,
    knr           INTEGER CONSTRAINT a_knr_notnull NOT NULL,
    konr          INTEGER,
    adatum        DATE DEFAULT CURRENT_DATE,
    atext         CHARACTER (30),
    fertigist     DATE,
    fertigsoll    DATE,
    astat         INTEGER DEFAULT 1 CONSTRAINT astat_notnull NOT NULL,
    CONSTRAINT a_knr_ref_kunde FOREIGN KEY (knr) REFERENCES kunde,
    CONSTRAINT konr_ref_kontakt FOREIGN KEY (konr) REFERENCES kontakt,
    CONSTRAINT astat_ref_aufstat FOREIGN KEY (astat) REFERENCES
    aufstat(astnr)
    )
```

2. Dieses Beispiel zeigt die CREATE TABLE-Anweisung für die CALL-DML-Tabelle FIRMA im Schema FIRMASCH der Datenbank CALLFIRMA (siehe „SESAM/SQL-Server - CALL-DML-Anwendungen“ [11]).

```
CREATE CALL DML TABLE callfirma.firmasch.firma

    (schluessel CHARACTER(006) PRIMARY KEY,
    aname       CHARACTER(015) CALL DML ' ' AA8,
    apreis      NUMERIC(05,02) CALL DML -0 AB6,
    abestand    NUMERIC(04) CALL DML -0 AC4,
    knachname   CHARACTER(015) CALL DML ' ' AD2,
    kvorname    CHARACTER(012) CALL DML ' ' AEZ,
    kstrasse    CHARACTER(015) CALL DML ' ' AFX,
    kpostlz     CHARACTER(005) CALL DML ' ' AGV,
    kstadt      CHARACTER(015) CALL DML ' ' AHT,
    kseit       CHARACTER(006) CALL DML ' ' AJR,
    krabatt     NUMERIC(04,02) CALL DML 0 AKP,
    .
    .
    .
    pgehalt(010) NUMERIC(07,02) CALL DML 0 AT5)
    USING SPACE CALLFIRMA.FIRMA
```

Siehe auch

ALTER TABLE, CREATE SCHEMA, CREATE SPACE

CREATE USER - Berechtigungsschlüssel erzeugen

CREATE USER erzeugt einen neuen Berechtigungsschlüssel.

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen.

```
CREATE USER berechtigungsschlüssel
            AT CATALOG catalog
```

---

berechtigungsschlüssel
Name für den Berechtigungsschlüssel. Die ersten 10 Zeichen des Berechtigungsschlüssels müssen innerhalb der Datenbank eindeutig sein.

AT CATALOG *catalog*
Name der Datenbank, für die der neue Berechtigungsschlüssel gilt.

Beispiel

Das Beispiel definiert zwei Berechtigungsschlüssel für die Datenbank MEINEDB.

```
CREATE USER berta AT CATALOG meinedb
CREATE USER hugo AT CATALOG meinedb
```

Siehe auch

DROP USER, CREATE SYSTEM_USER

DECLARE - Cursor vereinbaren

DECLARE vereinbart einen Cursor. Mit dem Cursor kann auf die einzelnen Sätze einer Ergebnistabelle zugegriffen werden. Der aktuelle Satz, auf den der Cursor zeigt, kann gelesen werden. Bei einem änderbaren Cursor können die Sätze auch geändert und gelöscht werden.

Eine statische Cursorvereinbarung muss im Programmtext vor allen Anweisungen stehen, in denen der Cursor verwendet wird. Alle Anweisungen, die diesen Cursor verwenden, müssen in derselben Übersetzungseinheit stehen.

Die statische DECLARE-Anweisung ist nicht ausführbar. Die Anweisung ist also in Programmen statisch nur am Anfang des Programms, d.h. vor den Verarbeitungsanweisungen, erlaubt.

Bei einer dynamischen Cursorvereinbarung (siehe EXECUTE-Anweisung im „DRIVE-Lexikon“ [3]) muss FOR *cursorbeschreibung* angegeben werden.

Bei DRIVE sind max. 20 dynamische und variable Cursor zulässig, bei mehr definierten Cursorn bricht das DRIVE-Programm ab. Diesen Programmabbruch können Sie mit WHENEVER &DML_STATE IN ('TOO MANY CURSORS') abfangen (siehe Anweisung WHENEVER und Handbuch „DRIVE - Programmiersprache“ [2], Abschnitt „Systemvariablen“).

Gültigkeitsbereich eines Cursors:

Der Gültigkeitsbereich eines Cursors wird aufgehoben bei
- Programmende (Ende der Lebensdauer erst bei Anwendungsende oder Transaktionsende)
- Programmabbruch
- DRIVE-Ende (STOP)
- DROP CURSOR *cursor* (DRIVE-Anweisung, im Dialog-Modus, dynamisch oder bei variablem Cursor)
- DROP CURSORS (DRIVE-Anweisung, nur im Dialog-Modus oder dynamisch)

Beim Übergang vom Dialog- in den Programm-Modus von DRIVE bleibt die Cursordefinition erhalten, die Cursorposition hingegen nicht, weil bei diesem Übergang keine Transaktion offen sein darf. Sie können die Cursorposition allerdings mit STORE speichern, falls es sich nicht um einen PREFETCH-Cursor handelt.

Im Programm-Modus von DRIVE bleibt ein Cursor, der mit PERMANENT definiert wurde, über das Ende eines mit CALL gerufenen Programms hinaus mit seiner Cursorposition erhalten. Ein Cursor, der mit TEMPORARY definiert wurde, wird bei Programmende geschlossen und bei einem COMMIT WORK auf höherer Programmebene gelöscht.

Der Gültigkeitsbereich des Cursors im Programm-Modus wird in jedem Falle aufgehoben beim Übergang in den Dialog-Modus, bei Programmabbruch und bei DRIVE-Ende (STOP bzw. COMMIT WORK WITH STOP).

```
DECLARE cursor [ { PERMANENT | TEMPORARY } ]
               [ SCROLL ] CURSOR
               [ PREFETCH n ]
               [ FOR cursorbeschreibung ]

cursorbeschreibung::= abfrageausdruck [ ORDER BY { spalte | spalte(posnr) | spaltennummer }
                                        [ { ASCENDING | DESCENDING } ] ]
                                      [ FOR { UPDATE [OF spalte,...] | READ ONLY } ]
```

cursor
Name für den Cursor. Innerhalb einer Übersetzungseinheit dürfen nicht mehrere Cursor mit demselben Namen vereinbart werden. Der Gültigkeitsbereich des Cursors ist auf die Übersetzungseinheit beschränkt, in der der Cursor vereinbart wurde.

PERMANENT
PERMANENT funktioniert innerhalb von Programmen, die mit CALL aufgerufen werden.
Die Position des Cursors bleibt über das Ende des mit CALL gerufenen Programms hinaus erhalten, wenn weder im gerufenen Programm noch im rufenden Programm zwischen den CALL-Aufrufen ein COMMIT WORK durchlaufen wird. Die Cursorposition wird nur aufgehoben, wenn der Cursor explizit mit CLOSE geschlossen wird bzw. beim Übergang in den Dialog-Modus.
Beim ersten Ablauf des mit CALL gerufenen Programms muss der Cursor mit der OPEN-Anweisung geöffnet werden, bei weiteren Abläufen darf keine OPEN-Anweisung mehr auf diesen Cursor gegeben werden.
Im rufenden Programm darf keine COMMIT WORK-Anweisung stehen.

TEMPORARY
Vorbelegung
TEMPORARY funktioniert innerhalb von Programmen, die mit CALL aufgerufen werden.
Der Cursor wird beim Ende des mit CALL gerufenen Programms geschlossen, seine Position aufgehoben (Gültigkeitsbereich beendet). Beim nächsten COMMIT WORK auf höherer Programmebene wird der Cursor gelöscht (Lebensdauer beendet).

SCROLL
Der Cursor kann mit FETCH NEXT/PRIOR/FIRST/LAST/RELATIVE/ABSOLUTE in beliebiger Reihenfolge auf jeden Satz der Ergebnistabelle positioniert werden.

SCROLL dürfen Sie nur angeben, wenn für die Cursorbeschreibung von *cursor* keine FOR UPDATE-Klausel vereinbart wird.

Wenn Sie SCROLL angeben, ist der Cursor *cursor* nicht änderbar, es gilt implizit die FOR READ ONLY-Klausel.

SCROLL nicht angegeben:
Der Cursor kann nur auf den jeweils nächsten Satz positioniert werden. Bei FETCH ist nur die Positionsangabe NEXT erlaubt.

PREFETCH *n*

Die PREFETCH-Klausel steigert die Performance durch den sog. Schubmodus.

Alternativ zur PREFETCH-Klausel kann der Schubmodus durch eine PRAGMA-Anweisung mit der Pragmaklausel PREFETCH aktiviert werden (siehe Abschnitt „PRAGMA - Pragmaklauseln vereinbaren“ auf Seite 144). Beide Arten der Schubmodusaktivierung sind funktional gleichwertig. Voraussetzung ist, dass mit einer SESAM/SQL-Version ab 2.1 gearbeitet wird.

Wird mit SESAM/SQL V2.0 gearbeitet, so führt die Verwendung von PREFETCH-Klauseln oder PREFETCH-Pragmas zu SESAM-Warnungen (SQLSTATE Klasse 01).

Ein Cursor, bei dem auf eine der beiden Arten der Schubmodus aktiviert wurde, heißt **PREFETCH-Cursor**.

Folgende Anweisungen sind bei einem PREFETCH-Cursor nicht erlaubt:
FETCH PRIOR, FIRST, LAST, RELATIVE, ABSOLUTE (nur Positionieren mit FETCH NEXT erlaubt)
STORE und RESTORE
DELETE... WHERE CURRENT OF...
UPDATE... WHERE CURRENT OF...

n

Schubfaktor *n*-1 gibt die Anzahl der Sätze an, die bei der ersten FETCH-Anweisung nach OPEN von SESAM in einen Puffer eingelesen werden (Schub). Bei der 2. bis n-ten FETCH-Anweisung muss nicht auf die Datenbank zugegriffen werden. *n* ist eine Ganzzahl vom Datentyp SMALLINT. *n* muss größer oder gleich 2 und kleiner oder gleich 32000 sein. Ist l die Summe der Längen aller selektierten Satzelemente, dann sollte n * l kleiner oder gleich 30000 sein. Bei einer Bildschirmausgabe kann die Anzahl der Sätze, die auf einen Bildschirm ausgegeben werden können, als Richtlinie dienen. In Abhängigkeit von l werden ggf. weniger als *n*-1 Sätze in den Schubpuffer eingelesen.

FOR-Klausel

Die FOR-Klausel kann nur im Programm-Modus bei einer statischen Cursordeklaration weggelassen werden. Wird sie weggelassen, so wird ein so genannter **variabler Cursor** deklariert. Ein solcher Cursor wird erst durch eine nachfolgende dynamische Deklaration mit FOR-Klausel gegenüber dem Datenbanksystem bekannt. Bei der dynamischen Deklaration mit EXECUTE müssen Sie die FOR-Klausel angeben. Bis auf die FOR-Klausel müssen die statische und die dynamische Deklaration eines Cursors gleich sein. Innerhalb der PREFETCH-Klausel darf jedoch der Schubfaktor *n* variiert werden. Alle anderen Cursor-Anweisungen (OPEN, FETCH, CLOSE, DROP CURSOR, STORE, RESTORE, UPDATE, DELETE, CYCLE) können für den variablen Cursor statisch angegeben werden, wodurch die Performance steigt (siehe Handbuch „DRIVE - Programmiersprache“ [2], Abschnitt „Dynamische SQL-Anweisungen“).

cursorbeschreibung

Statischen oder dynamischen Cursor vereinbaren.

cursorbeschreibung definiert die Ergebnistabelle und die Eigenschaften des Cursors. Ein Satz der Ergebnistabelle wird frühestens dann bestimmt, wenn der Cursor mit OPEN geöffnet wird, und spätestens bei einem FETCH.

abfrageausdruck

Abfrage-Ausdruck zur Auswahl von Sätzen und Spalten aus Basistabellen oder Views.

Der Wert für Benutzervariablen in *abfrageausdruck* wird erst beim Öffnen des Cursors ermittelt. Die Literale CURRENT_USER und SYSTEM_USER sowie Zeitfunktionen, die in *abfrageausdruck* vorkommen, werden erst beim Öffnen des Cursors ausgewertet.

ORDER BY

Die ORDER BY-Klausel bezeichnet die Spalten, nach denen die Ergebnistabelle sortiert werden soll. Es wird zuerst nach den Werten der ersten angegebenen Spalte sortiert. Wenn Sätze in der ersten Spalte vorkommen, die nach den Vergleichsregeln gleiche Werte haben, werden diese gemäß der zweiten Sortierspalte sortiert usw. In SESAM/SQL sind NULL-Werte beim Sortieren kleiner als alle Nicht-NULL-Werte.

Die Reihenfolge der Sätze mit gleichen Werten in allen Sortierspalten ist undefiniert.

ORDER BY dürfen Sie nur angeben, wenn für die Cursorbeschreibung von *cursor* keine FOR UPDATE-Klausel vereinbart wird.

Wenn Sie ORDER BY angeben, ist der Cursor *cursor* nicht änderbar, es gilt implizit die FOR READ ONLY-Klausel.

ORDER BY nicht angegeben:
Die Reihenfolge der Sätze der Cursortabelle ist undefiniert.

spalte

Name der Spalte, nach der sortiert werden soll. Die Spalte muss Teil der Ergebnistabelle sein, die mit *abfrageausdruck* erzeugt wird.

Für *spalte* können Sie eine einfache Spalte angeben. Der Spaltenname darf nicht mit einer Tabellenangabe qualifiziert werden.

spalte(*posnr*)
Element einer multiplen Spalte, nach dem sortiert werden soll. Das Spaltenelement muss Teil der Ergebnistabelle sein, die mit *abfrageausdruck* erzeugt wird.

posnr ist eine vorzeichenlose Ganzzahl, die die Positionsnummer des Spaltenelements in der multiplen Spalte angibt.

spaltennummer
Nummer der Spalte, nach der sortiert werden soll.

spaltennummer ist eine vorzeichenlose Ganzzahl mit:

1 ≤ *spaltennummer* ≤ Anzahl der Ergebnisspalten.

Durch die Angabe der Spaltennummer können Sie auch Spalten als Sortierbegriff verwenden, die keinen oder keinen eindeutigen Spaltennamen haben.

spaltennummer kann eine einfache Spalte oder eine multiple Spalte mit Dimension 1 bezeichnen.

ASCENDING
Die Werte der zugehörigen Spalte werden aufsteigend sortiert.

DESCENDING
Die Werte der zugehörigen Spalte werden absteigend sortiert.

FOR READ ONLY
Die FOR READ ONLY-Klausel legt fest, dass der Cursor *cursor* nur zum Lesen der Sätze der Ergebnistabelle verwendet werden kann (Nicht-änderbarer Cursor oder Cursor zum Lesen).

Ist der zu Grundeliegende Abfrageausdruck nicht änderbar, so gilt die FOR READ ONLY-Klausel implizit (siehe Abschnitt „Änderbarkeit von Abfrageausdrücken“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]). Sie gilt auch dann, wenn in der Cursorvereinbarung SCROLL oder ORDER BY angegeben wurden.

FOR UPDATE

Die FOR UPDATE-Klausel darf nur angegeben werden, wenn der zu Grundeliegende Abfrageausdruck änderbar ist (siehe Abschnitt „Änderbarkeit von Abfrageausdrücken“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]) und weder SCROLL noch ORDER BY angegeben wurden. Mit der FOR UPDATE-Klausel können Sie festlegen, welche Spalten der zu Grundeliegenden Tabelle über den Cursor mit UPDATE...WHERE CURRENT OF geändert werden dürfen.

Wurde für den betreffenden Cursor ein Pragma PREFETCH vereinbart, so schaltet die FOR UPDATE-Klausel die Wirksamkeit dieses Pragmas aus (siehe Abschnitt „Pragmaklausel PREFETCH“ auf Seite 151).

FOR UPDATE nicht angegeben:
Ist der Cursor änderbar (siehe unten) und wurde keine FOR READ ONLY-Klausel angegeben, können alle Spalten der zu Grundeliegenden Tabelle mit UPDATE...WHERE CURRENT OF geändert werden.

OF *spalte*,...
Nur die angegebenen Spalten dürfen mit UPDATE...WHERE CURRENT OF geändert werden.
Für *spalte* geben Sie den Namen einer Spalte der Tabelle an, auf die sich der änderbare Cursor bezieht. *spalte* ist der einfache Name der Spalte aus der zu Grundeliegenden Tabelle, auch wenn im Abfrage-Ausdruck der Cursorbeschreibung neue Spaltennamen definiert werden.

Beispiel

Es wird ein änderbarer Cursor CUR vereinbart. Die zu Grundeliegende Tabelle ist TAB. Nur die Spalte SP der Tabelle TAB darf über den Cursor CUR geändert werden. Dazu wird in der Cursorbeschreibung eine FOR UPDATE-Klausel mit dem Spaltennamen SP angegeben.

```
DECLARE cur CURSOR FOR

    SELECT korr.sp AS spalte FROM tab AS korr

    FOR UPDATE OF sp
```

In der FOR UPDATE-Klausel wird der einfache ursprüngliche Spaltenname SP verwendet, obwohl in der SELECT-Liste der Spaltenname und in der FROM-Klausel die Tabelle umbenannt wurden.

OF *spalte*,... nicht angegeben:
Jede Spalte der zu Grundeliegenden Tabelle kann mit UPDATE...WHERE CURRENT OF geändert werden.

Änderbarer Cursor

Nur wenn der Cursor änderbar ist, kann er zum Ändern oder Löschen mit den Anweisungen UPDATE... WHERE CURRENT OF... bzw. DELETE... WHERE CURRENT OF... verwendet werden. Ein Cursor ist änderbar, wenn seine Cursorbeschreibung änderbar ist, d.h. der zu Grundeliegende Abfrageausdruck ist änderbar, und es ist keine ORDER-BY-Klausel angegeben (siehe Metavariable *abfrageausdruck*). Zusätzlich darf in der Cursorvereinbarung keine SCROLL-Klausel angegeben werden. Ist die PREFETCH-Klausel angegeben, so kann auch ein änderbarer Cursor nicht zum Ändern oder Löschen verwendet werden.

Beispiel 1

Eine Cursordeklaration soll in der Cursorbeschreibung variabel gehalten werden. Statisch wird eine DECLARE ... CURSOR-Anweisung ohne FOR-Klausel angegeben.

```
DECLARE cur_anzeige SCROLL CURSOR;
```

Mit einer EXECUTE-Anweisung wird die Cursorbeschreibung zum Ablaufzeitpunkt nachdeklariert. (Der Ausdruck in Hochkommas darf max. 256 Zeichen lang sein, andernfalls muss mit CONCAT gearbeitet werden. Dabei zählt DRIVE einen Teil der Blanks für die Einrükkung des Codes mit.)

```
EXECUTE 'DECLARE cur_anzeige SCROLL CURSOR FOR '||
              'SELECT land, nachname, vorname, gehalt '||
        'FROM db_mitarbeiter '||
        'WHERE (land = &hland) ' ||
               ' AND (gehalt >= &g_unter) AND (gehalt <= &g_ober);';
```

Alle Anweisungen, die sich auf den Cursor beziehen, können statisch im Programm angegeben werden.

```
CYCLE cur_anzeige INTO &anzeigesatz.*;
DISPLAY FORM LINE &anzeigesatz;
END CYCLE;
DROP CURSOR cur_anzeige;
...
COMMIT WORK;
EXECUTE 'DECLARE cur_aendern CURSOR FOR ' ||
        'SELECT land, nachname, vorname, gehalt ' ||
         'FROM db_mitarbeiter ' ||
         'WHERE (land = &hland) AND
               (gehalt = &höchstsatz);';
```

Zwischen der DROP-Anweisung und einer EXECUTE 'DECLARE ...'-Anweisung auf den gleichen Cursornamen (gemäß statischer Cursordeklaration ohne FOR-Klausel) sollte ein COMMIT WORK stehen.

Beispiel 2

Es wird ein änderbarer Cursor `cur` vereinbart. Die zu Grundeliegende Tabelle ist `tab`. Nur die Spalte `sp` der Tabelle `tab` darf über den Cursor `cur` geändert werden. Dazu wird in der Cursorbeschreibung eine FOR UPDATE-Klausel mit dem Spaltennamen `sp` angegeben.

```
DECLARE cur CURSOR FOR
   SELECT korr.sp AS spalte FROM tab AS korr
   FOR UPDATE OF sp;
```

In der FOR UPDATE-Klausel wird der einfache ursprüngliche Spaltenname `sp` verwendet, obwohl in der SELECT-Liste der Spaltenname und in der FROM-Klausel die Tabelle umbenannt wurden.

Beispiel 3

Es wird ein statischer Cursor mit einer Eingabevariablen vereinbart.

```
DECLARE cur_auftrag CURSOR FOR
   SELECT anr, adatum, atext, astat
   FROM auftrag
   WHERE knr >= &KUNDENNR;
```

Bei OPEN *cur_auftrag* wird die Cursorbeschreibung mit dem aktuellen Wert von &KUNDENNR ausgewertet. Bei RESTORE *cur_auftrag* bleibt die Cursorbeschreibung des letzten OPEN *cur_auftrag* gültig.

Siehe auch

CLOSE, DELETE, FETCH, INSERT, OPEN, SELECT, UPDATE

DROP INDEX - Index löschen

DROP INDEX löscht einen Index. Der Index kann entweder explizit durch die Anweisung CREATE INDEX oder implizit durch die Definition einer Integritätsbedingung (UNIQUE, PRIMARY KEY) erzeugt worden sein.

Welche Indizes definiert sind, erfahren Sie im View INDEXES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Wird ein explizit definierter Index zusätzlich von einer Integritätsbedingung verwendet, wird der Index nicht gelöscht, sondern in einen impliziten Index umbenannt. Der neue Indexname beginnt mit UI, gefolgt von einer 16stelligen Nummer.

Indizes, die nur implizit durch Integritätsbedingungen (UNIQUE, PRIMARY KEY) erzeugt wurden, werden erst dann gelöscht, wenn die zugehörige Integritätsbedingung gelöscht wird.

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem der Index gehört.

```
DROP INDEX index
```

---

index
Name des Index, der gelöscht werden soll.

Der einfache Name des Index kann durch einen Datenbank- und Schemanamen qualifiziert werden.

Siehe auch

CREATE INDEX

DROP SCHEMA - Schema löschen

DROP SCHEMA löscht ein Datenbankschema.

Welche Schemata definiert sind, erfahren Sie im View SCHEMATA des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein.

```
DROP SCHEMA schema { CASCADE | RESTRICT }
```

---

schema
Name des Schemas.

Der einfache Name des Schemas kann durch einen Datenbanknamen qualifiziert werden.

CASCADE
Das Schema *schema* und alle Objekte des Schemas werden gelöscht. Auch Views und Integritätsbedingungen, die sich auf die in *schema* enthaltenen Basistabellen oder Views beziehen, werden gelöscht.

RESTRICT
Das Löschen des Schemas *schema* ist nur möglich, wenn es leer ist. Alle Basistabellen, Views und Integritätsbedingungen des Schemas müssen zuvor gelöscht werden.

Siehe auch

CREATE SCHEMA, DROP TABLE, DROP VIEW

DROP SPACE - Space löschen

DROP SPACE löscht einen leeren Anwender-Space. Alle Basistabellen und Indizes des Space müssen zuvor gelöscht werden. Die Space-Datei wird mit binär Null überschrieben, wenn dies beim Erzeugen oder Ändern des Space mit dem Parameter DESTROY festgelegt wurde.

Welche Anwender-Spaces definiert sind, erfahren Sie im View SPACES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Space sein.

```
DROP SPACE space RESTRICT
```

---

space
Name des Space. Der Space muss leer sein.

Der einfache Name des Space kann durch einen Datenbanknamen qualifiziert werden.

DROP SPACE und Transaktionen

Nach der Anweisung DROP SPACE darf in derselben Transaktion keine CREATE SPACE-Anweisung folgen.

Siehe auch

CREATE SPACE, ALTER SPACE

DROP STOGROUP - Storage Group löschen

DROP STOGROUP löscht eine Storage Group. Eine Storage Group kann nicht gelöscht werden, wenn sie für Spaces verwendet wird oder in der Medientabelle eingetragen ist (siehe „SESAM/SQL-Server - Basishandbuch“ [8]).

Welche Storage Groups definiert sind, erfahren Sie im View STOGROUPS des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer der Storage Group sein.

```
DROP STOGROUP stogroup RESTRICT
```

---

stogroup

Name der Storage Group. Die Storage Group muss unbenutzt sein.

Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

Siehe auch

CREATE STOGROUP, ALTER STOGROUP

DROP SYSTEM_USER - Systemzugang löschen

DROP SYSTEM_USER löscht einen Systemzugang, d.h. die Zuordnung eines Berechtigungsschlüssels zu einem Systembenutzer. Es muss eine Kombination von Systembenutzer und Berechtigungsschlüssel angegeben werden, für die mit CREATE SYSTEM_USER ein Systemzugang definiert wurde.

Ein Systemzugang kann nicht gelöscht werden, wenn er die letzte Zuordnung eines Systembenutzers zum Berechtigungsschlüssel des universellen Benutzers ist.

Ist momentan eine SQL-Transaktion des Systembenutzers aktiv, so wird dessen Systemzugang nur gelöscht, wenn noch ein anderer Systemzugang für den Systembenutzer existiert.

Welche Berechtigungsschlüssel welchen Systembenutzern zugeordnet sind, erfahren Sie im View SYSTEM_ENTRIES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen. Wenn die Zuordnung eines Berechtigungsschlüssels mit dem Sonder-Privileg CREATE USER und mit GRANT-Berechtigung (siehe Abschnitt „GRANT - Privilegien vergeben“ auf Seite 139) zu einem Systembenutzer gelöscht werden soll, so muss der aktuelle Berechtigungsschlüssel zusätzlich die GRANT-Berechtigung besitzen.

```
DROP SYSTEM_USER { utm_benutzer | bs2000_benutzer }
          FOR berechtigungsschlüssel
          AT CATALOG catalog

utm_benutzer::=( { rechnername|* },{ utm_anwendungsname|* },{ utm_benutzerkennung|* } )
bs2000_benutzer::=( { rechnername|* },[*],{ bs2000_benutzerkennung|* } )
```

---

utm_benutzer
: Systemzugang eines UTM-Systembenutzers löschen.

  Die Angabe des UTM-Benutzers muss exakt so erfolgen, wie sie mit CREATE SYSTEM_USER definiert wurde. * bedeutet den Systemzugang, der mit * definiert wurde, nicht alle entsprechenden Systemzugänge.

  rechnername
  : Symbolischer Rechnername als alphanumerisches Literal.

Falls DCAM auf dem Rechner nicht verfügbar ist, ist dem Rechner fest der symbolische Name ‘HOMEPROC’ zugeordnet.

* Alle Rechner.

utm_anwendungsname
Name der UTM-Anwendung als alphanumerisches Literal.

* Alle UTM-Anwendungen.

utm_benutzerkennung
Bei lokalen UTM-Systembenutzern geben Sie die UTM-Benutzerkennung als alphanumerisches Literal an, die mit KDCSIGN definiert wurde. Bei UTM-D geben Sie den Namen der lokalen UTM-Session (LSES) an.

* Alle UTM-Benutzerkennungen.

bs2000_benutzer
Systemzugang eines BS2000-Systembenutzers löschen.

Die Angabe des BS2000-Benutzers muss exakt so erfolgen, wie sie mit CREATE SYSTEM_USER definiert wurde. * bedeutet den Systemzugang, der mit * definiert wurde, nicht alle entsprechenden Systemzugänge.

rechnername
Symbolischer Rechnername als alphanumerisches Literal. Falls DCAM auf dem Rechner nicht verfügbar ist, ist dem Rechner fest der symbolische Name ‘HOMEPROC’ zugeordnet.

* Alle Rechner.

bs2000_benutzerkennung
BS2000-Benutzerkennung als alphanumerisches Literal.

* Alle BS2000-Benutzerkennungen.

FOR *berechtigungsschlüssel*
Name des Berechtigungsschlüssels, dem der Systembenutzer zugeordnet ist.

AT CATALOG *catalog*
Name der Datenbank, für die die Zuordnung des Systembenutzers zum Berechtigungsschlüssel gelöscht werden soll.

Beispiel

Im Beispiel werden zwei Systemzugänge gelöscht. Die Systemzugänge müssen genauso angegeben werden, wie sie mit CREATE SYSTEM_USER definiert wurden. Die Berechtigungsschlüssel HUGO und BERTA werden nicht gelöscht.

```
DROP SYSTEM_USER (*,,'einkauf') FOR hugo AT CATALOG meinedb

DROP SYSTEM_USER (*,,'verkauf') FOR berta AT CATALOG meinedb
```

Siehe auch

CREATE SYSTEM_USER, CREATE USER, DROP USER

DROP TABLE - Basistabelle löschen

DROP TABLE löscht eine Basistabelle und die zugehörigen Indizes.

Das Löschen einer Basistabelle entzieht dem aktuellen Berechtigungsschlüssel alle Tabellen- und Spalten-Privilegien für diese Basistabelle. Tabellen- und Spalten-Privilegien, die weitergegeben wurden, werden ebenfalls entzogen.

Welche Basistabellen definiert sind, erfahren Sie im View BASE_TABLES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem die Tabelle gehört.

i Diese Anweisung kann deklarative Anweisungen eines DRIVE-Programms zerstören.

```
DROP TABLE tabelle { CASCADE | RESTRICT }
```

---

tabelle
Name der Basistabelle, die gelöscht werden soll.

CASCADE
Die Basistabelle *tabelle* und alle zugehörigen Indizes werden gelöscht. Außerdem werden alle Views und Integritätsbedingungen, die sich auf *tabelle* beziehen, gelöscht.

RESTRICT
Das Löschen der Basistabelle *tabelle* ist nur eingeschränkt möglich. Die Basistabelle *tabelle* kann nicht gelöscht werden, wenn sie in einer View-Definition oder in einer Integritätsbedingung einer anderen Basistabelle verwendet wird.

Siehe auch

CREATE TABLE, ALTER TABLE

DROP USER - Berechtigungsschlüssel löschen

DROP USER löscht einen Berechtigungsschlüssel und die zugehörigen Systemzugänge. Ein Berechtigungsschlüssel kann nicht gelöscht werden, wenn er Eigentümer von Schemas, Spaces oder Storage Groups ist, wenn er Grantor eines Privilegs ist oder wenn momentan eine SQL-Transaktion des Berechtigungsschlüssels aktiv ist.

Es werden alle temporären Views des angegebenen Berechtigungsschlüssels in der angegebenen Datenbank gelöscht, falls solche noch existieren[1].

Der Berechtigungsschlüssel des universellen Benutzers kann nicht gelöscht werden.

Welche Berechtigungsschlüssel definiert sind, erfahren Sie im View USERS des INFORMATION_SCHEMA. Welcher Berechtigungsschlüssel Eigentümer ist, ist in den Views SCHEMATA, SPACES und STOGROUPS abgelegt. Ob der Berechtigungsschlüssel Grantor eines Privilegs ist, erfahren Sie aus den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES und CATALOG_PRIVILEGES (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss das Sonder-Privileg CREATE USER besitzen. Wenn ein Berechtigungsschlüssel mit dem Sonder-Privileg CREATE USER und mit GRANT-Berechtigung (siehe Abschnitt „GRANT - Privilegien vergeben“ auf Seite 139) gelöscht werden soll, so muss der aktuelle Berechtigungsschlüssel zusätzlich die GRANT-Berechtigung besitzen.

```
DROP USER berechtigungsschlüssel AT CATALOG catalog RESTRICT
```

---

berechtigungsschlüssel
: Name des Berechtigungsschlüssels, der gelöscht werden soll.

AT CATALOG *catalog*
: Name der Datenbank, aus der der Berechtigungsschlüssel gelöscht werden soll.

Siehe auch

CREATE USER, CREATE SYSTEM_USER, DROP SYSTEM_USER

[1] Temporäre Views werden ab SESAM/SQL V3.0 nicht mehr unterstützt

DROP VIEW - View löschen

DROP VIEW löscht die Definition eines View.

Das Löschen einer View-Definition entzieht dem aktuellen Berechtigungsschlüssel alle Tabellen- und Spalten-Privilegien für diesen View. Tabellen- und Spalten-Privilegien des View, die weitergegeben wurden, werden ebenfalls entzogen.

Welche Views definiert sind, erfahren Sie im View VIEWS des INFORMATION_SCHEMA. Von welchen Tabellen Views abhängig sind, ist im View VIEW_TABLE_USAGE des INFORMATION_SCHEMA abgelegt (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Der aktuelle Berechtigungsschlüssel muss Eigentümer des Schemas sein, zu dem der View gehört.

```
DROP VIEW tabelle { CASCADE | RESTRICT }
```

---

tabelle
: Name des View, der gelöscht werden soll.

CASCADE
: Der View *tabelle* und alle Views, in deren Definition *tabelle* verwendet wird, werden gelöscht.

RESTRICT
: Das Löschen des View *tabelle* ist nur eingeschränkt möglich. Der View kann nicht gelöscht werden, wenn er in einer anderen View-Definition verwendet wird.

Siehe auch

CREATE VIEW

GRANT - Privilegien vergeben

GRANT vergibt Tabellen- und Spalten-Privilegien für Basistabellen und Views und Sonder-Privilegien für Datenbanken und Storage Groups. Ist die GRANT-Anweisung in einer CREATE SCHEMA-Anweisung enthalten, darf die GRANT-Anweisung keine Sonder-Privilegien vergeben.

Der aktuelle Berechtigungsschlüssel muss die angegebenen Privilegien vergeben dürfen:

– Er ist der Berechtigungsschlüssel des universellen Benutzers.
– Er ist Eigentümer der Tabelle, der Datenbank bzw. der Storage Group.
– Er besitzt die GRANT-Berechtigung zur Weitergabe der Privilegien.

Welcher Berechtigungsschlüssel Eigentümer ist, ist in den Views SCHEMATA, SPACES und STOGROUPS abgelegt.
Ob der Berechtigungsschlüssel die GRANT-Berechtigung für ein Privileg besitzt, erfahren Sie aus den Views TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES und CATALOG_PRIVILEGES (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Nur der Berechtigungsschlüssel, der die Privilegien vergeben hat, der sog. Grantor, kann die Privilegien wieder entziehen.

Die GRANT-Anweisung hat zwei Formate, ein Format für die Vergabe von Tabellen- und Spalten-Privilegien, das andere für die Vergabe von Sonder-Privilegien.

GRANT-Format für Tabellen- und Spalten-Privilegien

```
GRANT     { ALL PRIVILEGES |
              { SELECT |
                DELETE |
                INSERT |
                UPDATE [(spalte,...)] |
                REFERENCES [(spalte,...)]
              }, ...
          }

          ON [ TABLE ] tabelle

          TO { PUBLIC | berechtigungsschlüssel }, ...

          [ WITH GRANT OPTION ]
```

ALL PRIVILEGES
Alle Tabellen- und Spalten-Privilegien werden vergeben, die der aktuelle Berechtigungsschlüssel vergeben darf. ALL PRIVILEGES umfasst die Privilegien SELECT, DELETE, INSERT, UPDATE und REFERENCES.

SELECT | DELETE ...
Die Tabellen- und Spalten-Privilegien werden einzeln vergeben. Sie können mehrere Privilegien angeben. Folgende Angaben sind möglich:

SELECT
Privileg, das das Lesen von Sätzen der Tabelle erlaubt.

DELETE
Privileg, das das Löschen von Sätzen der Tabelle erlaubt.

INSERT
Privileg, das das Einfügen von Sätzen in die Tabelle erlaubt.

UPDATE [(*spalte*,...)]
Privileg, das das Ändern von Sätzen der Tabelle erlaubt.

Das Ändern kann auf die angegebenen Spalten eingeschränkt werden. *spalte* muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:
Es ist das Ändern von allen Spalten der Tabelle erlaubt. Auch später hinzugefügte Spalten dürfen geändert werden.

REFERENCES [(*spalte*,...)]
Privileg, das die Definition von Referenzbedingungen erlaubt, die sich auf die Tabelle beziehen.
Die Referenzierung kann auf die angegebenen Spalten eingeschränkt werden. *spalte* muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:
Es ist das Referenzieren von allen Spalten der Tabelle erlaubt. Auch später hinzugefügte Spalten dürfen referenziert werden.

ON [TABLE] *tabelle*
Name der Tabelle, für die Sie Privilegien vergeben wollen.

Wenn Sie die GRANT-Anweisung innerhalb einer CREATE SCHEMA-Anweisung verwenden, dürfen Sie den Tabellennamen nur mit den Datenbank- und Schemanamen aus der CREATE SCHEMA-Anweisung qualifizieren.

Die Tabelle kann eine Basistabelle oder ein View sein. Für einen nicht änderbaren View können Sie nur das Privileg SELECT vergeben.

TO PUBLIC
Die Privilegien werden der Allgemeinheit verliehen. Jeder Berechtigungsschlüssel besitzt zusätzlich zu seinen eigenen die Privilegien, die an PUBLIC verliehen wurden. Auch später hinzugefügte Berechtigungsschlüssel besitzen diese Privilegien.

TO *berechtigungsschlüssel*
Die Privilegien gelten für den Berechtigungsschlüssel *berechtigungsschlüssel*. Sie können mehrere Berechtigungsschlüssel angeben.

WITH GRANT OPTION
Die angegebenen Berechtigungsschlüssel erhalten zusätzlich zu den Privilegien die GRANT-Berechtigung, d.h. die Berechtigungsschlüssel sind berechtigt, die erhaltenen Privilegien an andere Berechtigungsschlüssel weiterzugeben. Die Klausel WITH GRANT OPTION dürfen Sie nicht in Verbindung mit PUBLIC verwenden.

WITH GRANT OPTION nicht angegeben:
Die angegebenen Berechtigungsschlüssel können die verliehenen Privilegien nicht weitergeben.

GRANT-Format für Sonder-Privilegien

```
GRANT     { ALL SPECIAL PRIVILEGES |
            { CREATE USER |
              CREATE SCHEMA |
              CREATE STOGROUP |
              UTILITY |
              USAGE
            }, ...
          }

          ON { CATALOG catalog | STOGROUP stogroup }

          TO { PUBLIC | berechtigungsschlüssel }, ...

          [ WITH GRANT OPTION ]
```

ALL SPECIAL PRIVILEGES
Alle Sonder-Privilegien werden vergeben, die der aktuelle Berechtigungsschlüssel vergeben darf. ALL SPECIAL PRIVILEGES umfasst die Sonder-Privilegien CREATE USER, CREATE SCHEMA, CREATE STOGROUP, UTILITY und USAGE.

CREATE USER | CREATE SCHEMA | CREATE STOGROUP | UTILITY | USAGE

Die Sonder-Privilegien werden einzeln vergeben. Sie können mehrere Sonder-Privilegien angeben. Folgende Sonder-Privilegien sind möglich:

CREATE USER

Sonder-Privileg, das das Definieren und Löschen von Berechtigungsschlüsseln erlaubt. Sie dürfen das Privileg CREATE USER nur für eine Datenbank vergeben.

CREATE SCHEMA

Sonder-Privileg, das das Definieren von Datenbank-Schemata erlaubt.
Sie dürfen das Privileg CREATE SCHEMA nur für eine Datenbank vergeben.

CREATE STOGROUP

Sonder-Privileg, das das Definieren von Storage Groups erlaubt.
Sie dürfen das Privileg CREATE STOGROUP nur für eine Datenbank vergeben.

UTILITY

Sonder-Privileg, das die Verwendung von Utility-Anweisungen erlaubt.
Sie dürfen das Privileg UTILITY nur für eine Datenbank vergeben.

USAGE

Sonder-Privileg, das die Verwendung der Storage Group erlaubt.
Sie dürfen das Privileg USAGE nur für eine Storage Group vergeben.

ON CATALOG *catalog*

Name der Datenbank, für die Sie Sonder-Privilegien vergeben wollen.

ON STOGROUP *stogroup*

Name der Storage Group, für die Sie das Privileg USAGE vergeben wollen. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

TO PUBLIC

Die Privilegien werden der Allgemeinheit verliehen. Jeder Berechtigungsschlüssel besitzt zusätzlich zu seinen eigenen die Privilegien, die an PUBLIC verliehen wurden. Auch später hinzugefügte Berechtigungsschlüssel besitzen diese Privilegien.

TO *berechtigungsschlüssel*

Die Privilegien gelten für den Berechtigungsschlüssel *berechtigungsschlüssel*. Sie können mehrere Berechtigungsschlüssel angeben.

WITH GRANT OPTION

Die angegebenen Berechtigungsschlüssel erhalten zusätzlich zu den Privilegien die GRANT-Berechtigung, d.h. die Berechtigungsschlüssel sind berechtigt, die erhaltenen Privilegien an andere Berechtigungsschlüssel weiterzugeben. Die Klausel WITH GRANT OPTION dürfen Sie nicht in Verbindung mit PUBLIC verwenden.

WITH GRANT OPTION nicht angegeben:
Die angegebenen Berechtigungsschlüssel können die verliehenen Privilegien nicht weitergeben.

Beispiel

Die erste GRANT-Anweisung vergibt mehrere Tabellen-Privilegien, die zweite vergibt das Sonderprivileg CREATE SCHEMA an einen bereits bestehenden Berechtigungsschlüssel.

```
GRANT SELECT,INSERT,UPDATE ON TABLE telefonliste TO berta

GRANT CREATE SCHEMA ON CATALOG meinedb TO hugo
```

Siehe auch

REVOKE, CREATE SCHEMA

PRAGMA - Pragmaklauseln vereinbaren

Während Pragmas in ESQL/COBOL und im Utility-Monitor als spezielle SQL-Kommentare zusammen mit der zu beeinflussenden SQL-Anweisung eingegeben werden, werden sie in DRIVE mit einer eigenen DRIVE-SQL-Anweisung, der PRAGMA-Anweisung, vereinbart.

Die vereinbarten Klauseln werden von DRIVE in Pragmas für SESAM/SQL umgesetzt, d.h. spezielle SQL-Kommentare, mit denen die Ausführung von SQL-Anweisungen beeinflusst oder überwacht werden kann.

Einsatzmöglichkeiten und Nutzen

Mit Pragmas kann der SESAM V2-Anwender:

– Den sog. „Schubmodus“ aktivieren, d.h. vorgeben, wieviele Sätze einer (statischen, dynamischen oder variablen) Cursortabelle bei einer FETCH-Anweisung maximal „im voraus“ gelesen werden sollen, um die Ausführung nachfolgender FETCH-Anweisungen erheblich zu beschleunigen.

Diese Funktionalität wird nur dann wirksam, wenn die SESAM/SQL-Version 2.1 (oder eine höhere Version) eingesetzt wird. Die SESAM/SQL-Version 2.0 unterstützt die „Schubmodus“-Funktionalität nicht, so dass die Verwendung der „Schubmodus“-Pragmaklausel PREFETCH in DRIVE zu einem Fehler führt (vgl. Abschnitt „Fehlerbehandlung bei Pragmas“ auf Seite 153).

– Für New-Style-DML-Anweisungen (SELECT bzw. Cursorverarbeitung, INSERT, UPDATE, DELETE) eine lesbare Darstellung des internen Zugriffsplans des SQL-Optimizers auf Datei ausgeben.

– Die Ausführungsvorschrift (SQL-Zugriffsplan) zur Abarbeitung einer New-Style-DML-Anweisung beeinflussen.

– Die verwendete Join-Methode (Sort Merge Join oder Nested Loop Join) auswählen.

– Den Sperrmodus für SQL-DML-Anweisungen einstellen.

– Den Isolationslevel für Datenbankzugriffe einer New-Style-DML-Anweisung unabhängig vom Isolationslevel der Transaktion festlegen, in der die Anweisung ausgeführt wird.

– Einer Oldest-Style-Tabelle (Nur-CALL-DML-Tabelle) neue Oldest-Style-Spalten hinzufügen (vgl. Utility-Anweisung MIGRATE und DDL-Anweisung ALTER TABLE).

– Bearbeitung von Basistabellen im Zustand „check pending“ (vgl. Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7]).

Eigenschaften der PRAGMA-Anweisung

Eine PRAGMA-Anweisung ist eine Programmanweisung, die zum Übersetzungszeitpunkt ausgewertet wird. Pragmas können statisch und dynamisch definiert werden:

– Eine statische PRAGMA-Anweisung kann überall zwischen PROCEDURE und END PROCEDURE stehen und wirkt nur auf die textuell nächste statische SQL-Anweisung. Sie wird zum Zeitpunkt der expliziten Sourceübersetzung (COMPILE-Anweisung) oder der impliziten Sourceübersetzung ausgewertet (DO- oder CALL-Anweisung: Vorstufe der Prozedurausführung).

– Eine dynamische PRAGMA-Anweisung kann überall stehen, wo EXECUTE erlaubt ist. Sie wirkt nur auf die chronologisch nächste dynamische SQL-Anweisung und wird bei der impliziten Anweisungsübersetzung ausgewertet (EXECUTE-Anweisung: Generierung und Übersetzung als Vorstufe der Anweisungsausführung).

Eine PRAGMA-Anweisung ist nicht ausführbar und leitet daher keine Transaktion ein.

Die DRIVE-Wirkung besteht darin, dass diese SQL-Anweisung mit dem speziellen Kommentar

"--%PRAGMA { pragma_klausel },..."

präfigiert wird. Die DRIVE-Wirkung der PRAGMA-Anweisung entspricht daher einer Umsetzung der Pragmaklauseln in die SESAM-konforme Verwendung wie in ESQL/COBOL-Programmen und im Utility-Monitor.

Ob jedoch die DRIVE-Wirkung auch zu einer SESAM-Wirkung führt, hängt davon ab, ob die jeweiligen Pragmaklauseln auf die SQL-Anweisung wirken:

– PREFETCH zeigt nur bei DECLARE-Anweisungen und indirekt bei den zugehörigen FETCH-Anweisungen eine SESAM-Wirkung.

– EXPLAIN INTO, IGNORE INDEX, OPTIMIZATION LEVEL, SIMPLIFICATION und ISOLATION LEVEL zeigen nur bei den Anweisungen DECLARE, SELECT, INSERT, UPDATE und DELETE eine SESAM-Wirkung.

– JOIN hat nur bei der SELECT-Anweisung eine SESAM-Wirkung.

– LOCK MODE hat nur bei DML-Anweisungen eine SESAM-Wirkung.

– UTILITY MODE zeigt nur bei der Anweisung ALTER TABLE eine SESAM-Wirkung.

Hat eine Pragmaklausel keine SESAM-Wirkung, so tritt in DRIVE beim Übersetzen (statisches PRAGMA) bzw. bei der Ausführung (dynamisches PRAGMA) der zugeordneten SQL-Anweisung ein Fehler mit &WARNING = 'SQL WARNING' und &SQL_CLASS = '01' auf, siehe auch Seite 153.

PRAGMA-Klauseln

```
PRAGMA '{ CHECK { ON | OFF } |
          DATA TYPE OLDEST |
          EXPLAIN INTO dateiname |
          IGNORE INDEX index |
          ISOLATION LEVEL { READ UNCOMMITTED |
                            READ COMMITTED |
                            REPEATABLE READ |
                            SERIALIZABLE
                          } |
          JOIN [ { SORT MERGE | NESTED LOOP } ] |
          LOCK MODE EXCLUSIVE |
          OPTIMIZATION LEVEL level |
          PREFETCH n |
          SIMPLIFICATION { ON | OFF } |
          UTILITY MODE { ON | OFF } |
       }, ...'
```

Pragmaklausel CHECK

Die Pragmaklausel CHECK hat folgende Form:

```
CHECK { ON | OFF }
```

Voreinstellung ist ON. Mit der Klausel 'CHECK OFF' kann unter bestimmten Voraussetzungen mit DML-Anweisungen auf Basistabellen im Zustand „check pending“ zugegriffen werden (vgl. Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7]).

Pragmaklausel DATA TYPE

DATA TYPE legt fest, dass eine Spalte im Attributformat für Nur-CALL-DML-Tabellen angelegt wird.

Die Klausel hat nur eine SESAM-Wirkung, wenn sie bei der Anweisung ALTER TABLE ... ADD COLUMN ... angegeben ist und die Tabelle eine Nur-CALL-DML-Tabelle ist.

```
DATA TYPE OLDEST
```

Pragmaklausel EXPLAIN

EXPLAIN dient dazu, den vom Optimizer gewählten Zugriffsplan auszugeben. Sie können dieses Pragma nur verwenden, wenn der aktuelle Berechtigungsschlüssel das Sonder-Privileg UTILITY besitzt (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Die Klausel hat nur in folgenden SQL-Anweisungen eine SESAM-Wirkung:
– DECLARE
– DELETE
– INSERT
– SELECT
– UPDATE

Bei einer statisch formulierten Anweisung hat das Pragma nur dann eine Wirkung, wenn Sie das Programm mit Datenbankkontakt vorübersetzen. Dies ist in DRIVE immer der Fall.

```
EXPLAIN INTO datei
```

---

datei

Name der SAM-Datei, in die die Erklärung ausgegeben wird. Wenn die Datei bereits existiert, wird die Erklärung angehängt.

Wenn *datei* eine BS2000-Kennung angibt, wird diese Kennung verwendet, ansonsten die Kennung des Data Base Handlers für die mit der SQL-Anweisung angesprochene Datenbank. In beiden Fällen muss der Data Base Handler Schreibrechte für die Datei besitzen.

Für *datei* geben Sie ein alphanumerisches Literal an.

Bei dynamisch formulierten Anweisungen wird die Erklärung zum Ausführungszeitpunkt der EXECUTE-Anweisung ausgegeben, bei statisch formulierten Anweisungen wird die Erklärung zum Zeitpunkt der Vorübersetzung ausgegeben.

Die Erklärung besteht aus der SQL-Anweisung und einer aufbereiteten Darstellung des Zugriffsplans. Die Darstellung von Zugriffsplänen ist im Handbuch „SESAM/SQL-Server - Performance“ [12] beschrieben.

Die Datei können Sie mit SHOW-FILE anzeigen. Um die Datei mit EDT lesen zu können, müssen Sie folgendes Kommando eingeben:

```
SET-FILE-LINK LINK-NAME=EDTSAM,FILE-NAME=datei,...,BUFFER-LENGTH=(STD,2),...
```

Ab der EDT-Version 16.5A können Sie auch eingeben:

```
@OPEN F=datei,TYPE=CATALOG
```
bzw.
```
@OP F=datei,T=C
```

Beispiel

```
SET &explainfile = '$YDRI2O.EXPLAINFILE';

/* Die Datei $YDRI2O.EXPLAINFILE muss mit USER-ACC = ALL-USERS */
/* katalogisiert sein */

EXECUTE 'SET SESSION AUTHORIZATION ''"DRI-USER1" ''';

/* Der Berechtigungsschlüssel DRI-USER1 muss das Sonder-Privileg UTILITY*/
/* für die voreingestellte Datenbank besitzen */

EXECUTE 'PRAGMA ''EXPLAIN INTO ''''|| &explainfile || ''''''';

DISPLAY FORM 'Ausgabe des SQL-Zugriffsplans in Datei ' || &explainfile;

EXECUTE 'DECLARE C1 CURSOR FOR cursorbeschreibung';
```

Pragmaklausel IGNORE INDEX

Der spezifizierte Index wird bei der Festlegung der Join-Reihenfolge, des Join-Algorithmus und bei der Auswahl des optimalen Zugriffspfads (auf Basisrelationen) ignoriert.

```
IGNORE INDEX index
```

Pragmaklausel ISOLATION LEVEL

ISOLATION LEVEL legt den Isolationslevel für die Datenbankzugriffe einer SQL- Anweisung fest.

Die Klausel hat nur in folgenden SQL-Anweisungen eine SESAM-Wirkung:
– DECLARE
– DELETE
– INSERT
– SELECT
– UPDATE

```
ISOLATION LEVEL
              { READ UNCOMMITTED |
                 READ COMMITTED |
                 REPEATABLE READ |
                 SERIALIZABLE }
```

Die Isolationslevel sind bei der Anweisung SET TRANSACTION beschrieben.

Wenn Sie das Pragma ISOLATION LEVEL angegeben haben, erfolgt jeder Datenbankzugriff, der mit dieser Anweisung zusammenhängt, unter diesem Isolationslevel.

Wenn Sie einen geringeren Isolationslevel angeben, als für die Transaktion festgelegt ist, ist der festgelegte Isolationslevel der Transaktion nicht mehr garantiert!

Pragmaklausel JOIN

Durch Angabe des Pragmas wird die verwendete Join-Methode (Sort Merge Join oder Nested Loop Join) ausgewählt.

```
JOIN [SORT MERGE | NESTED LOOP]
```

Wird durch das Pragma OPTIMIZATION LEVEL die Anzahl der betrachteten Planalternativen so eingeschränkt, dass kein Nested Loop Join mehr betrachtet wird (OPTIMIZATION LEVEL<6), so kann auch durch das Pragma JOIN kein Nested Loop Join erzwungen werden. Bei Mehrfachjoins wird das Pragma auch bei den anfallenden Zwischenjoins berücksichtigt.

Ist das Pragma JOIN nicht angegeben, so wählt der Optimizer die zu verwendende Joinmethode aus.

Pragmaklausel LOCK MODE

Das Pragma LOCK MODE stellt den Sperrmodus ein. Es wirkt nur in SQL-DML-Anweisungen.

```
LOCK MODE EXCLUSIVE
```

Ist LOCK MODE EXCLUSIVE angegeben, so erfolgt jeder Zugriff auf die Datenbank, der direkt oder indirekt mit dieser SQL-Anweisung zusammenhängt, mit Exklusiv-Sperren.

Ohne Pragma LOCK MODE wird der Sperrmodus vom System festgelegt.

Pragmaklausel OPTIMIZATION LEVEL

Die Option *n* steuert die Anzahl der Planalternativen, die im Verlauf der Zugriffspfadauswahl erzeugt und bewertet werden.

```
OPTIMIZATION LEVEL n
```

Es werden zunächst alle Planvarianten untersucht und mittels Heuristiken nur die aussichtsreichsten Varianten, die im allgemeinen eine Verbesserung der Auswertungskosten versprechen, ausgewählt. Die Anzahl der Planvarianten, die weiterverfolgt werden, hängt von *n* ab. Der Wert *n* kann zwischen 1 und 10 gewählt werden; die Voreinstellung ist 9. Ab einem Wert $n \leq 5$ wird in jedem Optimierungsschritt nur noch eine Planvariante untersucht.

Im einzelnen werden folgende Stufen unterschieden:

– $n \leq 9$

  Es werden verschiedene Join-Reihenfolgen betrachtet.

– $n \leq 8$

  Beim Nested-Loop-Join wird untersucht, ob die Vertauschung der beiden Join-Partner vorteilhaft ist.

– $n \leq 7$

  Durchführung der Sort-Minimierung.

– $n \leq 6$

  Bei der Join-Optimierung wird neben dem Sort-Merge auch der Nested-Loop-Join betrachtet.

Bei der Zugriffsauswahl werden alle Möglichkeiten betrachtet, eine geforderte Sortierung zu erreichen (physische Sortierung im DBH-Kern, Sort durch Index-Scan).

– $n \leq 5$

  Durchführung der Unterabfrage-Optimierung und Abspeicherung von mehrfach benötigten Zwischenergebnisrelationen.

– $n \leq 4$

  Durchführung der Range-Konstruktion; d.h. mehrere atomare Prädikate auf derselben Spalte werden zu einem Indexzugriff zusammengefasst.

Pragmaklausel PREFETCH

PREFETCH steuert den Schubmodus der SQL-Anweisung FETCH (Cursor positionieren). Der Schubmodus beschleunigt die Ausführung der Anweisung FETCH. Er ist nur wirksam, wenn FETCH den Cursor auf den nächsten Satz der Cursortabelle positioniert (FETCH NEXT).

Über das Pragma PREFETCH können Sie den Schubmodus einschalten und einen Blokkungsfaktor (n) festlegen. Bei Ausführung der ersten Anweisung FETCH NEXT... werden dann die Spaltenwerte des aktuellen Satzes gelesen, die folgenden n-1 Sätze der zugehörigen Cursortabelle werden in einem Zwischenpuffer gespeichert. Bei Ausführung der nachfolgenden n-1 Anweisungen FETCH NEXT..., die denselben Cursor bezeichnen, kann dann direkt, ohne DBH-Kontakt, auf den nächsten Satz zugegriffen werden.

Enthält die Cursorbeschreibung der DECLARE-Anweisung für statische oder dynamische Cursor eine FOR UPDATE-Klausel, wird das Pragma PREFETCH ignoriert (d.h. es hat keine SESAM-Wirkung), der Schubmodus wird nicht eingeschaltet.

```
PREFETCH n
```

---

n

Blockungsfaktor. Den Blockungsfaktor müssen Sie als vorzeichenlose Ganzzahl angeben (Datentyp SMALLINT).

Hat der Blockungsfaktor (*n*) einen Wert > 0, werden maximal n-1 Sätze der spezifizierten Cursortabelle in einem Zwischenpuffer gespeichert. Hat der Blockungsfaktor den Wert 0, bleibt das Pragma PREFETCH ohne Wirkung. D.h. Sie können die Wirkung des Pragmas und damit den Schubmodus ein- bzw. ausschalten, indem Sie für *n* einen Wert > 0 bzw. den Wert 0 angeben.

Bei eingeschaltetem Schubmodus gelten folgende Einschränkungen:

In derselben Übersetzungseinheit ist für den PREFETCH-Cursor nur noch die Anweisung FETCH NEXT erlaubt. Folgende SQL-Anweisungen sind nicht mehr ausführbar:
- UPDATE ... WHERE CURRENT OF *cursor*
- DELETE ... WHERE CURRENT OF *cursor*
- STORE *cursor*
- FETCH *cursor* mit einer anderen Cursorpositionierung als NEXT
- FETCH *cursor* mit einer von der ersten Anweisung FETCH NEXT verschiedenen INTO-Klausel, falls *cursor* statisch ist.

Pragmaklausel UTILITY MODE

Das Pragma UTILITY MODE legt fest, ob für die SQL-Anweisung, in der dieses Pragma angegeben ist, die Transaktionssicherung wirksam sein soll. Die Transaktionssicherung ermöglicht, dass im Fehlerfall eine Transaktion auf einen konsistenten Zustand zurückgesetzt wird.

Das Pragma UTILITY MODE ist nur in der SQL-Anweisung ALTER TABLE wirksam

Es wirkt nur, wenn die ALTER TABLE-Anweisung Spalten einer Basistabelle hinzufügt, ändert oder löscht. Bei einer ALTER TABLE-Anweisung, die Integritätsbedingungen zufügt oder löscht, bleibt das Pragma UTILITY MODE ohne Wirkung.

```
UTILITY MODE { ON | OFF }
```

ON
: Bei der Ausführung der SQL-Anweisung wird die Transaktionssicherung ausgeschaltet. Die zugehörige ALTER TABLE-Anweisung eröffnet keine Transaktion. Sicherungsdaten für die ALTER TABLE-Anweisung werden nicht gespeichert. Führt ein Fehler zum Abbruch der Anweisung, ist das Rücksetzen auf einen konsistenten Zustand nicht möglich. Der Space, auf dem die zu ändernde Basistabelle liegt, ist im Fehlerfall defekt und muss mit Hilfe der Utility-Anweisung RECOVER repariert werden (siehe Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7]).

OFF
: Das Pragma hat keine Auswirkungen. Die Transaktionssicherung bleibt eingeschaltet.

Eine ALTER TABLE-Anweisung, für die das Pragma UTILITY MODE ON eingeschaltet und wirksam ist, wird in folgenden Fällen mit Fehlermeldung abgebrochen:

– wenn eine Transaktion aktiv ist

– wenn es sich um eine ALTER TABLE-Anweisung zum Löschen einer Spalte handelt, nämlich mit DROP COLUMN *spalte* CASCADE.

Wenn Sie für eine ALTER TABLE-Anweisung das Pragma UTILITY MODE nicht angeben, gilt als Standardeinstellung UTILITY MODE OFF.

Das Pragma UTILITY MODE ON hat zur Folge, dass nach einem Fehler oder Consistency Check der Space, auf dem die zu ändernde Basistabelle liegt, defekt ist. Um Datenverlust zu vermeiden, sollten Sie vor dem Absetzen der ALTER TABLE-Anweisung den Space sichern. Sie brauchen die Sicherung, wenn Sie einen defekten Space mit Hilfe der Utility-Anweisung RECOVER reparieren wollen.

Fehlerbehandlung bei Pragmas

Beim Umgang mit Pragmas können folgende Fehler gemacht werden:

– Die PRAGMA-Anweisung kann syntaktisch inkorrekt angegeben werden.

  In diesem Fall meldet DRIVE einen Syntaxfehler beim Übersetzen der PRAGMA-Anweisung. Bei einer dynamischen PRAGMA-Anweisung ist dann &ERROR mit 'SYNTAX ERROR' belegt.

Andernfalls wird die PRAGMA-Anweisung bei der nächsten SQL-Anweisung in der Weise wirksam (sog. DRIVE-Wirkung), dass der Inhalt des spezifizierten Literals mit dem Präfix %PRAGMA als SQL-Kommentar an SESAM weitergereicht wird. Hierbei können nun Fehler auftreten, die SESAM feststellt, so dass keine SESAM-Wirkung eintritt:

– Der Inhalt des in der PRAGMA-Anweisung zu spezifizierenden Literals ist syntaktisch inkorrekt.

  In diesem Fall meldet SESAM an DRIVE einen SQLSTATE der Klasse 01 (Warnung) und vorübersetzt (statisches Pragma) bzw. präpariert (dynamisches Pragma) die SQL-Anweisung ohne Pragmawirkung.

– Der Inhalt des in der PRAGMA-Anweisung zu spezifizierenden Literals ist zwar syntaktisch korrekt, aber die Pragmaklauseln werden einer SQL-Anweisung zugeordnet, bei der sie keine Wirkung haben (vgl. Beschreibung der einzelnen Klauseln).

  Dieser Fall tritt z.B. ein, wenn die Pragmaklausel PREFETCH mit der SESAM/SQL-Version 2.0 verwendet wird.

  SESAM meldet dann an DRIVE einen SQLSTATE der Klasse 01 (Warnung) und vorübersetzt (statisches Pragma) bzw. präpariert (dynamisches Pragma) die SQL-Anweisung ohne Pragmawirkung.

– Der Inhalt des in der PRAGMA-Anweisung zu spezifizierenden Literals ist syntaktisch korrekt, und die Pragmaklauseln werden einer SQL-Anweisung zugeordnet, bei der sie eigentlich SESAM-Wirkung haben, aber die SESAM-Wirkung ist aus anderen Gründen gehemmt (vgl. Beschreibung der einzelnen Klauseln).

  Dieser Fall tritt z.B. ein, wenn in der Pragmaklausel EXPLAIN eine BS2000-Datei spezifiziert wird, die gar nicht oder nicht SHAREABLE katalogisiert ist.

  In diesem Fall meldet SESAM an DRIVE ebenfalls einen SQLSTATE der Klasse 01 (Warnung) und vorübersetzt bzw. präpariert die SQL-Anweisung ohne Pragmawirkung.

DRIVE setzt alle Warnungen von SESAM (SQLSTATE-Klasse = 01) in folgende DRIVE-Warnung um:

```
&WARNING='SQL WARNING'
```

Ist für diese Ausnahmebedingung eine WHENEVER-Aktion definiert (siehe Seite 164), dann bestimmt diese die Reaktion auf die SQL-Warnung, z.B. Transaktion fortsetzen, zurücksetzen oder schließen.

Ist für diese Ausnahmebedingung keine WHENEVER-Aktion definiert, dann wird die Warnung ignoriert, da von DRIVE für Warnungen der Fehlerausgang 'CONTINUE' eingestellt ist. Das bedeutet, dass die SQL-Anweisung ohne Pragma-Wirkung ausgeführt wird.

REORG STATISTICS

REORG STATISTICS erzeugt die globale Statistik über die Werteverteilung der Spalten eines Index neu. Diese Statistik wird zur Optimierung von Zugriffen mit Suchbedingungen auf Tabellen herangezogen und sollte daher nach umfangreichen Datenänderungen wieder aktualisiert werden.

Der aktuelle Berechtigungsschlüssel muss entweder Eigentümer des Schemas sein, zu dem der Index gehört oder das Sonder-Privileg UTILITY für die Datenbank besitzen, zu dem der Index gehört.

```
REORG STATISTICS FOR INDEX index
```

---

index
Name des Index, dessen Statistik neu erzeugt werden soll.

Der einfache Name des Index kann durch einen Datenbank- und Schemanamen qualifiziert werden.

Siehe auch

CREATE INDEX

REVOKE - Privilegien entziehen

REVOKE entzieht Berechtigungsschlüsseln Tabellen- und Spalten-Privilegien oder Sonder-Privilegien. Falls noch temporäre Views[1] des Berechtigungsschlüssels auf der Tabelle beruhen, werden diese gelöscht.

Privilegien können einem Berechtigungsschlüssel nur von dem Berechtigungsschlüssel entzogen werden, der das Privileg vergeben hat, dem sog. Grantor (siehe Abschnitt „GRANT - Privilegien vergeben“ auf Seite 139).

Welche Privilegien welchen Berechtigungsschlüsseln zugeordnet sind, erfahren Sie in den Tabellen TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGE und CATALOG_PRIVILEGES des INFORMATION_SCHEMA (siehe Kapitel „Informationsschemata“ im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 1“ [6]).

Die REVOKE-Anweisung hat zwei Formate, ein Format für Tabellen- und Spalten-Privilegien, das andere für Sonder-Privilegien.

REVOKE-Format für Tabellen- und Spalten-Privilegien

```
REVOKE { ALL PRIVILEGES |
          { SELECT |
            DELETE |
            INSERT |
            UPDATE [(spalte,...)] |
            REFERENCES [(spalte,...)]
          }, ...
       }

       ON [ TABLE ] tabelle

       FROM { PUBLIC | berechtigungsschlüssel }, ...

       { CASCADE | RESTRICT }
```

ALL PRIVILEGES
: Alle Tabellen-Privilegien werden entzogen, die der aktuelle Berechtigungsschlüssel entziehen darf. ALL PRIVILEGES umfasst die Privilegien SELECT, DELETE, INSERT, UPDATE und REFERENCES.

[1] Temporäre Views werden ab SESAM/SQL V3.0 nicht mehr unterstützt

SELECT | DELETE | INSERT | UPDATE [(*spalte*,...)] | REFERENCES [(*spalte*,...)]
Die Tabellen- und Spalten-Privilegien werden einzeln entzogen. Sie können ein oder mehrere der folgenden Privilegien angeben:

SELECT
Privileg, das das Lesen von Sätzen der Tabelle erlaubt.

DELETE
Privileg, das das Löschen von Sätzen der Tabelle erlaubt.

INSERT
Privileg, das das Einfügen von Sätzen in die Tabelle erlaubt.

UPDATE [(*spalte*,...)]
Privileg, das das Ändern von Sätzen der Tabelle erlaubt.

Der Entzug des Privilegs kann auf die angegebenen Spalten beschränkt werden. *spalte* muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:
Das Privileg zum Ändern aller Spalten der Tabelle wird entzogen.

REFERENCES [(*spalte*,...)]
Privileg, das die Definition von Referenzbedingungen erlaubt, die sich auf die Tabelle beziehen.

Der Entzug des Privilegs kann auf die angegebenen Spalten beschränkt werden. *spalte* muss ein Spaltenname der angegebenen Tabelle sein. Sie können mehrere Spalten angeben.

(*spalte*,...) nicht angegeben:
Das Privileg zum Referenzieren aller Spalten der Tabelle wird entzogen.

ON [TABLE] *tabelle*
Name der Tabelle, für die Sie Privilegien entziehen wollen.

Die Tabelle kann eine Basistabelle oder ein View sein. Für einen nicht änderbaren View können Sie nur das Privileg SELECT entziehen.

FROM PUBLIC
Die Privilegien werden der Allgemeinheit entzogen. Individuelle Privilegien von Berechtigungsschlüsseln werden dadurch nicht berührt.

FROM *berechtigungsschlüssel*
Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, werden alle weitergegebenen Privilegien implizit gelöscht.
- Views, die auf Grund des angegebenen Privilegs definiert wurden, werden gelöscht.
- Referenzbedingungen, die auf Grund des angegebenen Privilegs definiert wurden, werden gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.
- Ein Privileg, auf Grund dessen ein View oder eine Referenzbedingung definiert wurde, kann nicht entzogen werden, wenn der View bzw. die Referenzbedingung noch existiert.

REVOKE-Format für Sonder-Privilegien

```
REVOKE { ALL SPECIAL PRIVILEGES |
          { CREATE USER |
            CREATE SCHEMA |
            CREATE STOGROUP |
            UTILITY |
            USAGE
          }, ...
       }

       ON { CATALOG catalog | STOGROUP stogroup }

       FROM {PUBLIC | berechtigungsschlüssel }, ...

       { CASCADE | RESTRICT }
```

ALL SPECIAL PRIVILEGES
Alle Sonder-Privilegien werden entzogen, die der aktuelle Berechtigungsschlüssel entziehen darf. ALL SPECIAL PRIVILEGES umfasst die Privilegien CREATE USER, CREATE SCHEMA, CREATE STOGROUP, UTILITY und USAGE.

CREATE USER | CREATE SCHEMA | CREATE STOGROUP | UTILITY | USAGE
Die Sonder-Privilegien werden einzeln entzogen. Sie können ein oder mehrere der folgenden Sonder-Privilegien angeben:

CREATE USER
Sonder-Privileg, das das Definieren von Berechtigungsschlüsseln erlaubt. Sie dürfen das Privileg CREATE USER nur für eine Datenbank entziehen.

CREATE SCHEMA
Sonder-Privileg, das das Definieren von Datenbank-Schemata erlaubt. Sie dürfen das Privileg CREATE SCHEMA nur für eine Datenbank entziehen.

CREATE STOGROUP
Sonder-Privileg, das das Definieren von Storage Groups erlaubt. Sie dürfen das Privileg CREATE STOGROUP nur für eine Datenbank entziehen.

UTILITY
Sonder-Privileg, das die Verwendung von Utility-Anweisungen erlaubt. Sie dürfen das Privileg UTILITY nur für eine Datenbank entziehen.

USAGE
Sonder-Privileg, das die Verwendung der Storage Group erlaubt. Sie dürfen das Privileg USAGE nur für eine Storage Group entziehen.

ON CATALOG *catalog*
Name der Datenbank, für die Sie Sonder-Privilegien entziehen wollen.

ON STOGROUP *stogroup*
Name der Storage Group, für die Sie das Privileg USAGE entziehen wollen. Der einfache Name der Storage Group kann durch einen Datenbanknamen qualifiziert werden.

FROM PUBLIC
Die Privilegien werden der Allgemeinheit entzogen. Individuelle Privilegien von Berechtigungsschlüsseln werden dadurch nicht berührt.

FROM *berechtigungsschlüssel*
Die Privilegien werden dem Benutzer mit dem Berechtigungsschlüssel *berechtigungsschlüssel* entzogen. Sie können mehrere Berechtigungsschlüssel angeben.

CASCADE

Der Grantor kann die von ihm vergebenen Privilegien uneingeschränkt entziehen:

- Alle angegebenen Privilegien werden entzogen.
- Wurde ein angegebenes Privileg an andere Berechtigungsschlüssel weitergegeben, werden alle weitergegebenen Privilegien implizit gelöscht.
- Views, die auf Grund des angegebenen Privilegs definiert wurden, werden gelöscht.
- Referenzbedingungen, die auf Grund des angegebenen Privilegs definiert wurden, werden gelöscht.

RESTRICT

Für das Entziehen von Privilegien gelten Einschränkungen:

- Ein Privileg, das an andere Berechtigungsschlüssel weitergegeben wurde, kann nicht entzogen werden, solange ein solches weitergegebenes Privileg existiert.
- Ein Privileg, auf Grund dessen ein View oder eine Referenzbedingung definiert wurde, kann nicht entzogen werden, wenn der View bzw. die Referenzbedingung noch existiert.

Beispiel

Die folgende REVOKE-Anweisung entzieht das UPDATE-Privileg für alle Spalten der Tabelle TELEFONLISTE.

```
REVOKE UPDATE ON TABLE telefonliste FROM berta RESTRICT
```

Siehe auch

GRANT

UTILITY - UTILITY-Anweisung durchreichen

Mit dieser Anweisung werden UTILITY-Anweisungen an SESAM/SQL durchgereicht.

`UTILITY` *utility-anweisung*

Die Utility-Anweisungen werden als DRIVE-String durchgereicht. Der String wird dabei jedoch nicht vollständig geprüft.

Es sind folgende Utility-Anweisungen möglich:

Anweisung	**Bedeutung**
ALTER MEDIA DESCRIPTION	Medientabelle ändern
CHECK CONSTRAINTS	Integritätsbedingungen prüfen
CHECK FORMAL	Tabellen und Indizes formal prüfen
COPY	Sicherungskopie erstellen
CREATE CATALOG	Datenbank (Catalog Space) anlegen
CREATE MEDIA DESCRIPTION	Ersten Medieneintrag für eine datenbankspezifische Dateiart in die Medientabelle aufnehmen
CREATE REPLICATION	Aus Sicherungskopie Replikat erzeugen
DROP MEDIA DESCRIPTION	alle Mediensätze für eine datenbankspezifische Dateiart aus der Medientabelle löschen
LOAD	Anwenderdaten in eine Basistabelle laden
MIGRATE	Datenbanken und Tabellen migrieren
MODIFY	Informationen (Metadaten) über die Sicherungsbestände pflegen
RECOVER	Rücksetzen und Reparieren, Indizes neu aufbauen
REFRESH REPLICATION	Replikat aktualisieren
REORG	Catalog Space und Anwender Spaces reorganisieren
UNLOAD	Anwenderdaten aus einer Basistabelle ausgeben

Näheres zu diesen Anweisungen finden Sie im Handbuch „SESAM/SQL-Server - SQL-Sprachbeschreibung, Teil 2“ [7].

5 DRIVE-Anweisungen

Dieses Kapitel enthält die Änderungen und Ergänzungen zum „DRIVE-Lexikon“ [3]. Die Abschnittsbezeichnungen und Kapitelnummern beziehen sich auf dieses Handbuch.

Übersicht

- WHENEVER-Anweisung:

  Neuer Operand &WARNING, um Warnungen in DRIVE zu behandeln.

- PAGE PRINT-Anweisung:

  Das Schlüsselwort POSITION ist Pflicht.

- DRIVE-Metavariable charprim:

  Neue Stringfunktion SQLMSGSTRING, um den Meldungstext der SQL-Schnittstelle zu ermitteln.

- DRIVE-Anweisungen und -Parameter für FHS-DE:

  Da FHS-DE nicht unterstützt wird, entfallen die Anweisungen ADD BOX, REMOVE BOX und REPLACE BOX sowie der FHS-DE-spezifischen Operanden der Anweisung DISPLAY screenformat.

5.1 WHENEVER - Fehlerausgang definieren

(Zu Kapitel 3, Beschreibung der WHENEVER-Anweisung auf Seite 213)

Diese Anweisung ist gültig

– im TIAM- und UTM-Betrieb
– im Programm-Modus

WHENEVER definiert einen Fehlerausgang, wenn in einem Programm ein Fehler auftritt. WHENEVER muss im Programm im Deklarationsteil hinter den Definitionen von internen Unterprogrammen stehen. Werden für ein Ereignis mehrere WHENEVER angegeben, gilt die letzte Anweisung.

Mit der Anweisung WHENEVER werden die Einträge der Systemvariablen &KFKEY, &ERROR (= &ERROR_STATE.ERROR), &WARNING (= &ERROR_STATE.WARNING) und &DML_STATE (= &ERROR_STATE.DML_STATE) abgefragt und Fehlerausgänge definiert. Die Beschreibung der Systemvariablen und ihrer Einträge finden Sie in „DRIVE-Programmiersprache“ [2].

Wird sowohl für &ERROR als auch für &DML_STATE ein Eintrag abgefragt und treten beide Ereignisse gleichzeitig ein, wird der Fehlerausgang für &ERROR ausgeführt, falls &SQL_CODE > 0, ansonsten wird der Fehlerausgang für &DML_STATE ausgeführt.

Tritt ein Fehler ein, wird das entsprechende Zählfeld hochgezählt.

Wird kein Fehlerausgang definiert, bricht DRIVE das Programm ab.
Ausnahme: Das Programm wird fortgesetzt bei den &ERROR-Einträgen "OK END", "TOO LONG" und "TOO SHORT", bei den &DML_STATE-Einträgen "TABLE END", "DIRTY READ" und "SQL CONV WARNING", sowie bei allen &WARNING-Einträgen.

```
WHENEVER { &KFKEY [ IN ( literal, ... ) ] |
           &ERROR [ IN ( error, ... ) ] |
           &DML_STATE [ IN ( status, ... ) ] |
           &WARNING [ IN ( warning, ...) ]
         }

         { CONTINUE | CALL subprogname | BREAK }
```

&KFKEY IN	Die Bedingung tritt beim Betätigen der Taste *literal* ein. Sie wird nur bei Programmen ohne Fensteroberfläche ausgewertet. Dabei wird von DRIVE die Aktion CONTINUE gesetzt. Die Bedingung wird nur ausgeführt, wenn keine andere Fehlerbedingung auftritt. Bei einer DISPLAY-Anweisung mit Überlauf führt das Betätigen der Taste *literal* zum Abbruch der Ausgabe. Als nächste Anweisung wird entweder CONTINUE, CALL oder BREAK ausgeführt.
literal	Bezeichnung einer Taste (K1, K3 - K14 oder F1 - F20)
&ERROR IN	Der Eintrag von &ERROR kann nach folgenden Anweisungen abgefragt werden: CALL (nicht CALL *subprogname*) CASE CYCLE FOR / WHILE DISPLAY [FORM / LIST / SCREEN] DO ENTER END CYCLE einer CYCLE WHILE- oder CYCLE FOR-Schleife END CYCLE einer CYCLE *cursorname*-Schleife bei Remote-Zugriff END DISPATCH (Nicht abgefragt werden kann &ERROR nach CALL-Anweisungen, die Programme im entfernten System aufrufen.) END IF EXECUTE (nach EXECUTE und nach EXECUTE mit einer der aufgeführten Anweisungen) FILL {FORM / LIST} IF PROCEDURE SEND MESSAGE SET SYSTEM SQL-Anweisungen mit INTO-Klausel Anweisungen für die Dateibearbeitung Remote-Zugriffen auf eine SESAM- oder UDS-Datenbank Wird IN (*error*, ...) nicht angegeben, ist dies gleichbedeutend mit der Angabe aller möglichen Angaben für *error*.
error	Mit *error* wird der Eintrag festgelegt, für den ein Fehlerausgang definiert wird.

Welche Einträge von &ERROR abgefragt werden können, entnehmen Sie der DRIVE-Programmiersprache [2], Abschnitt Systemvariablen.

Für *error* muss ein Literal angegeben werden.

&DML_STATE IN

Der Eintrag von &DML_STATE kann für alle SQL-Anweisungen und nach EXECUTE, das eine SQL-Anweisung ausführt, abgefragt werden. Außerdem nach END CYCLE innerhalb einer "CYCLE cursorname INTO"-Schleife, wenn der Wert von SQLCODE < 0 ist.

Wird IN (*status*, ...) nicht angegeben, ist dies gleichbedeutend mit der Angabe aller möglichen Angaben für *status*.

status

Mit *status* wird der Eintrag festgelegt, für den ein Fehlerausgang definiert wird.

Welche Einträge von &DML_STATE abgefragt werden können, entnehmen Sie der „DRIVE-Programmiersprache“ [2], Abschnitt „Systemvariablen“.

Für *status* muss ein Literal angegeben werden.

Der Meldungstext des SQL-Fehlers kann mit Hilfe der Stringfunktion SQLMSGSTRING ermittelt werden, näheres siehe Seite 169.

&WARNING IN

Mit &WARNING können Warnungen der SQL2-Schnittstelle ausgewertet werden.

Wird IN (*warning*, ...) nicht angegeben, ist dies gleichbedeutend mit der Angabe aller möglichen Angaben für *warning*.

warning

Mit *warning* wird der Eintrag festgelegt, für den ein Fehlerausgang definiert werden soll. Mögliche Werte:

SQL WARNING:
Es liegt eine Warnung der SQL2-Schnittstelle vor. Dieser Wert wird nur gesetzt, wenn kein schwerer Fehler vorliegt, d.h. &ERROR und &DML_STATE besitzen den Wert OK.

Der Meldungstext der Warnung kann mit Hilfe der Stringfunktion SQLMSGSTRING ermittelt werden, näheres siehe Seite 169.

CONTINUE

Bei Eintritt eines definierten Fehlerereignisses wird das Programm fortgesetzt. Die Systemvariable &ERROR_STATE wird dann mit den oben beschriebenen Fehlerinformationen versorgt.

CALL *subprogname*

Bei Eintritt eines definierten Fehlerereignisses wird das interne Unterprogramm *subprogname* aufgerufen. Die Systemvariable &ERROR_STATE wird mit den oben beschriebenen Fehlerinformationen versorgt.

Tritt bei der Abarbeitung des internen Unterprogramms erneut ein Fehler auf, für den ein Fehlerausgang bei WHENEVER definiert wurde, wird das Programm abgebrochen. Die Systemvariable &ERROR_STATE wird in dem internen Unterprogramm nicht verändert.

BREAK — Bei Eintritt eines definierten Fehlerereignisses wird das Programm abgebrochen.

Beispiel

Anweisung	**Ereignis**	**&ERROR=**	**&DML_STATE=**	**&WARNING**
SET &v=&a(&i)	INDEX ERROR	'INDEX ERROR'	unverändert	unverändert
OPEN *cursorname*	SQL ERROR	unverändert	'SQL ERROR'	’OK’
FETCH *cursorname* INTO ...	DIRTY READ	unverändert	'DIRTY READ'	’OK’
SELECT * INTO	SQL CONV WARNING	unverändert	’SQL CONV WARNING’	’OK’
SELECT ...	SQL WARNING	unverändert	’OK’	’SQL WARNING’

5.2 PAGE PRINT - Seitenhintergrundmuster beschreiben

(Zu Kapitel 4, Beschreibung der Report-Anweisung PAGE PRINT auf Seite 254)

Bei der Ausgabeposition muss das Schlüsselwort POSITION angegeben werden.

```
PAGE PRINT {ausdruck POSITION x y [ CM | INCH | UNITS ] ...
           ...
           }
```

Beispiel

```
PAGE PRINT 'Beispiel-Report' POSITION 2 5 CM;
```

5.3 charprim - Stringfunktionen

(Zu Kapitel 5, Beschreibung der Metavariablen *charprim* auf Seite 299)

Die DRIVE-Metavariable *charprim* erhält die neue Funktion SQLMSGSTRING. Mit dieser kann der SESAM-Meldungstext bei SQL-Fehlern oder SQL-Warnungen ermittelt werden.

```
charprim::={ ...|
             SQLMSGSTRING |
             ... }
```

SQLMSGSTRING — Ermittelt den SQL-Fehlertext oder SQL-Warnungstext.

Wenn diese Funktion in einer SET-Anweisung verwendet wird, dann muss als Empfangsvariable eine DRIVE-Variable vom Datentyp CHAR(240) deklariert werden.

Wurde eine SQL-Anweisung ordnungsgemäß ausgeführt, dann wird die Empfangsvariable mit Leerzeichen versorgt.

Beispiel

```
...
FETCH CURSOR INTO &CURSOR.*;
DISPLAY FORM SQLMSGSTRING;
...
```

5.4 Wegfall der DRIVE-Anweisungen und -Operanden für FHS-DE

(Zu Kapitel 3, „DRIVE-Anweisungen“)

Da FHS-DE nicht unterstützt wird, entfallen die zugehörigen DRIVE-Anweisungen und -Operanden. Details entnehmen Sie bitte folgender Tabelle:

Anweisung	**Seite**	**Bemerkung**
ADD BOX	17	Anweisung entfällt
DISPLAY screenformat	95	Operanden CURSOR und MESSAGE entfallen
REMOVE BOX	187	Anweisung entfällt
REPLACE BOX	190	Anweisung entfällt

6 Hinweise zur DRIVE-Programmierung

Dieses Kapitel enthält die Änderungen und Ergänzungen zu „DRIVE-Programmiersprache“ [2]. Die Abschnittsbezeichnungen und Kapitelnummern beziehen sich auf dieses Handbuch, sofern nicht explizit auf ein anderes Handbuch verwiesen wird.

6.1 Datenkonvertierung bei SQL-Anweisungen

(neuer Abschnitt)

6.1.1 Verträglichkeit von Datentypen und Werten

Bei der Verwendung von Werten in Berechnungen, Prädikaten und Zuweisungen, müssen die Datentypen der beteiligten Operanden verträglich sein.

Zwei Datentypen sind verträglich, wenn sie eine der folgenden Bedingungen erfüllen:

– Beide Datentypen sind alphanumerisch (CHARACTER oder CHARACTER VARYING).

– Beide Datentypen sind numerisch (SMALLINT, INTEGER, NUMERIC, DECIMAL, XDEC, REAL, DOUBLE PRECISION oder FLOAT).

– Beide Datentypen sind DATE.

– Beide Datentypen sind TIME.

– Beide Datentypen sind TIMESTAMP.

Ist die Verträglichkeit der Datentypen gegeben, dann wird die Verträglichkeit der Wertebereiche überprüft. Ein Wert ist verträglich zum Wertebereich des Zieldatentyps, wenn er ein Element des Wertebereichs ist.

Anwendung von Datenkonvertierungsregeln

Konvertierungsregeln werden in folgenden Situationen angewandt:

– Bei allen statischen oder dynamischen SQL-Anweisungen mit mindestens einer DRIVE Eingabe- oder Ausgabevariable.

– Bei allen SQL-Anweisungen, die sich auf einen Wert beziehen (Konstante, Variable, Ausdruck, Subquery).

6.1.2 Verstoß gegen Konvertierungsregeln

Ist ein Wert nicht konvertierbar, dann erhält das DRIVE-Programm einen entsprechenden Wert in &SQL_STATE. Die folgende Tabelle erläutert alle Werte von SQLSTATE, die in Zusammenhang mit der Konvertierung auftreten können.

SQLSTATE	SQL-Meldungstext, Bedeutung und Meldungszeitpunkt
00 000	„SQL-Anweisung erfolgreich ausgeführt“: Bedeutung: Die Wertekonvertierung war ohne Einschränkung möglich. Meldungszeitpunkt: DO
01 004	„Zeichen einer Zeichenkette am Ende abgeschnitten“ Bedeutung: Es liegt ein Problem in der Klasse der alphanumerischen Datentypen beim Lesen vor. Die Länge des SQL-Wertes unterscheidet sich um mindestens ein Zeichen von der Länge der DRIVE-Variablen. Es wird eine verkürzte Konvertierung durchgeführt und mit einer SQL-Warnung darauf hingewiesen. Die DRIVE-Variable enthält somit eine rechtsbündig verkürzte alphanumerische Zeichenkette. Meldungszeitpunkt: DO
22 001	„Signifikante Zeichen einer Zeichenkette am Ende abgeschnitten“ Bedeutung: Der Wertebereich ist nicht verträglich. Es liegt ein Problem in der Klasse der alphanumerischen Datentypen beim Schreiben vor. Die Länge des zugewiesenen Wertes unterscheidet sich von der Länge der SQL-Column (mindestens 1 Zeichen zu viel). Die Konvertierung wird abgebrochen, die SQL-Anweisung wird nicht ausgeführt. Meldungszeitpunkt: DO

SQLSTATE	SQL-Meldungstext, Bedeutung und Meldungszeitpunkt
22 003	„Numerischer Wert zu groß oder zu klein“ Bedeutung: Der Wertebereich ist nicht verträglich. Es liegt ein Problem in der Klasse der numerischen Datentypen vor (Lesen oder Schreiben). Der Absolutwert der zu konvertierenden Zahl passt nicht in den Wertebereich des Zieldatentyps. Die Konvertierung wird abgebrochen, die SQL-Anweisung wird nicht ausgeführt. Meldungszeitpunkt: DO
42 SR1	„Werte nicht vergleichbar“ Bedeutung: Die Datentypen sind nicht verträglich. Die Wertekonvertierung zwischen SQL-Column und dem Bezugsdatentyp ist nicht möglich (Lesen oder Schreiben). Der SQL-Auftrag kann nicht ausgeführt werden. Meldungszeitpunkt: statisch bei COMPILE, dynamisch bei DO

Reaktion auf Konvertierungsfehler

Die folgende Liste zeigt, wie DRIVE auf Konvertierungsfehler reagiert und welche Maßnahmen im Programm ergriffen werden können.

- SQLSTATE=01004

  Bei diesem SQLSTATE wurde die SQL-Anweisung zwar ausgeführt, es wurde jedoch ein verkürzter Datentyp übergeben.

  DRIVE leitet die Warnung um in die &DML_STATE- Ausnahmebehandlung SQL CONV WARNING. Der Default-Fehlerausgang für diese Warnung ist CONTINUE, d.h. standardmäßig wird das Programm fortgesetzt.

  Mit WHENEVER &DML_STATE IN ('SQL CONV WARNING') *aktion* kann das Programm die Kontrolle übernehmen und in einem Fehlerausgang z.B. wie folgt auf die Warnung reagieren:
  – CONTINUE, falls die Programmlogik eine Warnung erwartet,
  – BREAK PROCEDURE, falls die Programmlogik keine Warnung erwartet.

- SQLSTATE=22001, 22003 und 42SR1 (dynamisch)

  Bei diesen SQLSTATEs wurde die SQL-Anweisung nicht ausgeführt.

  DRIVE leitet die Information um in die &DML_STATE- Ausnahmebehandlung SQL ERROR. Der Default-Fehlerausgang für diese Warnung ist BREAK PROCEDURE, d.h. standardmäßig wird das Programm abgebrochen.

Mit WHENEVER &DML_STATE IN ('SQL ERROR') *aktion* kann das Programm die Kontrolle übernehmen. Es wird jedoch nicht empfohlen, das Programm mit CONTINUE fortzusetzen. Vielmehr sollten die eingesetzten DRIVE-Variablen überprüft und ggf. korrigiert werden. Möglicherweise liegt ein Implementierungsfehler oder eine nicht abgestimmte Datenbankänderung vor.

6.1.3 Numerische Datentypen im SQL-Umfeld

(Zu Abschnitt 3.2.1.2 auf Seite 30)

Für Festpunktzahlen mit höchstmöglicher Genauigkeit werden bei SQL die Datentypen NUMERIC und DECIMAL verwendet, die eine Genauigkeit bis maximal 31 Stellen bieten. Auf DRIVE-Seite gibt es für diese Fälle den Datentyp XDEC (Extended DECIMAL) mit den entsprechenden Eigenschaften.

Daher sollten Sie XDEC immer dann verwenden, wenn ein SQL-Datenfeld vom Typ NUMERIC oder DECIMAL über eine DRIVE-Variable gelesen oder geschrieben wird, denn kein anderer DRIVE-Datentyp bietet eine derartige Genauigkeit, auch DOUBLE hat z.B. nur maximal 16 Stellen Genauigkeit.

Wenn Sie Werte über Variablen übergeben wollen und dabei maximale Genauigkeit erforderlich ist, beachten Sie bitte folgendes:

- Bei statischen SQL-Anweisungen kann der Datentyp XDEC uneingeschränkt bis zur maximalen Genauigkeit von 31 Stellen verwendet werden.
- In dynamischen SQL-Anweisungen bieten der DRIVE-Datentyp XDEC bzw. die SQL-Datentypen NUMERIC und DECIMAL nur dann maximale Genauigkeit, wenn nicht mehr als 18 Stellen deklariert sind. Dies liegt daran, dass bei dynamischer SQL intern die ESQL-COBOL-Schnittstelle verwendet wird, welche nur maximal 18 Stellen erlaubt.

  Sind mehr als 18 Stellen deklariert, dann wird zuerst die maximal mögliche Anzahl von Vorkommastellen in die COBOL-Deklaration übernommen und danach erst die Nachkommastellen. Wird die Anweisung ausgeführt, dann hängt das Verhalten vom übergebenen Wert ab, siehe auch Beispiel 2 und 3:
  - Falls nicht alle Nachkommastellen übernommen werden können, dann werden diese gerundet.
  - Falls nicht alle Vorkommastellen übernommen werden können, dann wird der Aufruf mit SQLSTATE=22003 zurückgewiesen.

  Um diese Schwierigkeiten zu vermeiden, sollten Sie in solchen Fällen einen statischen Aufruf verwenden.Davon betroffen sind die SQL-Anweisungen SELECT und OPEN CURSOR/FETCH zum Lesen aus der Datenbank sowie INSERT INTO und UPDATE zum Schreiben in die Datenbank.

Die oben genannten Einschränkungen gelten nicht, wenn Sie den DRIVE-Wert durch ein numerisches Literal oder eine Konstante übergeben.

Beispiele

1. Ein SQL-Datenbankfeld vom Typ NUMERIC (30,0) wird statisch mit DRIVE gelesen. Wenn maximale Genauigkeit garantiert sein soll, müssen Sie die DRIVE-Variable wie folgt deklarieren:

   ```
   DECLARE VAR &VAR1 XDEC (30,0)
   ```

   Falls Sie &VAR1 als INTEGER deklarieren, dann wird das Lesen abgelehnt mit SQLSTATE=22003 (Value too large).
   Wenn Sie &VAR als DOUBLE deklarieren, dann wird der Wert zwar gelesen (SQLSTATE=OK), jedoch nur mit einer Genauigkeit von 16 Stellen!

2. Ein SQL-Datenbankfeld vom Typ NUMERIC (24,4) wird mit einer dynamischen SELECT-Anweisung gelesen. Dieser Typ wird an der ESQL-COBOL-Schnittstelle umgewandelt in COBOL-DECIMAL (18,0), d.h. es werden 18 der 20 möglichen Vorkommastellen zugelassen. Dies führt zu folgendem Verhalten:

   – Ein Wert mit maximal 18 Vorkommastellen ohne Nachkommastellen wird unverändert gelesen.

   – Ein Wert mit maximal 18 Vorkommastellen und ein oder mehreren Nachkommastellen wird so gerundet, dass alle Nachkommastellen wegfallen.

   – Ein Wert mit mehr als 18 Vorkommastellen kann nicht gelesen werden, der Aufruf wird mit SQLSTATE 22003 abgelehnt.

   Entsprechendes gilt für das Schreiben in die Datenbank.

3. Ein DRIVE-Feld vom Typ XDEC (20,3) wird mit einer dynamischen UPDATE-Anweisung in die Datenbank geschrieben. Dieser Typ wird an der ESQL-COBOL-Schnittstelle umgewandelt in COBOL-DECIMAL (18,1), d.h. es werden alle 17 Vorkommmastellen sowie eine Nachkommastelle zugelassen. Dies hat folgenden Effekt:

   – Ein Wert mit maximal einer Nachkommastelle wird unverändert in die Datenbank geschrieben.

   – Ein Wert mit 2 oder 3 Nachkommastellen wird immer auf eine Nachkommastelle gerundet.

   Entsprechendes gilt für das Lesen aus der Datenbank.

6.2 Fehler und Warnungen abfangen, Endekriterien

Mit Hilfe der DRIVE-Anweisung WHENEVER (siehe Seite 164) können Sie Fehler abfangen und auf Warnungen reagieren.

6.2.1 Reaktion auf Fehler

(Änderung und Ergänzungen zu Abschnitt 4.2)

Tritt in einem Programm ein Ablauffehler auf oder wird ein ungültiger Variablenwert zugewiesen, dann wird das Programm im Standardfall abgebrochen.

Wenn Sie einen solchen Programmabbruch verhindern wollen, müssen Sie mit der Anweisung WHENEVER einen Fehlerausgang definieren.

6.2.1.1 Reaktion auf Ablauffehler

Bei Ablauffehlern in Programmen unterscheidet man die beiden Fehlerklassen ERROR (DRIVE-Fehlermeldungen) und DML_STATE (Datenbank-Fehlermeldungen)

Je nachdem, ob Sie Fehler der Klasse ERROR oder der Klasse DML_STATE abfangen wollen, müssen Sie eine der folgenden Varianten der WHENEVER-Anweisung verwenden:

– WHENEVER &ERROR IN (error, ...) CONTINUE

– WHENEVER &DML_STATE IN (status, ...) CONTINUE

– WHENEVER &ERROR IN (error, ...) BREAK

– WHENEVER &DML_STATE IN (status, ...) BREAK

– WHENEVER &ERROR IN (error, ...) CALL subprogname

– WHENEVER &DML_STATE IN (status, ...) CALL subprogname

Sie können mit WHENEVER alle Einträge der &ERROR- und &DML_STATE-Systemvariablen abfragen (siehe Abschnitt „Systemvariablen“ im Handbuch „DRIVE - Programmiersprache“ [2]).

Beispiel

```
WHENEVER &ERROR IN ('INDEX ERROR')
   CALL fehlerbearb;
```

Reaktionen bei den Fehlerklassen ERROR und DML_STATE

Werden die Anweisungen fehlerfrei ausgeführt, erhalten die Felder &ERROR bzw. &DML_STATE den Wert 'OK' zugewiesen. &SQL_STATE erhält den Wert 00000 zugewiesen.

Tritt ein Ablauffehler auf, der zu den bei WHENEVER beschriebenen Fehlerklassen gehört, dann kommt es zu unterschiedlichen Aktionen:

– Die Ausführung der aktuellen Anweisung wird in jedem Fall abgebrochen. Geänderte Variablenwerte sind dann nicht mehr gültig, die Variablen erhalten ihre alten Werte.

  Eine Ausnahme bildet die Anweisung DISPLAY SCREEN, bei der die Datenwerte aus technischen Gründen teilformatspezifisch und nicht anweisungsspezifisch konsistent gehalten werden.

– Ist zum aufgetretenen Fehler keine Aktion definiert, wird das Programm, außer bei „TABLE END“, „DIRTY READ“, „SQL CONV WARNING“ „OK END“, „TOO LONG“ und „TOO SHORT“, abgebrochen. Das Verhalten ist so, als wäre die Aktion BREAK angegeben.

– Ist zum aufgetretenen Fehler die Aktion CONTINUE definiert, wird das Programm hinter der fehlerhaften Anweisung fortgeführt.

– Ist zum aufgetretenen Fehler die Aktion CALL *subprogname* definiert, wird zunächst das interne Unterprogramm *subprogname* aufgerufen und das rufende Programm danach hinter der fehlerhaften Anweisung fortgesetzt.

  Tritt allerdings bei der Abarbeitung des Unterprogramms *subprogname* ein weiterer Fehler auf, wird das Programm unabhängig von definierten Fehlerausgängen abgebrochen. Beim Ausführen der SYSTEM-Anweisung wird in jedem Fall, d.h. bei fehlerfreiem Ausführen und im Fehlerfall, die entsprechende Systemvariable mit dem Returncode des gerufenen Subsystems versorgt (siehe Anweisung WHENEVER auf Seite 164).

  Bei der Abarbeitung des internen Unterprogramms *subprogname* wird die Systemvariable &ERROR_STATE nicht mehr verändert, so dass die darin enthaltene Fehlerinformation in diesem Unterprogramm abgefragt werden kann. Auch wenn in *subprogname* ein externes Unterprogramm aufgerufen wird, bleibt der Inhalt von &ERROR_STATE erhalten.

– Wenn das Datenbanksystem SESAM eine Transaktion intern zurücksetzt (SQLSTATE=40xxx), dann kann das DRIVE-Programm nicht auf den SQLSTATE reagieren, der das interne Rücksetzen der Transaktion ausgelöst hat. In diesem Fall können Sie also keinen Fehlerausgang mit WHENEVER programmieren.

6.2.1.2 Fehlerausgang bei Zuweisung ungültiger Variablenwerte

Wird einer Variablen ein ungültiger Wert zugewiesen, wird im Standardfall das Programm abgebrochen.
Wenn Sie einen solchen Programmabbruch verhindern wollen, müssen Sie einen 'CHECK ERROR'- oder 'CONVERSION ERROR'-Fehlerausgang definieren mit einer der beiden Anweisungen:

– WHENEVER &ERROR IN ('CHECK ERROR', 'CONVERSION ERROR') CONTINUE

– WHENEVER &ERROR IN ('CHECK ERROR', 'CONVERSION ERROR')
CALL subprogname

In beiden Fällen gilt:

– die fehlerhafte Zuweisung wird nicht ausgeführt,

– der ursprüngliche Inhalt der Variablen bleibt bestehen,

– DRIVE versorgt die Systemvariable &ERROR_STATE: Die Komponente &ERROR erhält den Wert 'CHECK ERROR' oder 'CONVERSION ERROR', die Komponente &VAR_NAME wird mit dem Namen der Variablen versorgt, die den Fehler verursachte sowie mit LINE_NR, DISPLACEMENT STATEMENT.

6.2.1.3 Abbildung von SQLSTATE auf &DML_STATE und &SQL_STATE

(neuer Abschnitt)

Die folgende Tabelle zeigt, wie der genormte SQL-Retuncode SQLSTATE auf die DRIVE-Systemvariablen &DML_STATE und &SQL_STATE abgebildet wird.

SQLSTATE	&DML_STATE	&SQL_STATE	&SQL_CLASS	&SUB_CLASS
00000	'OK'	00000	00	000
01004	'SQL CONV WARNING'	01004	01	004
01SA1	'DIRTY READ'	01SA1	01	SA1
01xxx	'OK'	01xxx	01	xxx
02xxx	'TABLE END'	02xxx	02	xxx
07xxx	'SQL ERROR'	07xxx	07	xxx
21xxx	'SQL ERROR'	21xxx	21	xxx
22020	RAISE 4100	22020	22	020
22021	RAISE 4100	22021	22	021
22xxx	'SQL ERROR'	22xxx	22	xxx

SQLSTATE	&DML_STATE	&SQL_STATE	&SQL_CLASS	&SUB_CLASS
23xxx	'SQL ERROR'	23xxx	23	xxx
24xxx	'CURSOR SQL ERROR'	24xxx	24	xxx
25xxx	'SQL ERROR'	25xxx	25	xxx
26xxx	'SQL ERROR'	26xxx	26	xxx
28xxx	'SQL ERROR'	28xxx	28	xxx
2Dxxx	RAISE 4100	2Dxxx	2D	xxx
33xxx	RAISE 4100	33xxx	33	xxx
34xxx	RAISE 4100	34xxx	34	xxx
3Dxxx	'SQL ERROR'	3Dxxx	3D	xxx
3Fxxx	'SQL ERROR'	3Fxxx	3F	xxx
40xxx	'TA CANCELLED'	40xxx	40	xxx
42SE2	RAISE 4100	42SE2	42	SE2
42SH2	RAISE 4100	42SH2	42	SH2
42xxx	'SQL ERROR'	42xxx	42	xxx
44xxx	'SQL ERROR'	44xxx	44	xxx
51xxx	'ADMIN SYS ERROR'	51xxx	51	xxx
52xxx	'ADMIN SYS ERROR'	52xxx	52	xxx
55xxx	'ADMIN SYS ERROR'	55xxx	55	xxx
56xxx	'ADMIN SYS ERROR'	56xxx	56	xxx
57xxx	'ADMIN SYS ERROR'	57xxx	57	xxx
58xxx	'ADMIN SYS ERROR'	58xxx	58	xxx
59xxx	'SYSTEM ERROR'	59xxx	59	xxx
81SA2	'ADMIN SYS ERROR'	81SA2	81	SA2
81SA6	'TEMP SYS ERROR'	81SA6	81	SA6
81SB0	'TEMP SYS ERROR'	81SB0	81	SB0
81SB1	'TEMP SYS ERROR'	81SB1	81	SB1
81SB2	'TEMP SYS ERROR'	81SB2	81	SB2
81SB5	'DB NOT AVAILABLE'	81SB5	81	SB5
81SB7	'SQL ERROR'	81SB7	81	SB7
81SBA	'TEMP SYS ERROR'	81SBA	81	SBA
81SC7	'TEMP SYS ERROR'	81SC7	81	SC7
81SCA	'TEMP SYS ERROR'	81SCA	81	SCA
81SD2	'ADMIN SYS ERROR'	81SD2	81	SD2

SQLSTATE	&DML_STATE	&SQL_STATE	&SQL_CLASS	&SUB_CLASS
81SP3	'TEMP SYS ERROR'	81SP3	81	SP3
81xxx	'SYSTEM ERROR'	81xxx	81	xxx
91xxx	'LIMIT REACHED'	91xxx	91	xxx
95xxx	'SQL ERROR'	95xxx	95	xxx

6.2.2 Reaktion auf SQL-Warnungen

(neuer Abschnitt)

SQL-Warnungen liegen vor, wenn nach der ordnungsgemäßen Durchführung einer SQL-Anweisung der SQLSTATE=01xxx zurückgegeben wird (xxx = Subklasse). Die Warnung kann zwei Arten von Ursachen haben:

– Warnung zu einem gelesenen Datenbanksatz

– Warnung der SQL-Schnittstelle

Mit der Anweisung WHENEVER können Sie in DRIVE-Programmen auf diese SQL-Warnungen reagieren.

Warnungen zu einem gelesenen Datenbanksatz

Zu einem von DRIVE gelesenen Datenbanksatz können folgende Warnungen auftreten:

SQLSTATE=01004: „Gelesene Zeichenkette rechtsbündig verkürzt“
SQLSTATE=01SA1: „Gelesener Satz von Updatetransaktion geändert“

DRIVE leitet diese Warnungen um in folgende &DML_STATE-Ausnahmebehandlung:

SQLSTATE=01004: &DML_STATE = ’SQL CONV WARNING’, siehe auch Seite 173
SQLSTATE=01SA1: &DML_STATE = ’DIRTY READ’

Der Default-Fehlerausgang für beide Warnungen ist CONTINUE, d.h. das Programm wird standardmäßig ohne weitere Aktion fortgesetzt.

Auf diese Warnungen kann mit WHENEVER &DML_STATE IN (’...’) *aktion* reagiert werden.

Warnungen der SQL-Schnittstelle

Eine Warnung der SQL-Schnittstelle ist aus SQL-Sicht ein ’Gutfall’, d.h. &DML_STATE=’OK’. In SQLSTATE wird 01xxx zurückgegeben (xxx ungleich 01004 und 01SA1). Dabei ist xxx die Subklasse, die genaue Auskunft über die Ursache der Warnung gibt.

DRIVE leitet diese Warnungen um in folgende &WARNING-Ausnahmebehandlung:

&WARNING = 'SQL WARNING'

Der Default-Fehlerausgang ist CONTINUE, d.h. das Programm wird standardmäßig ohne weitere Aktion fortgesetzt.

Mit WHENEVER &WARNING IN ('SQL WARNING') *aktion* können Sie einen Fehlerausgang programmieren. Dabei ist zu beachten:

- Mit der WHENEVER-Anweisung in dieser Form wird explizit auf Warnungen des SQL2-Subsystems reagiert. Wenn Sie die IN-Klausel weglassen, dann werden Warnungen von allen Subsystemen verarbeitet, die diese Schnittstelle aktuell bedienen können. Da künftig weitere Subsysteme hinzukommen können, sollten Sie immer die IN-Klausel programmieren.
- Falls kein Warnungsfall vorliegt, ist &WARNING mit 'WARNING OK' versorgt.
- Der Fehlerausgang kann z.B. wie folgt auf die Warnung reagieren:
  - CONTINUE, falls die Programmlogik eine Warnung erwartet,
  - BREAK PROCEDURE, falls die Programmlogik keine Warnung erwartet.
- Der Text der SQL-Warnung kann mit Hilfe der Funktion SQLMSGSTRING ermittelt werden, näheres siehe Seite 169.

6.3 Dynamische SQL-Anweisungen

(Ergänzung zu Abschnitt 4.6.1)

Sie können mit EXECUTE alle in Kapitel 4 aufgeführten Anweisungen dynamisch ausführen, indem Sie sich mit den Stringfunktionen zum Ablaufzeitpunkt einen String erzeugen, dessen Inhalt eine dynamisch ausführbare SQL-Anweisung ist (siehe Abschnitt „4.5.1 Stringfunktionen“ in „DRIVE Programmiersprache“ [2]).

6.4 Abkürzung „.*“

(Ergänzung zu Abschnitt 3.5.1 auf Seite 72)

Bei einer strukturierten Variablen &variable bedeutet &variable.* die abkürzende Schreibweise für die Liste aller Komponenten auf der nächsten Stufe.

Beispiel

```
DCL VAR
   1 &a,
      2 a1,
         3 a11 CHAR (1),
         3 a12 INT,
         3 a13 DATE,
      2 a2 CHAR (2),
      2 a3 INT;
```

`&a.*` ist dann gleichbedeutend mit `&a.a1,&a.a2,&a.a3;`

Einsatzbereich von „.*“

Die folgende Tabelle zeigt, bei welchen DRIVE-Anweisungen „.*“ eingesetzt werden kann und welche Einschränkungen dabei gelten.

DRIVE-Anweisung	**Einschränkung**
CALL	nicht bei RETURN
CASE	(1)
CYCLE	(1)
DCL FORM	nicht bei RETURN
DCL LIST	nicht bei RETURN
DISPLAY FORM	nicht bei RETURN
DISPLAY LIST	(keine Einschränkung)
DO	(keine Einschränkung)
ENTER	(keine Einschränkung)
FILL	nicht bei RETURN
IF	(1)
READ FILE	(keine Einschränkung)
SEND MESSAGE	(keine Einschränkung)
SET	nur auf der rechten Seite innerhalb eines Aggregats
WRITE FILE	(keine Einschränkung)

(1) In der Bedingung ist eine *-Variable nur in folgenden Fällen erlaubt:
bei '=' nur auf der rechten Seite innerhalb eines Aggregats
bei 'IN' nur auf der rechten Seite als Liste von Werten

6.5 Einschränkungen und Inkompatibilitäten

6.5.1 Deklaration von Variablen mit dem Konstrukt LIKE TABLE

(Ergänzung zu Abschnitt 3.3.3 auf Seite 49)

Es können in einem DRIVE-Programm maximal 36 Variablen mit der Komponentenstruktur einer SQL-Tabelle deklariert werden:

```
DECLARE VARIABLE &variable LIKE TABLE tabelle
```

Werden auf diese Weise mehr als 36 Variablen deklariert, dann wird die Übersetzung des DRIVE-Programms (COMPILE) mit dem Fehler DRI0032 beendet. Die Übersetzungsliste enthält ab der 36. Variablendeklaration mit LIKE TABLE folgenden Fehlertext:

```
DRI0536 42SF0 Cursorname not unique
```

6.5.2 Namenseindeutigkeit von DRIVE-SQL-Programmen

(Ergänzung zum „DRIVE-Lexikon“ [3], Kapitel 3, Beschreibung der COMPILE-Anweisung)

Wird ein DRIVE-Programm mit statischen SQL-Anweisungen übersetzt, dann erzeugt dieser einen SQL-Zwischenmodul.

Der Name dieses Zwischenmoduls ist maximal 7 Zeichen lang und wird aus dem ursprünglichen Programmnamen (= Namen des PLAM-Bibliothekselements) gebildet. Ist dieser länger als 7 Zeichen, dann wird er nach der 4-3-Regel verkürzt, d.h. es werden die 4 ersten und die 3 letzten Zeichen genommen. Der Name wird durch einen internen Suffix ergänzt, damit der Modul trotz der Verkürzung in einer DRIVE-Session eindeutig identifiziert werden kann.

Für diesen Suffix sind maximal 191 Ausprägungen möglich.

```
COMPILE program ...
```

Werden in einer DRIVE-Session mehr als 191 Programme mit gleichem SQL-Modulnamen ausgeführt, dann gibt DRIVE ab dem 192. Programm folgende Fehlermeldung aus:

```
DRI0536 81SD1 Modulname used for two single sql-modules
```

6.5.3 Geändertes Verhalten der WHENEVER-Anweisung

Tritt ein Konvertierungsfehler mit SQLSTATE=01004 auf, dann verhält sich die WHENEVER-Anweisung anders als in der Vorgängerversion:

- Enthält ein DRIVE-Programm die Anweisung `WHENEVER &DMLSTATE` *aktion*, dann wird *aktion* jetzt ausgeführt.
- Enthält ein DRIVE-Programm die Anweisung `WHENEVER &ERROR` *aktion*, dann wird *aktion* jetzt nicht mehr ausgeführt, da SQLSTATE=01004 nicht mehr in einen CONVERSION ERROR umgesetzt wird.

Näheres zu SQLSTATE=01004 finden Sie auf Seite 173.

6.5.4 Multiple Variablen in der SELECT-Liste einer SQL-Anweisung

(Ergänzung zum „DRIVE-SQL-Lexikon“ [4], Kapitel 3, SELECT-Anweisung auf Seite 135)

Multiple Variablen in der SELECT-Liste sind nur in einer statischen SELECT-Anweisung erlaubt.

`SELECT` *select-liste* `INTO,...`

Wird eine multiple Variable wie z.B. `...,&var(1-3),...` bei einer dynamischen SELECT-Anweisung angegeben, dann wird diese mit SQLSTATE=42SL3 und dem Meldungstext `Platzhalter „?“ in der SELECT-Liste` zurückgewiesen.

6.5.5 Insert-Column-Liste bei der SQL-Anweisung INSERT

(Ergänzung zum „DRIVE-SQL-Lexikon“ [4], Kapitel 3, INSERT-Anweisung auf Seite 111)

Bei der SQL-Anweisung INSERT darf in der Insert-Column-Liste die Gesamtanzahl der Spalten und Ausprägungen (multiplen Spalten) maximal 1000 betragen. Diese Einschränkung gilt sowohl für statische als auch für dynamische INSERT-Anweisungen.

`INSERT INTO` *tabelle* `(`*insert-column-liste*`) VALUES ( ... )`

Hat die Liste mehr als 1000 Spalten/Ausprägungen, dann wird die INSERT-Anweisung mit SQLSTATE=07009 abgewiesen. Dieser Grenzwert kann insbesondere dann überschritten werden, wenn man wie im folgenden Beispiel multiple Spalten verwendet.

Beispiel

Folgende INSERT-Anweisung hat 1001 Terme in der Liste und wird daher mit SQLSTATE=07009 abgewiesen:

```
INSERT INTO tabelle1 (
                      feld1,
                      multi-feld1(250),
                      multi-feld2(250),
                      multi-feld3(250),
                      multi-feld4(250)
                     )
       VALUES        (
                      wert1,
                      &multivar1,
                      &multivar2,
                      &multivar3,
                      &multivar4
                     )
```

7 Datenbanken

7.1 Datenbanken bearbeiten

Eine Einführung in die Struktur einer relationalen Datenbank sowie in die Begriffe und Konzepte von SQL finden Sie in den SESAM-Handbüchern „SESAM/SQL-Server V2“ [6] und [8].

Dieses Kapitel weist stichwortartig auf die wichtigsten Bearbeitungsmöglichkeiten hin.

Eine Datenbank ist die 'physische' Datenbasis, die abgefragt und verändert werden kann. Sie umfasst die Datensätze sowie die Beschreibung der Sätze und ihrer logischen Organisation (Metadaten). Aus relationaler Sicht ist jede Datenbank logisch in Basistabellen und Systemtabellen organisiert.

Basistabellen sind im relationalen Schema der Datenbank definiert.

Die Datensätze einer Datenbank können gelesen (Selektions-Operation) oder neu eingefügt, gelöscht oder in ihren Werten geändert werden (Manipulations-Operationen).

Beim **Selektieren** kann mit der SELECT-Anweisung ein einzelner Satz aus der Datenbank gelesen werden. Gibt es mehrere Treffer, gibt DRIVE eine Fehlermeldung aus.

Sollen mehrere Sätze gelesen werden, muss eine Ergebnistabelle (Cursortabelle) spezifiziert werden, die die ausgewählten Sätze enthält. Diese Ergebnistabelle wird durch das Deklarieren eines Cursor (Satzzeiger) mit der Metavariablen *cursorbeschreibung* erzeugt (siehe „DRIVE-SQL-Lexikon“ [4], Metavariable *cursorbeschreibung*).

Beim **Manipulieren** können mit **einer** SQL-Anweisung mehrere Sätze, die bestimmte Bedingungen erfüllen, in **gleicher** Weise geändert werden (**mengenorientierte Änderung**, siehe „DRIVE-SQL-Lexikon“ [4], Metavariable *abfrageausdruck*).

Sollen hingegen **mehrere** Sätze auf **verschiedene** Weise geändert werden, so müssen sie in einer Ergebnistabelle (Cursortabelle) ausgewählt und mit dem Cursor einzeln angesprochen werden (**satzweise Änderung**). Geändert wird die Basistabelle, die dem Cursor bei der Deklaration zu Grunde liegt.

Eine Cursortabelle brauchen Sie auch, wenn Sie einen oder mehrere Sätze verändern oder löschen und die Bedingungen dafür erst zum Ablaufzeitpunkt festlegen wollen.

Beim **Einfügen** von Sätzen in die Datenbank bewirkt die RETURN-Klausel der INSERT-Anweisung, dass sich der DRIVE-Anwender die vom Datenbank-System vergebene Zahl zur Identifizierung eines Satzes in eine DRIVE-Variable ausgeben lassen kann durch den Inhalt des Zählfeldes beim Compound Key (*-Angabe in der Klausel VALUES).

Sätze werden aus einer Tabelle ausgewählt durch die Angabe von Bedingungen in der WHERE-Klausel der SELECT-Anweisung oder der Metavariablen *select-ausdruck*. Mehrere Bedingungen können durch die logischen Operatoren AND oder OR (siehe „DRIVE-SQL-Lexikon“ [4], Metavariable *bedingung*) verbunden werden. Zusätzlich können Ergebnistabellen (Views, Cursortabellen) Datensätze aus mehreren Tabellen enthalten. Die Tabellen werden verknüpft (Join), indem:

1. in der FROM-Klausel der SELECT-Anweisung oder der Metavariablen *select-ausdruck* alle betroffenen Tabellen angegeben werden und
2. entweder in der WHERE-Klausel ein Satzelement einer Tabelle mit dem Satzelement einer anderen Tabelle durch den Vergleichsoperator „=“ verbunden wird (dabei müssen die Satzelemente, die verglichen werden, den gleichen Datentyp haben)

   oder innerhalb der FROM-Klausel eine Joinbedingung durch eine ON-Klausel referenziert wird.

Um beim Selektieren und Manipulieren auf die Sätze einer Cursortabelle zugreifen zu können, muss die Cursortabelle mit OPEN geöffnet und der Cursor mit FETCH auf die einzelnen Sätze positioniert werden.

Die aktuelle Cursorposition kann mit STORE über das Transaktionsende hinaus gespeichert und in einer späteren Transaktion mit RESTORE wiederhergestellt werden.

Für das Arbeiten mit einer Cursortabelle ergibt sich folgende Reihenfolge von Anweisungen:

DECLARE *cursor*....

OPEN *cursor*...

FETCH *cursor*...

Bearbeiten der Cursortabelle mit DELETE, UPDATE, falls der Cursor änderbar ist

STORE *cursor*

RESTORE *cursor*

CLOSE *cursor*

DROP CURSOR *cursor (*ggf.*)*

Views (Sichten) sind virtuelle Tabellen, die einen Ausschnitt aus einer oder mehreren Basistabellen definieren.

In der Deklaration des View wird ein Name vereinbart und eine Datenbank-Abfrage mit einem *abfrageausdruck* angegeben. Der View umfasst die Ergebnistabelle dieser Datenbank-Abfrage. Der Inhalt der Ergebnistabelle wird erst bestimmt, wenn sich eine Anweisung auf den deklarierten View bezieht. Wenn der View änderbar ist, können über den View auch Datensätze in die Basistabelle eingefügt, geändert oder gelöscht werden.

Transaktionen erhalten und sichern den konsistenten Zustand einer Datenbank. Eine Transaktion ist eine logische Folge von Anweisungen, die zwischen zwei Sicherungspunkten liegen. Bei Transaktionssicherung sollen entweder alle oder keine der in einer Transaktion liegenden Anweisungen ausgeführt werden. Der Beginn der ersten Transaktion wird nicht mit einer bestimmten SQL-Anweisung definiert. Die erste TA-initiierende SQL-Anweisung (siehe „DRIVE-Programmiersprache“ [2], Abschnitt 10.1.2) nach dem Programmstart oder dem nach Festschreiben oder Rücksetzen der vorherigen Transaktion wird vom Datenbanksystem als Transaktionsbeginn gewertet.

Fehlerbehandlung

Für bestimmte Fehlerklassen und DRIVE-Anweisungen können im Programm-Modus Reaktionen auf Fehler mit der WHENEVER-Anweisung festgelegt werden (siehe Abschnitt „Fehler und Warnungen abfangen, Endekriterien“ auf Seite 176). DRIVE reagiert dann nicht standardmäßig mit Programmabbruch, sondern führt die bei WHENEVER angegebenen Aktionen durch.

Fehler bei verteilter Verarbeitung können Sie in der Systemvariablen &ERROR_STATE.ERROR über den Wert 'DIS ERROR' abfragen.
Zusätzlich wird die Variable &DISTRIBUTION_STATE versorgt (siehe „DRIVE-Programmiersprache“ [2], Kapitel 12 und 13).

NULL-Wert-Darstellung

Mit der Anweisung PARAMETER DYNAMIC NULL FORM legen Sie für ein alphanumerisches und/oder numerisches Datenfeld je ein Zeichen für die Darstellung des NULL-Werts am Bildschirm fest (siehe auch „DRIVE-Lexikon“ [3], Anweisung PARAMETER DYNAMIC).

Für alphanumerische Datenfelder kann als NULL-Wertzeichen jedes beliebige Zeichen verwendet werden. Für numerische Datenfelder ist als NULL-Wertzeichen nur eine Ziffer oder eines der Sonderzeichen * + - , . zugelassen.

Das NULL-Wertzeichen für alphanumerische Datenfelder ist auch gültig für Felder der Zeit-Datentypen. Das NULL-Wertzeichen für numerische Datenfelder ist auch gültig für Felder des Datentyps INTERVAL.

Das mit PARAMETER DYNAMIC vereinbarte NULL-Wertzeichen kann für DRIVE-Bildschirmformate durch Angabe eines anderen NULL-Wertzeichens in der NULL-Klausel der DECLARE FORM-Anweisung überschrieben werden.
Ein NULL-Wertzeichen, das über DECLARE FORM NULL festgelegt wurde, gilt nur für das jeweilige mit DECLARE FORM definierte Bildschirmformat.

Wenn Sie kein NULL-Wertzeichen explizit festlegen, gilt für alphanumerische oder numerische Datenfelder das Zeichen @ als Standardvorgabewert.

Entsprechend können Sie mit der Anweisung PARAMETER DYNAMIC NULL LIST ein Zeichen für die Darstellung des NULL-Werts bei Druckerausgaben festlegen. Hier ist der Standardvorgabewert ein Punkt.

7.2 Datenbank-Unterstützung

Dieses Kapitel beschreibt:

- die Objekte, die die einzelnen Datenbanksysteme kennen und wie sie angesprochen werden (Seite 192)
- Besonderheiten bei SESAM V2 (Seite 194)
- Syntaxunterschiede zwischen SQL-Dialekten und DRIVE, insbesondere die DRIVE-Unterstützung der Cursorverarbeitung (ab Seite 197)
- Datenschutz über Zugriffsrechte und Benutzeridentifikationen bei SESAM/SQL (Seite 201).

Mit der 4GL-Sprache DRIVE können Datenbank-Anwendungen schnell und komfortabel entwickelt werden. Die Zugriffe auf Datenbanken werden in SQL (Structured Query Language) formuliert, der mit Abstand am weitesten verbreiteten relationalen Datenbank-Sprache.

Unterstützte DB-Server

DRIVE ermöglicht mit SQL-Anweisungen den Zugriff auf den DB-Server SESAM/SQL V2.

In einer DRIVE-Sitzung kann nur mit einem BS2000-Server gearbeitet werden.

Mit der Funktion der Verteilten Transaktionsverarbeitung VTV können Sie jedoch verteilte DRIVE-Anwendungen programmieren, die auf alle BS2000-Server zugreifen (siehe „DRIVE-Programmiersprache“ [2], Kapitel „Verteilte Transaktionsverarbeitung“).

Verweise auf andere Handbücher

Eine ausführliche Beschreibung der SQL-Anweisungen finden Sie in der SQL-Sprachbeschreibung „SQL für SESAM/SQL“ V2 [6].

Die genaue Syntax der DRIVE-SQL-Anweisungen wird in Kurzform im Handbuch „DRIVE-SQL-Lexikon“ [4] dargestellt.

Komplexe Anweisungsteile, die sowohl in DRIVE- als auch in SQL-Anweisungen enthalten sind, werden gesondert beschrieben im Kapitel Metavariablen des „DRIVE-Lexikon“ [3] und im DRIVE-SQL-Lexikon [4]. Die DRIVE-Metavariablen stimmen weitgehend mit denen der Sprachbeschreibung für SESAM V2 überein, d.h. es gibt eine eindeutige Zuordnung.

7.2.1 SQL-Objekte in DRIVE

Ein DB-Server wird angesprochen durch Transaktionsanweisungen sowie durch Referenzen auf Datenbanken oder SQL-Objekte, die von diesem DB-Server (DBH) verwaltet werden. Zu den **persistenten** Objekten, d.h. Objekten, die in der Datenbank gespeichert werden, gehören Basistabellen der Anwendungen, Systemtabellen des DB-Servers, Spalten, Constraints, Indizes, Views und Synonyme. Namen und Strukturen persistenter SQL-Objekte gehören zu den Metadaten der jeweiligen Datenbank und werden vom Datenbanksystem in Systemtabellen verwaltet. Als Anwendung verwaltet DRIVE keine Informationen über persistente SQL-Objekte. Mit der Anweisung DECLARE VARIABLE... LIKE TABLE... ist es allerdings möglich, Variablen mit der Struktur von Basistabellen, Systemtabellen oder Views zu definieren.

Namen und Strukturen von **temporären SQL-Objekten** gehören zu den anwendungsspezifischen Daten und werden vom SQL-Laufzeitsystem des DB-Servers verwaltet. Sie existieren höchstens bis zum Ende der DRIVE-Sitzung. Temporäre SQL-Objekte sind vor allem Cursor sowie temporäre Tabellen und Korrelationen. Mit der Anweisung DECLARE VARIABLE ... LIKE ... können Sie Variablen mit der Struktur von DB-Tabellen oder Cursortabellen definieren.

Mit dem Zuweisen einer Datenbasis sind DRIVE alle Daten und Metadaten bekannt, d.h. alle Basis- und Systemtabellen und Systemviews eines SQL-Schemas. Die aktuelle Datenbasis weisen Sie zu mit den Anweisungen PARAMETER DYNAMIC oder OPTION mit den CATALOG-, SCHEMA- und AUTHORIZATION-Operanden (siehe „DRIVE-SQL-Lexikon“ [4]).

Eine **Basistabelle** wird spezifiziert durch den Tabellennamen und die Tabellenstruktur. Die Tabellenstruktur wird durch Spalten und Tabellenconstraints spezifiziert. Optional können Sie die physikalische Speicherungsart spezifizieren. Die folgende Tabelle zeigt, wie Sie Tabellennamen für den von DRIVE unterstützten DB-Server qualifizieren:

Datenbankserver	**Tabellenname**
SESAM V2	[catalog-name .] [schema-name .] tabellenname zwei Relationen sind genau dann gleich, wenn sie den gleichen Katalog-, Schema- und Tabellennamen haben

Die folgende Tabelle zeigt die Speicheroptionen für den von DRIVE unterstützten DB-Server:

Datenbankserver	**Speicheroptionen**
SESAM V2	USING SPACE [catalogname .] space

Die (ggf. strukturierte) **Spalte** einer Basistabelle wird spezifiziert mit dem Spaltennamen, dem Datentyp, einem optionalen Defaultwert und einem optionalen Spaltenconstraint. Der Spaltenname ist ein tabellenweit eindeutiger Name. Außerhalb einer Tabellenspezifikation wird eine Spalte (oder eine oder mehrere Spaltenkomponenten) mit der Metavariablen *satzelement* spezifiziert (siehe „DRIVE-SQL-Lexikon“ [4]).

Es gilt folgender Zugriffsschutz: Ein autorisierter Benutzer (Grantor) vergibt **Privilegien** und berechtigt einen anderen Benutzer (Grantee), eine bestimmte Anweisung auf einem bestimmten SQL-Objekt auszuführen, z.B. SELECT auf die Tabelle eines Schemas. Mit der Anweisung GRANT vergeben Sie Privilegien, mit der Anweisung REVOKE entziehen Sie Privilegien und regeln so den Zugriffsschutz in SQL.

Eine Datenbank kann in **Schemata** aufgeteilt sein, die von Anwendern oder vom Datenbanksystem eingerichtet werden. Ein anwenderdefiniertes Schema ist einem Anwender zugeordnet, der Eigentümer des Schemas ist. Es enthält Metadaten zu Basistabellen, Views, Indizes und Privilegien. Systemdefinierte Schemata fassen Systemviews zusammen und enthalten datenbankspezifische Metadaten.

Views sind virtuelle Tabellen, die einen Ausschnitt aus einer oder mehreren Basistabellen definieren. In SESAM V2 gibt es persistente Views (siehe folgender Abschnitt). Der Viewname ist eindeutig in Bezug auf alle datenbankweiten, persistenten Views und Tabellen.

Sie können temporäre **Synonyme** für Tabellen- und Viewnamen (sog. Korrelationen) in Suchabfragen verwenden. Diese Synonyme existieren nur solange wie die Suchabfrage in Bearbeitung ist.

Sie können Tabellen- und Spalten**constraints** definieren, die aus dem optionalen Constraintnamen und der Spaltenangabe bestehen (siehe CREATE TABLE auf Seite 116 und ALTER TABLE auf Seite 89). Der Constraintname ist datenbankweit eindeutig. Für SESAM V2 unterstützt DRIVE z.Zt. Indizes nicht in der Syntax. Basistabellen, die außerhalb von DRIVE mit Indizes spezifiziert wurden, können jedoch in DRIVE benutzt werden.

Bei SESAM V2 können Sie mit einer etwas eingeschränkten SELECT-Anweisung, die Basistabellen und andere persistente Views referenziert, **persistente Views** definieren (siehe „DRIVE-SQL-Lexikon“ [4], CREATE VIEW). Die „SQL-Sprachbeschreibung von SESAM“ [8] enthält einen Überblick zu den Systemviews von SESAM V2.

7.2.1.1 SQL-Objekte definieren mit DDL-Anweisungen

Alle Nutzdaten werden logisch in Basistabellen organisiert, die der Anwender für SESAM V2 mit DDL(Data Definition Language)-Anweisungen erzeugt.

DRIVE bietet weitere DDL-Anweisungen für die Definition von Datenstrukturen und Integritätsbedingungen. Diese Anweisungen sind ausführbare SQL-Anweisungen und müssen daher im Ausführungsteil von Programmen stehen (siehe auch Kapitel „DRIVE-SQL-Anweisungen“ auf Seite 83 und „DRIVE-SQL-Lexikon“ [4])

i Achten Sie darauf, dass eine DDL- Anweisung im Ausführungsteil eines Programms keine deklarative Anweisung im Deklarationsteil eines Programms zerstört: Z.B. Anlegen eines Cursors im Deklarationsteil und Ändern der Tabelle, auf die der Cursor deklariert wurde, über die ALTER TABLE-Anweisung im Ausführungsteil des Programmes. Wird anschließend versucht, über den Cursor auf die Tabelle zuzugreifen, kommt es zu einem SQL-Fehler.

Die DDL-Anweisungen können auch dynamisch, d.h. zum Ablaufzeitpunkt, angegeben werden (siehe „DRIVE-Lexikon“ [3], Anweisung EXECUTE).

i In ein- und derselben Transaktion können entweder nur DDL-Anweisungen oder keine DDL-Anweisungen ausgeführt werden, d.h. ein Mischen von DML- und DDL-Anweisungen ist nicht möglich.

7.2.2 Hinweise zur DB-Unterstützung

DRIVE V2.2 unterstützt den DB-Server SESAM V2 mit

– allen numerischen, alphanumerischen und Zeitpunktausdrücken (siehe „DRIVE-SQL-Lexikon“ [4], Metavariable *sqlausdruck*).

– allen Bedingungen (Metavariable *bedingung*) und

– allen Abfrageblöcken (Metavariable *abfrageausdruck*).

Mit der Anweisung DECLARE VARIABLE...LIKE TABLE/CURSOR... werden SESAM-Datentypen auf DRIVE-Datentypen abgebildet (siehe „DRIVE-Programmiersprache“ [2], Kapitel „Variablen und Konstanten einsetzen“). DRIVE-Datentypen werden den SESAM-Ausdrücken in der SELECT-Liste von Cursorbeschreibungen entsprechend den SESAM V2-Konventionen zugewiesen, d.h. mit der SQL-Norm übereinstimmend (siehe „DRIVE-SQL-Lexikon“ [4], DECLARE CURSOR-Anweisung und Metavariable *abfrageausdruck*).

SESAM V2 verwendet die zwei Information Schemata

– INFORMATION_SCHEMA (benutzerspezifischer Zugriff),

– SYS_INFO_SCHEMA (Zugriff zunächst nur durch den universellen Benutzer).

Informationen aus diesen Schemata können Sie mit SELECT-Anweisungen und Cursor-Deklarationen abfragen (siehe „DRIVE-SQL-Lexikon“ [4]).

Bei SESAM V2 spricht man bei Datenbanken auch von Katalogen. Ein kann Katalog mehrere SQL-Schemata mit mehreren Basistabellen enthalten. Es gibt einfache Tabellennamen und mit Katalog- und Schemanamen qualifizierte Tabellennamen (siehe Seite 196).

Die Anweisung PERMIT hat bei SESAM V2 im New-Style keine Wirkung, sondern setzt nur den SQLSTATE. Sie wird jedoch im Old-Style für den Zugriff auf passwortgeschützte CALL-DML-Tabellen benötigt. Die Anweisung GRANT vergibt Privilegien, REVOKE entzieht sie.

Es gibt folgende Anweisungen zur Session-Einstellung:

– SET CATALOG - Datenbanknamen voreinstellen (nur für dynamische SQL)

– SET SCHEMA - Schemanamen voreinstellen (nur für dynamische SQL)

– SET SESSION AUTHORIZATION (Berechtigungsschlüssel für die aktuelle SQL-Session voreinstellen)

Zum Konzept des Konsistenzlevels und dem Beginn einer Transaktion siehe „DRIVE-Programmiersprache“ [2], Kapitel „Transaktionskonzept“.

7.2.2.1 SESAM V2-DBH zuordnen

DRIVE greift auf eine SESAM V2-Datenbank mit Hilfe eines zugeordneten DBH (Database Handler) zu. Dabei sind folgende Einsatzformen zu unterscheiden:

– lokaler Zugriff im BS2000 (TIAM- oder UTM-Betrieb)

  DRIVE wird mit einem Konnektionsmodul (SESMOD im TIAM-, SESUTMC im UTM-Betrieb) gebunden, dem beim Starten der Name eines „independent DBH“ und der Konfigurationsname mitgegeben wird. Dieser DBH wurde zuvor mit einer passenden Konfigurationsdatei gestartet. Der zugeordnete DBH kann bis zu 254 Datenbanken verwalten, auf deren Daten und Metadaten in einer DRIVE-Sitzung (TIAM-Betrieb) oder im UTM-Vorgang (UTM-Betrieb) zugegriffen werden kann. Voraussetzung: Die erforderlichen Zugangsberechtigungen sind vorhanden.

  Bei verteilten Datenbanken mit SESAM-DCN kann neben diesem „Home-DBH“ mit weiteren „Remote-DBH“ gearbeitet und auf die von ihnen verwalteten Datenbanken zugegriffen werden. Der Zugriff ist für den DRIVE-Anwender transparent. Zu einem Zeitpunkt kann jede Datenbank nur von einem DBH verwaltet werden.

– verteilte DRIVE-Anwendungen mit UTM-D (Verteilte Transaktionsverarbeitung VTV)

  Als Server wird im BS2000 eine UTM-Anwendung eingesetzt, die die UTM-D-Anwendung auf dem anderen BS2000-Rechner adressiert und umgekehrt. Die Zuordnung von DBH und Datenbanken wird durch die Generierung der UTM-D-Anwendung im Client/Server bestimmt (Auftraggeber/Auftragnehmer).

Zugangsberechtigungen zu einer SESAM V2-Datenbank

Beim Anlegen einer Datenbank mit der SESAM V2-UTILITY-Anweisung CREATE CATALOG durch einen befugten BS2000-Systembenutzer kann ein BS2000-Kennwort und muss ein Berechtigungsschlüssel angegeben werden. Das Kennwort wird auf alle Dateien der Datenbank vererbt und muss beim Starten eines DBH, der die Datenbank verwalten soll, angegeben werden. Der Berechtigungsschlüssel identifiziert einen SQL-Benutzer, dem eine BS2000-Systembenutzer-Kennung zugeordnet ist. Die Standard-Voreinstellung ist die Kennung, unter der die Anweisung CREATE CATALOG angegeben wurde.

Ein Berechtigungsschlüssel zusammen mit einer Systembenutzer-Kennung heißt Systemzugang. Der mit CREATE CATALOG festgelegte Systemzugang ist der erste für die Datenbank und identifiziert den sog. universellen Benutzer. Dieser kann mit der UDL-Anweisung CREATE USER weitere Berechtigungsschlüssel einrichten und mit CREATE SYSTEM USER Berechtigungsschlüssel zu Systemzugängen erweitern.

Wollen Sie mit einem DRIVE-Programm auf eine SESAM V2-Datenbank zugreifen, muss ein Systemzugang eingerichtet sein, der je nach gewählter Einsatzform folgenden Bedingungen genügt:

– bei lokalem Zugriff im BS2000 muss die Systembenutzer-Kennung mit der Kennung von DRIVE als TIAM- oder UTM-Anwendung übereinstimmen

– Bei verteilten Anwendungen muss nur der Client als SQL-Benutzer ausgewiesen werden.

In DRIVE weist man sich über den Operanden AUTHORIZATION der PARAMETER DYNAMIC oder der OPTION-Anweisung als SESAM V2-Benutzer aus.
Die Angabe AUTHORIZATION gilt für die DRIVE-Sitzung und kann in transaktionslosem Zustand durch die DRIVE-SQL-Anweisung SET SESSION geändert werden (siehe „DRIVE-SQL-Lexikon“ [4]).

7.2.2.2 Kataloge und SQL-Schemata angeben

Durch die Organisation von Tabellen, Views und Indizes in Datenbanken und Schemata kann jedes dieser SQL-Objekte nur in Bezug auf einen bestimmten Katalog und ein bestimmtes SQL-Schema eindeutig identifiziert werden:
Der einfache Objektname wird qualifiziert mit dem Schemanamen und dem Katalognamen `[ [catalog-name . ] [schema-name . ] ]`.

Werden der Katalogname und/oder der Schemaname nicht angegeben, gelten die internen Standardvoreinstellungen. Diese ergeben sich für statische SQL aus den Operanden CATALOG und SCHEMA der OPTION-Anweisung; für dynamische SQL werden sie mit den Anweisungen SET CATALOG und SET SCHEMA eingestellt oder über die Anweisung PARAMETER DYNAMIC (siehe „DRIVE-SQL-Lexikon“ [4]).

Für die Übersichtlichkeit einer DRIVE-Anwendung mit SESAM/SQL V2 wird empfohlen, die Anwendung so in einzelne Programme zu strukturieren, dass jedes Programm zumindest statisch hauptsächlich auf SQL-Objekte eines Schemas zugreift. Dieses Schema und der dazugehörige Katalog können dann in der OPTION-Anweisung eingestellt werden. Zugriffe auf SQL-Objekte eines anderen Schemas des eingestellten Katalogs oder auf Schemata eines anderen Katalogs müssen dann qualifizierte Objektnamen verwenden.

7.2.3 Syntaxunterschiede SQL-Dialekte - DRIVE

Dieser Abschnitt listet die wesentlichen Funktionen und Syntaxelemente auf, die DRIVE zusätzlich zu oder abweichend von den jeweiligen SQL-Sprachbeschreibungen des DB-Servers unterstützt.

- Variablen

  Der Begriff der Variablen in DRIVE steht für eine Benutzervariable, d.h. für eine Variable, die der Programmierer im Deklarationsteil des Programms selbst definiert.
  Die DRIVE-Metavariable *varname* bezeichnet eine Benutzervariable. Diese muss das Präfix „&“ haben.
  Der Begriff Indikatorvariable bezeichnet in DRIVE-Anweisungen eine Variable, die zusammen mit der USING-Klausel bei Parameterübergaben in andere Programmiersprachen benötigt wird (siehe „DRIVE-Programmiersprache“ [2], Kapitel „Variablen und Konstanten einsetzen“). Der Indikatorwert gibt an, ob der zugehörige Parameterwert als NULL-Wert zu interpretieren ist oder nicht.
  Bei SESAM ist die Semantik der Indikatorvariablen erweitert (siehe „DRIVE-SQL-Lexikon“ [4]). Innerhalb von SQL-Anweisungen werden in DRIVE keine Indikatorvariablen benötigt. Vielmehr beinhalten DRIVE-Variablen stets auch NULL-Wertanzeigen.

  Mit den Klauseln LIKE TABLE und LIKE CURSOR der DECLARE VARIABLE-Anweisung können Sie komfortabel eine strukturierte Variable &varname definieren, deren Struktur der angegebenen Tabelle (Basistabelle, Systemtabelle, View) oder dem angegebenen Cursor (Ergebnistabelle) entspricht. Innerhalb von SQL-Anweisungen kann dann durch die Angabe von &varname.* komfortabel eine Variablenliste spezifiziert werden, die der Spaltenliste der Tabelle/des Cursors entspricht.

  Beispiele

  ```
  INSERT INTO TABLE tabelle VALUES (&varname.*);

  FETCH cursor INTO &varname.*;
  ```

- Dialog- und Programm-Modus

  Bei den Anweisungen FETCH und SELECT besteht in DRIVE ein Unterschied zwischen Dialog- und Programm-Modus:

Im Dialog-Modus werden die Werte von Satzelementen am Bildschirm ausgegeben. Die INTO-Klausel ist nicht erlaubt. Die INTO-Klausel muss im Programm-Modus angegeben werden. Die Werte von Satzelementen werden mit der INTO-Klausel in Variablen übertragen.

- Transaktionssicherung

  DRIVE bietet für die Anweisung ROLLBACK WORK die Klausel WITH RESET an. Die einfache ROLLBACK-Anweisung setzt alle Datenbank-Anweisungen in der aktuellen Transaktion zurück. Mit der WITH RESET-Klausel wird zusätzlich die DRIVE-Transaktion zurückgesetzt. D.h. die Inhalte von Variablen und Formaten werden auf den letzten COMMIT-Stand zurückgesetzt, und der Programmlauf wird nach der letzten COMMIT WORK-Anweisung fortgesetzt (siehe „DRIVE-Programmiersprache“ [2], Kapitel „Transaktionskopnzept“ und bei Verteilung das entsprechende Kapitel “Verteilte Transaktionsverarbeitung (VTV)“).

- DRIVE-Anweisungen und -Klauseln für Cursor

  DRIVE bietet eine DECLARE CURSOR-Anweisung an, die gegenüber der SQL-Norm erweitert ist und zu unterschiedlichen Cursortypen führt:

Cursortyp	**DECLARE-Anweisung**
permanenter Cursor	mit PERMANENT
temporärer Cursor	mit TEMPORARY (Vorbelegung)
SCROLL-Cursor	mit SCROLL
Schub-Cursor	mit PREFETCH n
UPDATE-Cursor	mit FOR UPDATE
statischer Cursor	im Deklarationsteil eines DRIVE-Programms
dynamischer Cursor	im Dialog-Modus oder mit einer EXECUTE-Anweisung zum Ablaufzeitpunkt
variabler Cursor	im Deklarationsteil eines DRIVE-Programms ohne FOR-Klausel und zum Ablaufzeitpunkt mit einer EXECUTE-Anweisung um eine dynamische FOR-Klausel ergänzt

  – Mit der DRIVE-Anweisung CYCLE *cursorname* INTO *variable* können Sie komfortabel in einer Cursorschleife Datenbanksätze in eine Variablenliste einlesen (siehe „DRIVE-SQL-Lexikon“ [4]).

  – Schubmodus

    Die PREFETCH-Klausel für Cursorverarbeitung im Schubmodus aus der SESAM V1 gilt mit derselben Semantik bei SESAM V2. Darüberhinaus bietet DRIVE bei SESAM V2 die Möglichkeit, den Schubmodus über eine PRAGMA-Anweisung zu aktivieren (siehe Seite 144).

- Zum variablen Cursor siehe „DRIVE-SQL-Lexikon“ [4]. Zum Gültigkeitsbereich des variablen Cursor siehe „DRIVE-Programmiersprache“ [2], Abschnitt „Auswirkung der Transaktionssteuerung auf Definitionen“.
- Die DRIVE-Anweisungen DROP CURSOR *cursor* und DROP CURSORS löschen Cursor. Die Anweisungen können im Dialog-Modus von DRIVE oder dynamisch, d.h. zum Ablaufzeitpunkt, erzeugt und ausgeführt werden. D.h. mit DROP können Dialog-Cursor, variable Cursor und dynamische Cursor gelöscht werden.
- Im Programm-Modus bleibt ein Cursor, der mit PERMANENT definiert wurde, über das Programmende hinaus mit seiner Cursorposition erhalten, wenn das Programm mit CALL aufgerufen wurde. Ein Cursor, der mit TEMPORARY definiert wurde, wird bei Programmende geschlossen, die Cursorposition geht verloren.
- In einer DRIVE-Sitzung können im Dialog- und Programm-Modus insgesamt maximal 20 nicht-statische SESAM Cursor angelegt werden.

- NULL-Wert-Darstellung

  DRIVE bietet mit der Anweisung PARAMETER DYNAMIC NULL die Möglichkeit, für Ausgaben die Darstellung von **NULL-Werten** abweichend vom Standard zu wählen (bei Bildschirmformaten das Sonderzeichen @, bei Druckerausgaben den Punkt „.“, siehe „DRIVE-Lexikon“ [3], Anweisung PARAMETER DYNAMIC). Die mit PARAMETER DYNAMIC vereinbarte NULL-Wert-Darstellung bei Ausgaben kann für DRIVE-Bildschirmformate durch Angabe einer anderen NULL-Wert-Darstellung in der NULL-Klausel der DECLARE FORM-Anweisung überschrieben werden. Wird die Darstellung des NULL-Wertes nicht mit der PARAMETER- oder DECLARE FORM-Anweisung vereinbart, gilt die Standard-Voreinstellung.

- Gültigkeitsbereich von Cursorn

  Ein Dialog-Cursor ist gültig zwischen den Anweisungen DECLARE und DROP. Er ist in keinem Programm gültig.

  Ein statischer oder variabler Programm-Cursor ist in dem Programm gültig, in dem er deklariert wurde: Im Deklarationsteil ab seiner DECLARE-Anweisung, im Ablaufteil (Body) und in jedem internen Unterprogramm.

  Ein dynamischer Programm-Cursor ist zum Ablaufzeitpunkt des Programms gültig für alle Anweisungen, die nach seiner Deklaration durchlaufen werden. Variable Programm-Cursor verhalten sich gegenüber dem DB-Server wie dynamische.

  Siehe auch „DRIVE-Programmiersprache“ [2], Kapitel „Transaktionskonzept“.

7.2.4 Lebensdauer von Cursorn

Die Lebensdauer eines Dialog-Cursor reicht von seiner DECLARE-Anweisung bis zum Löschen durch den Anwender (DROP) oder DRIVE am Ende der DRIVE-Sitzung.

Die Lebensdauer eines statischen Programm-Cursor beginnt zum Übersetzungszeitpunkt bei seiner DECLARE-Anweisung und zum Ablaufzeitpunkt beim Aufruf des Programms, in dem er deklariert wird, mit DO oder CALL.

Die Lebensdauer eines variablen Programm-Cursor beginnt bei DRIVE mit der statischen Deklaration seines Namens, beim DB-Server mit der dynamischen Deklaration seiner FOR-Klausel.

Bei einem reinem Übersetzungslauf (Anweisung COMPILE) endet die Lebensdauer aller Programm-Cursor mit der Übersetzung. Bei einer Übersetzung mit anschließender Ausführung (Anweisungen DO, DEBUG, CALL) bleiben die Cursor zum Ablaufzeitpunkt erhalten.

Temporäre statische, dynamische oder variable Programm-Cursor werden von DRIVE bei Transaktionsende (COMMIT WORK) auf der nächst höheren Programmstufe gelöscht oder spätestens beim Übergang in den Dialog-Modus. Permanente Programm-Cursor werden von DRIVE beim Übergang in den Dialog-Modus gelöscht, falls das Programm mit CALL aufgerufen wurde und im rufenden Programm kein COMMIT oder ROLLBACK WORK erfolgt. D.h. permanente offene Cursor behalten bei Programmende ihre Position unverändert bei. Dynamische und variable Cursor können auch anwenderseitig gelöscht werden (siehe DROP CURSOR-Anweisung im „DRIVE-SQL-Lexikon“ [4]).

Zur Lebensdauer von Cursorn siehe auch „DRIVE-Programmiersprache“ [2], Abschnitt „Auswirkung der Transaktionssteuerung auf Definitionen“).

7.2.4.1 Cursorposition

Die Cursorposition ist ein Verweis vor, auf oder hinter genau einen Satz der Cursor-Ergebnistabelle. Sie ist nur während der Lebensdauer des Cursor definiert und wird durch folgende Anweisungen beeinflusst:

Anweisung	**Cursorposition nach erfolgreicher Anweisungsausführung**
BREAK CYCLE	unspezifiziert
CLOSE	unspezifiziert
CYCLE	auf den nächsten Satz oder unspezifiziert
DECLARE CURSOR	unspezifiziert
DELETE ...WHERE CURRENT OF...	vor den nächsten Satz oder hinter den letzten Satz
DROP CURSOR(S)	undefiniert
END CYCLE	unspezifiziert

Anweisung	Cursorposition nach erfolgreicher Anweisungs-ausführung
FETCH	auf den positionierten Satz
OPEN	vor den ersten Satz
RESTORE	nach den zum letzten FETCH gehörigen Satz
STORE	vor den nächsten oder nach den letzten Satz und abgespeichert (kein FETCH mehr möglich)
UPDATE...WHERE CURRENT OF...	auf den zum letzten FETCH gehörigen Satz

DRIVE schließt offene temporäre statische Programm-Cursor am Ende des Programms, in dem sie deklariert wurden, oder bei Programmabbruch. Permanente offene Cursor hingegen bleiben offen und behalten daher ihre Cursorposition bei. Die Auswirkungen von Transaktionsanweisungen auf die Lebensdauer und Position eines Cursor ist in „DRIVE-Programmiersprache“ [2], Kapitel „Transaktionskonzept“ beschrieben.

7.2.5 Zugriffsschutz

DRIVE unterstützt die Schutzmechanismen des Datenbanksystems SESAM V2 und stellt Ihnen Anweisungen zur Verfügung, mit denen Zugriffsberechtigungen vergeben oder erbracht werden.

Es gibt folgende DRIVE-Anweisungen:

Anweisung	Funktion
PERMIT	Angabe von Benutzeridentifikationen, die beim Old-Style-Zugriff auf CALL-DML-Tabellen überprüft werden; hat nur Auswirkungen auf Old-Style
GRANT	Vergabe von Zugriffsrechten für Datenbanken, Basistabellen oder Views sowie für Storage Groups; hat nur Auswirkungen auf New-Style
SET SESSION AUTHORIZATION	Aktuellen Berechtigungsschlüssel festlegen für dynamische SQL-Anweisungen eines DRIVE-Programms; hat nur Auswirkungen auf New-Style
PAR DYN AUTHORIZATION	Aktuellen Berechtigungsschlüssel festlegen für SQL-Anweisungen einer DRIVE-Session

Die Transaktionssicherung, die auch den Zugriffsschutz berührt, ist in „DRIVE-Programmiersprache“ [2], Kapitel „Transaktionskonzept“ ausführlich beschrieben.

8 Anhang

8.1 Jahr-2000-Unterstützung bei DRIVE-Oldstyle

DRIVE bietet in der Oldstyle-Variante folgende Funktionen und Datumsvariablen an:

Sprache	Schnittstelle	4-stellige Jahres-Darstellung möglich	Ersatzlösung
DRIVE-Oldstyle	$GDATE	nein	Programm SF2GDAT4 verwenden
	$IDATE	nein	Programm SF2IDAT4 verwenden
	&DATE	nein	Programm SF2DATE verwenden

DRIVE bietet in der Oldstyle-Variante die Programme SF2GDAT4 und SF2IDAT4 zur Konvertierung von 2-stelligen in 4-stellige Jahresangaben sowie das Programm SF2DATE zur Information über das aktuelle Tagesdatum mit 4-stelliger Jahreszahl an.

Programmname	Parameter	Funktion
SF2GDAT4	P1, P2, P3	Umwandlung des P2-Werts aus einer Standard-Datumsangabe mit 2-stelliger Jahresangabe (JJMMTT) in die Form TT.MM.JJJJ, abhängig von P3-Wert.
SF2IDAT4	P1, P2, P3	Umwandlung des P2-Werts aus einem deutschen Datumsformat mit 2-stelliger Jahresangabe (TT.MM.JJ) in die Form JJJJMMTT, abhängig vom P3-Wert.
SF2DATE	P1	Rückmeldung des aktuellen Tagesdatums in der Form JJJJMMTT.

8.1.1 SF2GDAT4 - Jahreszahlenkonversion

Das Programm SF2GDAT4 wandelt ein durch *parameter-2* angegebenes Datum vom Standard-Datumsformat mit 2-stelliger Jahreszahl (JJMMTT) in ein deutsches Datumsformat mit 4-stelliger Jahreszahl (TT.MM.JJJJ) um.

Das Jahrhundert wird dabei durch den in *parameter-3* angegebenen Wert nach folgendem Algorithmus bestimmt ('sliding window'):

- Letztes Jahr des 100-Jahr Intervalls:
  J1J2J3J4 = aktuelles Jahr (4-stellig) + Wert in *parameter-3*

  Ist z.B. das aktuelle Jahr 1999 und ist der Wert in *parameter-3* gleich 30, dann ergibt dies ein 100-Jahr-Intervall von 1930 bis 2029.
- JJJJ bei der Rückgabe von *parameter-1* ergibt sich aus:

  100 * J1J2 + JJ falls J3J4 ≥ JJ
  100 * (J1J2 - 1) + JJ falls J3J4 < JJ

Beispiele siehe unten.

```
CALL SF2GDAT4 USING &par1=&par1, &par2, &par3;
```

&par1 Rückgabe des über *&par2* eingegebenen Datums mit 4-stelliger Jahreszahl in der (deutschen) Form TT.MM.JJJJ, wobei TT und MM entsprechend der MMTT-Angabe in *&par2* sind.

&par1 muss vom Typ alphanumerische Variable mit Länge 10 Byte sein.

Im Fehlerfall wird in *&par1* '0' zurückgeliefert.

&par2 Angabe eines Datums in der Form (JJMMTT).

&par2 muss vom Typ alphanumerische Variable oder Literal mit Länge 6 Byte sein. Der Wert von *&par2* muss eine positive ganze Zahl kleiner als 1000000 sein. Es wird nicht geprüft, ob Wert von *&par2* ein gültiges Datum ist.

&par3 Angabe eines Wertes, wie weit das 100-Jahres-Intervall vom aktuellen Datum aus in die Zukunft reichen soll.

&par3 muss vom Typ numerische Variable oder Literal mit Länge ≤ 4 Byte sein. Der Wert von *&par3* muss eine ganze Zahl sein. Die Summe aus dem aktuellen Jahr und dem Wert von *&par3* muss kleiner als 10000 sein.

Beispiel

```
PROC;
CRE VAR &AKTDAT   CHAR(8);
CRE VAR &AKTJAHR  NUM(2);
CRE VAR &DATUM6   CHAR(6);
CRE VAR &DATUM10  CHAR(10);
CRE VAR &DELTA    NUM(4);

/*  Berechnung des Datums mit gleitendem Fenster :                          */
/*                                                                          */
/* Das 100-Jahr Intervall, in das das ber. Jahr fällt, soll die Jahre       */
/* (aktuelles Jahr - 49) bis (aktuelles Jahr + 50) umfassen                 */

SET &DATUM6  = '721201';
SET &DELTA =  50;
...
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (1)
...

/* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll die      */
/* Jahre (aktuell. Jahr - 109) bis (aktuelles Jahr - 10) umfassen          */

SET &DATUM6  = '921201';
SET &DELTA =  -10;

CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (2)

/*  Berechnung des Datums mit festem Fenster :                              */
/*                                                                          */
/* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll die      */
/* Jahre 1950 - 2049 umfassen.                                              */
/* Berechnung des letzten Jahres des 100-Jahr Intervalls relativ zum       */
/* aktuellen Jahr.                                                          */

CALL SF2DATE USING &AKTDAT=&AKTDAT; /* Aktuelles Tagesdatum mit 4-st. Jahr */
SET &AKTJAHR = &AKTDAT(P=1,L=4);
SET &DELTA  = 2049 - &AKTJAHR;
SET &DATUM6   = '501201';

CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (3)
...
SET &DATUM6  = '011201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (4)

...
```

```
/*                                                                    */
/* Das 100-Jahr Intervall, in das das berechnete Jahr fällt, soll die */
/* Jahre 1890 - 1989 umfassen.                                        */
/* Berechnung des letzten Jahres des 100-Jahr Intervalls relativ zum  */
/* aktuellen Jahr.                                                    */

CALL SF2DATE USING &AKTDAT=&AKTDAT; /* Aktuelles Tagesdatum mit 4-st. Jahr */
SET &AKTJAHR = &AKTDAT(P=1,L=4);
SET &DELTA  = 1989 - &AKTJAHR;
SET &DATUM6   = '891201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                (5)
...
SET &DATUM6  = '901201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                (6)
```

Im Jahr 1997 liefert das Programm folgende Ergebnisse:

```
(1)  01.12.1972
(2)  01.12.1892
(3)  01.12.1950
(4)  01.12.2001
(5)  01.12.1989
(6)  01.12.1890
```

Im Jahr 2050 liefert das Programm folgende Ergebnisse :

```
(1)  01.12.2072
(2)  01.12.1992
(3)  01.12.1950
(4)  01.12.2001
(5)  01.12.1989
(6)  01.12.1890
```

8.1.2 SF2IDAT4 - Jahreszahlenkonversion

Das Programm SF2IDAT4 wandelt ein durch *parameter-2* angegebenes Datum im deutschen Format mit 2-stelliger Jahreszahl (TT.MM.JJ) in ein Standard-Datum mit 4-stelliger Jahreszahl (JJJJMMTT) um.

Das Jahrhundert wird dabei durch den in *parameter-3* angegebenen Wert nach folgendem Algorithmus bestimmt ('sliding window'):

- Letztes Jahr des 100-Jahr Intervalls:
  J1J2J3J4 = aktuelles Jahr (4-stellig) + Wert in *parameter-3*

  Ist z.B. das aktuelle Jahr 1999 und ist der Wert in *parameter-3* gleich 30, dann ergibt dies ein 100-Jahr-Intervall von 1930 bis 2029.

- JJJJ bei der Rückgabe von *parameter-1* ergibt sich aus:

  100 * J1J2 + JJ falls J3J4 ≥ JJ
  100 * (J1J2 - 1) + JJ falls J3J4 < JJ

Beispiele siehe unten.

```
CALL SFIGDAT4 USING &par1=&par1, &par2, &par3;
```

&par1 Rückgabe des über *&par2* eingegebenen Datums mit 4-stelliger Jahreszahl in der Form JJJJMMTT, wobei MM und TT entsprechend der MMTT-Angabe in *&par2* sind.

&par1 muss vom Typ alphanumerische Variable mit Länge 8 Byte sein.

Im Fehlerfall wird in *&par1* '0' zurückgeliefert.

&par2 Angabe eines Datums in der Form TT.MM.JJ.

&par2 muss vom Typ alphanumerische Variable oder Literal mit Länge 8 Byte sein. Der Wert von *&par2* muss eine Datumsangabe in Form TT.MM.JJ sein. Es wird nicht geprüft, ob Wert von *&par2* ein gültiges Datum ist.

&par3 Angabe eines Wertes, wie weit das 100-Jahres-Intervall vom aktuellen Datum aus in die Zukunft reichen soll.

&par3 muss vom Typ numerische Variable oder Literal mit Länge ≤ 4 Byte sein. Der Wert von *&par3* muss eine ganze Zahl sein. Die Summe aus dem aktuellen Jahr und dem Wert von *&par3* muss kleiner als 10000 sein.

Beispiel

```
PROC;
CRE VAR &DDAT8   CHAR(8);
CRE VAR &DATUM8  CHAR(8);
CRE VAR &DELTA   NUM(4);
...

SET &DDAT8 = '01.12.59';
SET &DELTA =  50;
...
CALL SF2IDAT4 USING &DATUM8=&DATUM8, &DDAT8, &DELTA;                    (1)
...
SET &DDAT8 = '01.12.47'
SET &DELTA =  -50;
...
CALL SF2IDAT4 USING &DATUM8=&DATUM8, &DDAT8, &DELTA;                    (2)
...
```

Im Jahr 1997 liefert das Programm folgende Ergebnisse:

```
(1)  19591201
(2)  18471201
```

Im Jahr 2009 liefert das Programm folgende Ergebnisse:

```
(1)  20591201
(2)  19471201
```

Ausführlichere Beispiele sind beim Programm SFGDAT4 zu finden.

8.1.3 SF2DATE - Aktuelles Tagesdatum

Das Programm SF2DATE liefert das aktuelle Tagesdatum in Form JJJJMMTT.

```
CALL SF2DATE USING &par1=&par1;
```

&par1 Rückmeldung des aktuellen Tagesdatums mit 4-stelliger Jahreszahl in der Form JJJJMMTT.

&par1 muss vom Typ alphanumerische Variable mit Länge 8 Byte sein.

Beispiel

```
PROC;
CRE VAR &AKTDAT CHAR(8);
...

CALL SF2DATE USING &AKTDAT=&AKTDAT;
```

Am 1.12.2000 liefert das Programm folgendes Ergebnis: `20001201`

8.2 DRIVE-Webanschluss

Mt Hilfe der Produktlinie Web*Transactions* lassen sich Daten von bestehenden DRIVE-Anwendungen im Internet präsentieren. Dadurch sind alle damit erstellbaren Daten, aus SESAM, aus LEASY und aus BS2000-Files über Standard-Web-Browser auch übers Internet verfügbar.

Aufsetzend auf den automatischen Generierungen der HTML-Seiten kann die grafische Verbesserung der Oberflächen nach und nach erfolgen.

Bestehendes Look & Feel kann beibehalten oder bei Bedarf auch beliebig parallel zu neuen Dialogführungstechniken angeboten werden. Alle Oberflächen werden zentral auf dem Server verwaltet, sind also bei Änderungen sofort wieder aktuell für alle Clients verfügbar.

Web*Transactions* - der WWW-Zugang auch für DRIVE-Server-Anwendungen

Beim Einsatz des Browsers besteht freie Auswahl. Ob Netscape, Microsoft Internet Explorer oder Mosaic - Web*Transactions* arbeitet mit allen.

Die Web-Server Software (HTTP-Daemon) bietet in Verbindung mit Web*Transactions* eine Kommunikationsverbindung zu Server-Anwendungen. Dabei kann Web*Transactions* neben den statischen HTML-Dokumenten auch Informationen die sich ändern oder interaktiv, d.h. abhängig von Benutzereingaben ermittelt werden, also dynamisch sind, erzeugen.

Damit gelingt es auf einfache Weise die Benutzeroberflächen der Anwendungen dynamisch in HTML-Formate umzuwandeln, die dann den Browsern zur Ausgabe übergeben werden. Dabei muss die Anwendungslogik der Server-Anwendung nicht geändert werden, denn Web*Transactions* stellt für die Anwendung nur einen zusätzlichen User dar. Das hat den Vorteil, dass trotz Einführung neuer WWW-Oberflächen, die Anwendung parallel dazu auch noch mit alphanumerischen Oberflächen bedienbar bleibt.

Darüber hinaus können Sie mit Web*Transactions* - wenn für die neue Darstellung gewünscht - die Präsentationslogik Ihrer DRIVE-Anwendung ändern; d.h. Sie können Masken zusammenfassen, trennen oder auch überspringen.

Web*Transactions* ist u.a. auch auf den Systemplattformen BS2000/OSD verfügbar und kann über das CGI-Interface an den ab BS2000/OSD V2 standardmäßig integrierten HTTP-Dämon angeschlossen werden.

Weitere Informationen zu Web*Transactions* finden Sie über
`http://www.siemens.de/webta.html`

8.2.1 Vorgehensweise

Für die Umstellung einer DRIVE-Anwendung auf Web-Betrieb sind folgende Schritte notwendig:

1. Maskenentwurf mittels IFG, bzw. Verwenden bestehender Masken
2. DRIVE-Anwendung erstellen bzw. bestehende DRIVE-Anwendung auf Web-Betrieb umstellen. Dabei beachten Sie bitte:
   - Bildschirmausgaben sind nur über FHS-EUA-Masken erlaubt (=#-Formate)
   - das Beenden der DRIVE-Anwendung muss mit einem STOP WITH DISPLAY SCREEN erfolgen.
   - Es wird empfohlen, möglichst alle Ausnahmefälle per WHENEVER abzufangen und die Fehlermeldung über eine Fehlermaske auszugeben werden. Nicht abgefangene Ereignisse werden von DRIVE über die DRIVE-Standard-HTML-Seite WEBERR.htm quittiert.

   Näheres finden Sie in Abschnitt „Einsatzvorbereitung und -hinweise“ auf Seite 212.
3. Umsetzen der FHS-Formate in entsprechende HTML-Dokumente

   Zur Umsetzung der einzelnen Formate sind Standarddienste im Web*Transactions* verfügbar. Aus einem FHS-Format sind über IFG eine C-Adressierungshilfe und ein Listenausdruck in Dateiform zu erstellen. Beides dient dann als Input für das Web*Transactions*-Programm **ifg2fld**. Mit diesem werden dann die Nettodatenbeschreibung (.fld-Datei) und eine erste HTML-Seite (.htm-Datei) erzeugt.

   Eine genauere Beschreibung finden Sie im Handbuch „Web*Transactions* V3.0 - Anschluss an *open*UTM-Anwendungen über UPIC“ [16].
4. Wahlweise kann die HTML-Seite nachbearbeitet werden

   Das von fhs2fld (bis Web*Transactions* V2.0) bzw. Web*Lab* erzeugte HTML-Template kann über einen beliebigen HTML-Editor weiterverarbeitet werden und dieser Output dann als Input für Web*Transactions* dienen.
5. Installieren des HTTP-Servers und von Web*Transactions*

   Web*Transactions* kann
   - entweder auf einem SINIX- oder Windows NT-Rechner betrieben werden, die Kopplung zur UTM-Anwendung erfolgt dann über UPIC und UTM-D (BS2000),
   - oder Web*Transactions* wird im POSIX-Subsystem des BS2000 installiert.

   Näheres siehe z.B. Handbuch „Web*Transactions* V3.0 - Anschluss an *open*UTM-Anwendungen über UPIC“ [16].

6. Browser-Installation

   Für die Browser-Einstellung gelten folgende Tipps:

   – für solche Dialoganwendungen, und die UTM-Anwendung ist eine Dialoganwendung, die i.d.R. feste Dialogabläufe voraussetzt, sollten Cache-Einstellungen vermeiden werden, da der Browser dann ggf. alte Seiten anzeigt.

   – genauso sollte vermieden werden, über Proxy-Rechner zu gehen, da auch hier Cache-Mechanismen generiert sind und ggf. bereits vom Proxy alte Seiteninhalte transferiert werden.

8.2.2 Einsatzvorbereitung und -hinweise

In der BS2000-Server-Anwendung sind im Web-Betrieb kein gleichzeitiger Interpreter- und Objekt-Einsatz in der gleichen UTM-Anwendung möglich: Objekt- und Interpreterbetrieb sind strikt zu trennen.

UTM-Versionsabhängigkeiten

Voraussetzung für die Server-Anwendung ist im BS2000 die Version ab V3.4 von UTM-D. Versionen < V3.4 sind nicht Web*Transactions*-fähig.

Für DRIVE-Interpreter und DRIVE-Compiler sind folgende Maßnahmen notwendig:

- DRIVE-Interpreter

Thema	**Angabe ab UTM V3.4**	**ab UTM V4.0 auch möglich**
First-TAC	DRIWEB	<beliebig>
Folge-TAC	WEBNEXT	SQLNEXT
Benutzerkennsatz (J/N)	Ja, DRIWEB@@@@@@@@@@	Ja
Programmname der ersten gerufenen Prozedur	beliebiger Prozedurname	beliebiger Prozedurname
Fehlerverhalten	Ausgabe einer Fehlermaske; bzw. Standardfehlermaske	keinerlei Fehlerbehandlung, es wird nicht erkannt, dass es sich um WebTA-Betrieb handelt.

- DRIVE-Compiler

Thema	**Angabe ab UTM V3.4**	**ab UTM V4.0 auch möglich**
First -TAC	DRIWEB	First-Objekt-TAC
Folge-TAC	DRT#W###	Folge-Objekt-TAC
Benutzerkennsatz (J/N)	Nein	Nein

Thema	**Angabe ab UTM V3.4**	**ab UTM V4.0 auch möglich**
Programmname der ersten gerufenen Prozedur	DRIWEB (zwingend !!!!)	
Fehlerbehandlung	Ausgabe einer Fehlermaske; bzw. Standardfehlermaske	keinerlei Fehlerbehandlung, es wird nicht erkannt, dass es sich um WebTA-Betrieb handelt.

Interpreterbetrieb

Folgende Zulieferungen bzw. Zuweisungen sind im Interpreterbetrieb mit Web-Anschluss notwendig:

– in der DRIVE-Bibliothek: X-Element SMO.ERROR

– in der DRIVE-Bibliothek: S-Element DRIWEB@@@@@@@@@@ (nur bei First-TAC DRIWEB notwendig)

– in der FHS-Bibliothek: R-Element WEBERR

Die DRIVE-Bibliothek ist die per PAR DYN zugewiesene Bibliothek. Die genannten Bibliothekselemente werden in der SIPLIB.DRIVE.022 ausgeliefert.

Im Objektbetrieb

Folgende Zulieferungen bzw. Zuweisungen sind im Objektbetrieb mit Web-Anschluss notwendig:

– in Anwendungen für Objektbetrieb ist der R-Modul SMO#ROR@ mit einzubinden

– in der FHS-Bibliothek: R-Element WEBERR

Die genannten Bibliothekselemente werden in der SIPLIB.DRIVE.022 ausgeliefert.

KDCDEF-Datei und Anwendungsgenerierung

Für Interpreter und Compiler werden KDCDEF-Rahmendateien für den Web-Betrieb ausgeliefert.

– Interpreter: Bibliothek SYSPRC.DRIVE.022, Element DRIKDCDEF.WEB

– Compiler: Bibliothek SYSPRC.DRIVE-COMP-LZS.022, Element DRCKDCDEF.WEB

Hinweis: Im WebTA-Betrieb sind natürlich keine festen Benutzernamen (PTERM-Namen) sinnvoll, deshalb gilt hier die Empfehlung, mit einem Terminalpool (TPOOL-Anweisung) zu arbeiten.

DRIVE-Standard-HTML-Template für Exceptions

Wenn in einer DRIVE-Anwendung zum Ablaufzeitpunkt Exceptions auftreten, die nicht über das Exception-Handling (WHENEVER-Anweisung) abgefangen werden, gibt das DRIVE-Laufzeitsystem im Web-Betrieb die mit ausgelieferte HTML-Seite WEBERR.htm aus. In dieser Seite wird dann der Exceptiongrund genannt (z.B. Conversion Error bei fehlerhaften Datenzuweisungen).

Diese mit der DRIVE-Korrektur ausgelieferten Daten sind in die entsprechenden Verzeichnisse von Web*Transactions* wie folgt zu übernehmen:

– **WEBERR.htm** nach ...../*webanwendungsname*/`config/improved` bzw. in das Verzeichnis, in das alle Web-Seiten abzulegen sind (je nach Web*Transactions* -Installation).

– **WEBERR.fld** nach ...../*webanwendungsname*/`config`.

Fehlerbehandlung in BS2000 Server-Anwendung

In der Server-Umgebung sind in die DRIVE-Bibliothek das Zwischencode- bzw. im Objektbetrieb das X- bzw. R-Element einzuspielen:

– SMO.ERROR: X-Element in der DRIVE-Bibliothek bei Interpreterbetrieb
– SMO#ROR@: R-Element bei Objektbetrieb

In die FHS-Bibliothek ist das Format WEBERR (R-Element) einzuspielen.

i Eine Fehlerbehandlung ist nur möglich, wenn der First-TAC DRIWEB ist und damit die DRIVE-Anwendung WebTA-Betrieb erkennt!!!!!

Im Fehlerfall werden dann o.g. Elemente dazu verwendet, an Web*Transactions* eine Fehlermeldung zurückzusenden und die Server-Anwendung ordnungsgemäß zu beenden.

Bisher führen lediglich ein USER- oder SYSTEMRAISE im DRIVE-Server zu einem Abbruch in Web*Transactions*.

Ab der UTM-Version V4.0 ist es möglich, mit beliebigen TACs an die Server-Anwendung zu gehen.Für diese TACs (Compiler oder Interpreter) ist es nicht möglich, eine WebTA-fähige Fehlerbehandlung zu machen, weil WebTA-Betrieb nicht erkannt wird! Beispielsweise führen Ausgaben im LINE-Modus (dynamische Formate), Rücksetzen von Transaktionen (mit Ausgabe einer Prompting-Meldung) zum Abbruch der Client-Anwendung und auch der Server-Anwendung.

Es empfiehlt sich daher, auch ab UTM V4.0 als First-TAC DRIWEB zu verwenden (sowohl Interpreter als auch Compiler).

8.2.3 Beispiel für Generierung einer BS2000-Server-Anwendung

KDCDEF: Objektbetrieb

```
OPTION GEN=ALL
ROOT  DRTWEB
MAX APPLINAME=DRIWEB, APPLIMODE=SECURE, KDCFILE=DRIWEB
FORMSYS ENTRY=KDCFHS,LIB=SYSLIB.DRIVE.022, TYPE=FHS
REM
REM
REM ************************************************************************
REM ***                                                                  **
REM ***                    KDCDEF-RAHMEN FUER DRIVE V2.2                 **
REM ***                         NEWSTYLE-VARIANTE                        **
REM ***                              FUER WEB                            **
REM ***                    ******* OBJEKT-BETRIEB ********               **
REM ************************************************************************
REM
REM ******** MAXIMALWERTE EINSTELLEN *******************************
REM
MAX KB=1024,SPAB=32767,NB=32700
MAX TASKS=6,ASYNTASKS=5
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(10000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10
MAX LOGACKWAIT=600,BRETRYNR=10,VGMSIZE=128
REM
REM ******** STEUERANWEISUNGEN FUER DEN KDCDEF-LAUF ************
REM
REM ******** KDCROOT DEFINIEREN ********************************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=SYS.MOD.SQL22
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=SYS.MOD.SQL22
REM
MPOOL <mempool>,SIZE=200,LIB=SYSLNK.DRIVE.022,SHARETAB=<sharetab>
REM
REM ******** TEILPROGRAMME DEFINIEREN **************************
REM
PROGRAM KDCADM,COMP=ILCS
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM DRTROOT,COMP=ILCS
PROGRAM DRTVORG,COMP=ILCS
REM
REM ******** USER-EXITS DEFINIEREN *****************************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
```

```
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM ****************** DRIVE-MODULE LADEN **********************
REM
MODULE EXTAB,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=STATIC
MODULE EXSTART,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRTMAT20,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=(POOL,<mempool>)
MODULE DRI#ERS@,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=(POOL,<mempool>)
REM
REM ****************** DRIVE-MODULE FUER OBJEKTBETRIEB LADEN ***
REM
MODULE DRIWEB@@,LIB=<useroml>,LOAD=(POOL,<mempool>)
MODULE <userspezif. DRIVE-Objekt>,LIB=<useroml>,LOAD=(POOL,<mempool>)
.
.
MODULE <userspezif. DRIVE-Objekt>,LIB=<useroml>,LOAD=(POOL,<mempool>)
MODULE SMO#ROR@,LIB=SIPLIB.DRIVE.022,LOAD=(POOL,<mempool>)
REM
REM ******** TRANSAKTIONSCODES FUER DRIVE **********************
REM
DEFAULT TAC PROGRAM=DRIVROOT
REM
TAC DRISQL   ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRTVORG
TAC DRISQLF  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT  ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC SQLENTER ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
TAC SQLLIST  ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
REM
REM ******** TRANSAKTIONSCODES FUER COMPILER ******************
REM
DEFAULT TAC PROGRAM=DRTROOT
REM
TAC DRTXSCXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXSCAX ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
TAC DRT#I### ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRT#O### ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXCRXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXCLXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
REM
REM ******** TRANSAKTIONSCODES FUER OBJEKTE ********************
REM
TAC DRIWEB ,CALL=FIRST,TYPE=D,EXIT=DRTVORG,TIME=300000
TAC <userspezuf. Tacname>,CALL=NEXT,TYPE=D,TIME=300000
.
TAC <userspezuf. Tacname>,CALL=NEXT,TYPE=D,TIME=300000
TAC SMO#ROR,CALL=NEXT,TYPE=D,TIME=300000
```

```
REM
REM ******** SYNCHRONE ADMINISTRATION ***************************
REM
DEFAULT TAC PROGRAM=KDCADM
TAC KDCTAC,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCLTERM,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCPTERM,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSWTCH,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCUSER,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSEND,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCAPPL,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCDIAG,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCLOG,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCINF,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCHELP,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSHUT,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCTCL,TYPE=D,STATUS=ON,CALL=BOTH,DBKEY=UTM,ADMIN=Y
REM
REM ******** ASYNCHRONE ADMINISTRATION ************************
REM
TAC KDCTACA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCLTRMA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCPTRMA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSWCHA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCUSERA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSENDA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCAPPLA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCDIAGA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCLOGA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCINFA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCHELPA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSHUTA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTCLA,TYPE=A,STATUS=ON,CALL=FIRST,DBKEY=UTM,ADMIN=Y
REM
REM ******** BELEGUNG DER FUNKTIONSTASTEN **********************
REM
SFUNC K1,RET=2OZ                          "BREAK-TASTE"
SFUNC F1,CMD=KDCOFF
SFUNC K2,CMD=KDCOUT
REM
REM ******** ANBINDUNG AN SINIX-RECHNER                  **
REM ******** PARTNERNAME                                 **
BCAMAPPL  <bcamname>,T-PROT=ISO
REM
TPOOL  BCAMAPPL=<bcamname>,    -
              LTERM=LTRM,NUMBER=50,PRONAM=<rechnername>,PTYPE=UPIC-R
REM
END
```

Literatur

[1] **DRIVE/WINDOWS V2.1** (BS2000)
Programmiersystem
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Einführung in das Programmiersystem DRIVE/WINDOWS und Erläuterung der Funktionen des Dialog-Modus' sowie Beschreibung der Installation, der Generierung und der Administration von DRIVE/WINDOWS

[2] **DRIVE/WINDOWS V2.1** (BS2000)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Bildschirm- und Listenformaten und Reports. Beschreibung des Transaktionskonzepts und der Verteilten Transaktionsverarbeitung. Beispiele.

[3] **DRIVE/WINDOWS V2.1** (BS2000)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen, Meldungen und Schlüsselwörter von DRIVE/WINDOWS

[4] **DRIVE/WINDOWS V2.1** (BS2000)
Lexikon der DRIVE-SQL-Anweisungen für SESAM/SQL 2
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM V2.x in Kurzform.

[5] **DRIVE/WINDOWS-COMP V2.1** (BS2000)
Compiler
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer und Administratoren
Inhalt
Beschreibung der Abweichungen zwischen Interpreter und Compiler und Darstellung des Compilierungsvorganges. Beschreibung des Generierens und Startens von Anwendungen von compilierten DRIVE-Objekten (TIAM- und UTM-Betrieb) unter besonderer Berücksichtigung des Versionsmischbetriebes.

[6] **SESAM/SQL-SERVER V2.2A** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 1: SQL-Anweisungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle Personen, die eine SQL-Datenbank mit SQL-Anweisungen bearbeiten.
Inhalt
Das Handbuch beschreibt die Programmeinbettung von SQL-Anweisungen und die SQL-Sprachelemente. In einem alphabetischen Nachschlagteil sind alle SQL-Anweisungen ausführlich dargestellt.

[7] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 2: Utilities
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle Personen, die mit der Verwaltung einer SESAM/SQL-Datenbank befasst sind.
Inhalt
Das Handbuch enthält eine alphabetische Beschreibung der Utility-Anweisungen; Utility-Anweisungen sind Anweisungen in SQL-Syntax und realisieren die Dienstprogrammfunktionen von SESAM/SQL.

[8] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Basishandbuch
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an alle Anwender und alle, die sich über SESAM/SQL informieren wollen.
Inhalt
Das Handbuch gibt einen Überblick über das Datenbanksystem und beschreibt Grundlagen, Konzepte und Zusammenhänge. Es ist die Basis für das Verständnis der weiteren SESAM/SQL-Handbücher.

[9] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
SESAM/SQL-Server Utility-Monitor
Benutzerhandbuch

Zielgruppe
Das Handbuch ist für den Datenbankverwalterund den Systemverwalter von SESAM/SQL-Server bestimmt.
Inhalt
Das Handbuch beschreibt die Bedienung des Utility-Monitors. Mit dem Utility-Monitor können DB-Verwaltungs- und Administrationsaufgaben u.a. im maskengeführten Dialog ausgeführt werden.

[10] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Meldungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle SESAM/SQL-Anwender.
Inhalt
Das Handbuch enthält sämtliche Meldungen zu SESAM/SQl nach Meldungsnummern sortiert.

[11] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
CALL-DML-Anwendungen
Benutzerhandbuch

Zielgruppe
Das Handbuch richtet sich an alle CALL-DML-Anwendungen-Programmierer.
Inhalt
Es enthält die Beschreibung der CALL-DML-Schnittstelle mit den DML-Anweisungen und dazugehörigen Beispielen. Außerdem sind unter anderem Binden, Laden, Anwenden im Teilhaberbetrieb und die CALL-DML-Dienstprogramme beschrieben.

[12] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Performance
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an den erfahrenen Anwender von SESAM/SQL.
Inhalt
Das Handbuch beschreibt, wie man Performance-Engpässe im Leistungsverhalten von SESAM/SQL erkennt. Es enthält Maßnahmen, die geeignet sind, das Systemverhalten zu beeinflussen.

[13] ***open*UTM** (BS2000/OSD)
Anwendungen generieren und betreiben
Benutzerhandbuch

Zielgruppe
Anwendungsplaner, Organisatoren, Anwender und Betreuer von UTM-Anwendungen

[14] **FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Programmschnittstellen von FHS für TIAM-, DCAM- und UTM-Anwendungen. Erstellen, Einsatz und Verwalten von Formaten.

[15] **LMS** (BS2000)
ISP-Format
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der Anweisungen zum Erstellen und Verwalten von PLAM-Bibliotheken und darin enthaltenen Elementen.
Häufige Anwendungsfälle werden an Hand von Beispielen erklärt.

[16] **Web*Transactions***
Anschluss an *open*UTM-Anwendungen über UPIC
Benutzerhandbuch

Zielgruppe
Alle, die mit Web*Transactions* UTM-Dialoganwendungen an das Web anschließen wollen.
Inhalt
Das Handbuch beschreibt alle für den Web-Anschluss von UTM-Dialoganwendungen notwendigen Schritte. Das Handbuch ergänzt das einführende Handbuch „Konzepte und Funktionen“ und das Referenzhandbuch „Templatesprache“ um alle UTM-spezifischen Informationen.

[17] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

Zielgruppe
BS2000/OSD-Systemverwaltung
Inhalt
Das Handbuch beschreibt
- die Generierung der Hardware- und Software-Konfiguration mit UGEN
- die Installationsdienste
  - Plattenorganisation mit MPVS
  - Programmsystem SIR
  - Datenträgerinstallation mit SIR
  - Configuration Update (CONFUPD)
  - Dienstprogramm IOCFCOPY.

Wenden Sie sich zum Bestellen von Handbüchern bitte an Ihre zuständige Geschäftsstelle.

Stichwörter

T

Inhalt

DRIVE V2.2

Ergänzungsband

Zielgruppe
Programmierer und Administratoren von DRIVE-Anwendungen im BS2000.

Inhalt

Dieses Handbuch beschreibt die neuen und geänderten Funktionen zu DRIVE V2.2:
– Einsatzhinweise
– Neue und geänderte DRIVE-SQL-Anweisungen
– Geänderte DRIVE-Anweisungen
– Programmierhinweise
– Datenbank-Unterstützung
– Jahr-2000-Unterstützung bei DRIVE-Oldstyle
– Web-Abschluss für DRIVE

Ausgabe: Januar 2000

Datei: drive_22.pdf

Fujitsu Siemens Computers GmbH
Handbuchredaktion
81730 München

Kritik
Anregungen
Korrekturen

Fax: (0 89) 6 36-4 04 43

e-mail: DOCetc@mchp.siemens.de
http://manuals.mchp.siemens.de

Absender

Kommentar zu DRIVE V2.2
Ergänzungsband

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.